А.В. ПРУЦКОВ

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ JAVA

ВВЕДЕНИЕ В КУРС С ПРИМЕРАМИ И ПРАКТИЧЕСКИМИ ЗАДАНИЯМИ

УЧЕБНИК

Рекомендовано
Научно-методическим советом "РГРТУ"
в качестве учебника для студентов высших учебных заведений,
обучающихся по направлению подготовки 2.09.03.04
«Программная инженерия» (квалификация "бакалавр"),
2.09.05.01 «Применение и эксплуатация автоматизированных систем
специального назначения» (квалификация "специалист")

Москва
КУРС
2018

УДК 004.42(075.8)
ББК 32.973.26-018.1я73
П85

ФЗ № 436-ФЗ	Издание не подлежит маркировке в соответствии с п. 1 ч. 4 ст. 11

Рецензент:
кафедра «Вычислительная и прикладная математика» Рязанского государственного радиотехнического университета (зав. кафедрой — д-р техн. наук, проф. *А.Н. Пылькин*)

Пруцков А.В.

П85 Программирование на языке Java. Введение в курс с примерами и практическими заданиями : учебник / А.В. Пруцков. — М.: КУРС, 2018. — 208 с.

ISBN 978-5-906923-51-6 (КУРС)

Учебник включает разделы, посвященные основам объектно-ориентированного программирования, элементам языка Java, его стандартным классам и библиотекам. Разделы содержат более 50 примеров. Отдельная глава содержит семь заданий, которые позволят закрепить учебный материал на практике.

В учебник вошли материалы лекционных и практических занятий дисциплин, связанных с изучением объектно-ориентированного программирования и языка Java и преподаваемых автором в высших учебных заведениях.

Предназначен для студентов технических направлений, в учебный план которых включены дисциплины «Объектно-ориентированное программирование», «Язык программирования Java», а также для всех желающих изучить язык программирования Java не только в теории, но и на практике.

УДК 004.42(075.8)
ББК 32.973.26-018.1я73

ISBN 978-5-906923-51-6 (КУРС)

КРАТКОЕ ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Назначение учебника

Этот учебник предназначен для читателей, владеющих основами программирования и желающих получить начальные знания программирования на языке Java. Выполнив практические работы на основе теоретических и справочных материалов, приведенных в учебнике, вы изучите язык Java и научитесь писать программы на этом языке.

Учебник включает лекционные материалы и задания практических работ курса, который автор преподавал студентам технических специальностей и направлений, связанных с программированием, в высших учебных заведениях: 2.09.03.04 «Программная инженерия» и 2.09.05.01 «Применение и эксплуатация автоматизированных систем специального назначения». Номера специальностей и направлений приводятся такими, какими они были на момент преподавания курса. За время преподавания автором были изданы учебные пособия [23, 24] и в соавторстве методические указания [9], материалы которых включены в этот учебник. Также в учебнике использовались материалы двухтомного учебника по информатике и программированию [10, 11]. Методика преподавания курса апробировалась на научно-методических конференциях [21].

Структура и содержание учебника

Учебник состоит из четырех частей:

1) теоретических и справочных материалов по основам языка Java с примерами;

2) практических работ и заданий к ним;

3) дополнительных материалов для преподавателей;

4) библиографического списка, использованного при написании учебника.

Теоретические и справочные материалы учебника поделены на разделы. Каждый раздел неделим и посвящен одной теме. Такое деление объясняется желанием автора дать возможность читателям, еще не освоившим язык программирования Java, использовать учебник в качестве справочного пособия.

В главе 1 «Основы языка программирования Java» излагаются основные понятия объектно-ориентированного программирования

и языка Java: классы и объекты. Классы включают поля, конструкторы и методы. Метод main является точкой входа в программу. Все классы в языке Java являются прямыми или косвенными подклассами класса Object и наследуют его методы. Для структуризации программы используются пакеты, в которые помещаются классы с общей функциональностью или решающие одну или схожие задачи. Приведен пример использования классов и объектов. В этой и последующих главах для стандартных классов используются короткие имена, так как это не вызывает неоднозначности.

Глава 2 «Элементы языка программирования Java» посвящена изложению базовых понятий языка Java, без которых невозможно начать писать программы на этом языке: базовых типов переменных, массивов, строк, управляющих конструкций и порядка ввода и вывода на консоль. Все типы в языке Java делятся на базовые и ссылочные. *Базовые* типы (целые, вещественные, символьный, логический) используются для определения полей классов. *Ссылочные* типы представлены в главе строками и массивами. Строки в языке Java — это объекты классов String, StringBuilder или StringBuffer. Массивы могут содержать элементы базовых и ссылочных типов. Операции сортировки, поиска, копирования, заполнения и сравнения собраны в классе-утилите Arrays. Язык Java имеет те же управляющие конструкции, как и большинство языков структурного программирования: ветвления и циклы.

В главе 3 «Объектно-ориентированное программирование» продолжается и расширяется изложение базовых понятий объектно-ориентированного программирования, начатых в главе 1. Принципы этого типа программирования (инкапсуляция, наследование и полиморфизм) позволяют сократить дублирование строк программы и тем самым сделать программу легко расширяемой и изменяемой. Классам и их элементам можно придать различные свойства, сделав их абстрактными или статическими. Чтобы задать общее поведение классов, описывающих сущности из разных предметных областей, используются интерфейсы. Перечисления позволяют задать не только совокупность значений объекта, но и каждому значению задать свое собственное поведение. Глава завершается примером проектирования иерархии классов с использованием рассмотренных понятий.

В главе 4 «Стандартные классы и библиотеки» представлены наиболее часто используемые стандартные классы и библиотеки языка Java. Язык Java предоставляет возможность разработки многоязычных интерфейсов программ. Чтение и запись в различные типы данных (файлы, массивы, строки) осуществляется единообразно с помощью потоков ввода и вывода данных. Ошибки при работе с потоками ввода или вывода, а также другие ошибки при выполнении программы обрабатываются с помощью исключений. Кроме массивов, для хранения и обработки данных используются коллекции.

Почти каждый раздел снабжен примерами, поясняющими описываемые понятия. Обозначение примера включает номер главы и через дефис — порядковый номер примера в этой главе. Текст программ в примерах оформлен в соответствии с соглашениями Java Code Conventions [41].

Знание английского языка важно при программировании, так как проекты по разработке программного обеспечения ведутся международными группами программистов. Поэтому все названия элементов программ и комментарии в примерах приведены на английском языке.

Читателю (и студентам) предлагается закрепить теоретический материал на практике. Для этого в учебник включена глава 5 «Практические работы». Практические работы охватывают все темы разделов учебника: строки и массивы; классы и их иерархии; потоки ввода и вывода; коллекции. Практические работы имеют различные варианты заданий, рассчитанные на одну студенческую группу (15–20 человек).

Существуют различные среды разработки программ на языке Java. Однако предполагается, что студенты будут использовать среду Eclipse. Освоить навыки работы в ней можно с помощью главы 6 «Разработка программ в среде Eclipse».

На лекционных занятиях использовались специально подобранные примеры для пояснения понятий языка Java и объектно-ориентированного программирования. Примеры приведены в главе 7 «Примеры программ для проведения лекционных занятий».

В главе 8 «Дополнительные материалы» приводятся методические рекомендации по проведению лекционных и практических занятий на основе этого учебника.

Книги, учебные материалы и информационные ресурсы глобальной сети Интернет из библиографического списка использовались в качестве вспомогательных источников информации для написания теоретических разделов учебника и для разработки вариантов заданий к практическим работам. Библиографический список упорядочен в алфавитном порядке.

Принципы, на которых построен учебник

Перечислим основные принципы, которые положены в основу учебника.

1. Последовательность изложения.

Понятия объясняются на основе уже известных понятий. При написании разделов материал излагался максимально последовательно, без перескоков от одного понятия к другому. Однако в нескольких случаях приходилось упоминать о каком-то понятии, без которого невозможно последующее изложение, а затем раскрывать его более полно в следующих разделах. Этот недостаток присущ многим книгам, посвященным языку Java и другим объектно-ориентированным языкам программирования.

2. Минимум строк программ в примерах. Программы должны быть объемом не более одной страницы.

Примеры программ в учебнике упрощены и сокращены, чтобы сосредоточить их на конкретной теме. Программы объемом более одной страницы сложно понимать. Поэтому такие программы в учебнике отсутствуют.

3. Минимум требуемого теоретического материала.

Теоретический материал включает необходимые сведения для выполнения практических работ.

Обозначения, используемые в учебнике

В учебнике используются следующие выделения шрифтами и специальный символ.

Сжатый шрифт без засечек — листинг программ в примерах и теоретическом материале, заголовки методов в описании классов и интерфейсов.

`Моноширинный шрифт с засечками` — текст, выводимый программой на экран или консоль.

Символ ❑ — конец примера. Отделяет пример от последующего текста.

Об авторе

Александр Викторович Пруцков — доктор технических наук, профессор кафедры «Вычислительная и прикладная математика» Рязанского государственного радиотехнического университета — на протяжении более 10 лет читает курсы по программированию и информатике, активно сотрудничает с российскими и международными компаниями из отрасли информационных технологий.

Заключение

Автор готов ответить на вопросы и дать консультации преподавателям, желающим использовать материалы учебника для постановки новых или обновления существующих курсов. Связаться с автором по всем интересующим вопросам можно через его интернет-ресурс http://prutzkow.com. С этого же интернет-ресурса можно загрузить примеры, приведенные в учебнике, в электронном виде.

1. ОСНОВЫ ЯЗЫКА ПРОГРАММИРОВАНИЯ JAVA

1.1. Классы как основа языка Java

1.1.1. Классы и их назначение

Классы и все, что есть в языке Java, предназначены только для того, чтобы решить задачу, написав как можно меньше строк программы.

Любая простая или сложная система выполняет совокупность действий, используя для этого необходимые исходные данные. Действия системы могут состоять из последовательности более простых действий.

Пример 1-01. Чтобы выдать текущий баланс банковского счета, банкомат должен выполнить следующие действия:

1) запросить ПИН-код у пользователя;

2) отослать ПИН-код и номер банковской карты по зашифрованному каналу в банк;

3) принять данные о балансе или сообщение об ошибке по зашифрованному каналу от банка;

4) выдать данные пользователю. ❑

В программе, написанной на объектно-ориентированном языке программирования, как простые, так и составные действия, а также данные, необходимые для их выполнения, представляются в виде классов и объектов. На основе понятия класса построен весь язык Java.

Класс определяет параметры состояния и поведение совокупности элементов предметной области. Параметры состояния хранятся в полях класса. Поведение определяется методами класса, доступными другим классам.

При проектировании класса следует учитывать, что класс должен быть предназначен для решения одной задачи.

Классы не существуют изолированно, а взаимодействуют друг с другом. Логически связанные классы объединяются в пакеты. Имя класса в пакете должно быть уникальным. С точки зрения файловой системы пакет представляет собой папку, в которой помещаются программные файлы, содержащие реализацию классов.

Существуют несколько типов взаимодействия классов между собой. Одним из типов взаимодействия является наследование.

Класс А называется подклассом класса Б, если класс А является непосредственным наследником класса Б. В этом случае класс Б называется суперклассом класса А.

Класс является шаблоном элементов предметной области. Непосредственно в программе используются конкретные экземпляры класса — *объекты*. Классы можно рассматривать как типы данных (с операциями по их обработке), а объекты — как переменные этих типов.

Структура класса

Класс состоит из полей, конструкторов и методов.

```
[public] [final] [abstract] class <имя класса> [extends <имя суперкласса>] [implements <имена интерфейсов>] {

    // Поля
    [<спецификаторы>] <тип поля> <имя поля>;

    // Конструкторы
    [<спецификаторы>] <имя класса> ([<параметры>]);

    // Методы
    [<спецификаторы>] <возвращаемый тип> <имя метода> ([<параметры>]) {
    <операторы>
    [return <значение>]
    }
}
```

Составные части, заключенные в квадратные скобки, являются необязательными и могут отсутствовать.

Поля описывают состояние объекта, по каким параметрам объект может изменяться.

Конструкторы выполняются при создании объекта. Основной задачей конструктора является присваивание начальных значений полям (инициализация значений полей).

Методы описывают поведение класса, т.е. какие действия объект может производить над значениями полей или с параметрами этих методов.

Строка конструктора и метода, содержащая спецификаторы, имя класса и параметры для конструктора и тип возвращаемого значения, имя и параметры для метода, называются *заголовком*. Остальная часть конструктора или метода называется *телом*.

Поля, конструкторы и методы могут следовать в описании класса в любом порядке. Однако согласно п. 3.1.3 соглашений по оформлению программы (Java Code Conventions) рекомендуется придерживаться следующего порядка: поля, методы, конструкторы.

1.1.2. Спецификаторы

Спецификаторы определяют правила изменения полей, вызова и переопределения методов классов, доступа к ним из других классов.

Действие спецификатора распространяется только на тот элемент класса, перед которым спецификатор помещен.

В языке Java имеются следующие спецификаторы типов класса и их элементов:

1) abstract (у классов и методов). Абстрактные классы используются как суперклассы в некоторой предметной области. Абстрактные классы могут иметь абстрактные методы. У абстрактных методов в классе определен только заголовок, но отсутствует тело. Абстрактные методы определяют поведение подклассов;

2) final (у классов, методов, полей, переменных). Класс с этим спецификатором не может иметь подклассы. Метод не может быть переопределен в подклассах. Поле или переменная не может менять свое значение после инициализации. Спецификатор final используется для объявления констант;

3) static (у методов и полей). Статические поля и методы являются общими для всех объектов класса. Для обращения к статическим полям и вызова методов не нужно создавать объект класса.

Спецификаторы доступа к полям, конструкторам и методам:

1) public: элементы класса доступны для всех классов в этом и других пакетах; используется для методов, реализующих поведение класса;

2) protected: элементы класса доступны классам, находящимся в том же пакете, и подклассам в других пакетах;

3) без спецификатора доступа (спецификатор доступа по умолчанию; называется также friendly или package-private): элементы класса доступны только подклассам и классам, находящимся в том же пакете;

4) private: элементы класса доступны только объекту данного класса; используется, чтобы запретить прямой доступ к полям из других классов и разрешить чтение и запись значений полей только через специальные методы (инкапсуляция).

Область видимости элементов класса зависит от спецификатора доступа и представлена в следующей таблице.

Спецификатор	private	Без спецификатора доступа	protected	public
Тот же класс	Да	Да	Да	Да
Подкласс в том же пакете	Нет	Да	Да	Да
Независимый класс в том же пакете	Нет	Да	Да	Да
Подкласс в другом пакете	Нет	Нет	Да	Да
Независимый класс в другом пакете	Нет	Нет	Нет	Да

1.1.3. Объекты класса

Класс представляет собой общее описание параметров и поведение некоторой сущности предметной области. В программе используются объекты — конкретные экземпляры класса. У объектов установлены значения полей, и их методы доступны для вызова.

Чтобы создать объект класса, необходимо выполнить два действия:

1) объявить переменную типа класса, которая будет хранить ссылку на ячейку памяти с содержимым объекта;

2) вызвать конструктор класса для создания объекта с помощью оператора new.

Создание объекта в два действия имеет общий вид

```
<имя класса> <имя объекта>;
<имя объекта> = new <имя класса>([<параметры>]);
```

Однако эти шаги можно объединить в одну команду:

```
<имя класса> <имя объекта> = new <имя класса>([<параметры>]);
```

Оператор new выделяет область памяти под объект. Переменная типа класса представляет собой ссылку на эту область памяти.

Чтобы обратиться к полям и методам объекта класса, необходимо записать имя объекта, точку и далее записать имя поля или метода:

```
<имя объекта>.<имя поля>;
<имя объекта>.<имя метода>([<параметры метода>]);
```

Обращение к полям и методам возможно только после создания объекта класса (кроме статических полей и методов).

1.1.4. Пример классов, объектов и их использования

Пример 1-02. Класс NumberOperator позволяет прибавлять и вычитать целые числа из аргумента, который хранится в поле number (строка 5 фрагмента программы).

Второе поле ZERO имеет значение 0 (строка 3) и является неизменяемым, константным, так как имеет спецификатор final.

Поле number может быть изменено только объектом этого класса (имеет спецификатор private). Поле ZERO доступно всем объектам любых классов без создания объекта класса NumberOperator, так как оно является статическим (имеет спецификатор static).

Класс имеет два конструктора. В конструкторах задается начальное значение поля number.

Первый конструктор (строки 7–9) не имеет параметров. Полю number присваивается значение ноль с помощью константы ZERO (строка 8).

Второй конструктор (строки 11–13) имеет один параметр. Значение параметра присваивается полю number (строка 12).

Значение поля number можно изменить с помощью метода setNumber (строки 19–21). Однако этот метод недоступен для вызова другими классами (имеет спецификатор private). Поэтому задать значение поля number можно только при вызове конструктора. Метод не возвращает результат работы, поэтому тип его возвращаемого значения обозначен как void.

Получить значение поля number можно с помощью метода getNumber (строки 15–17). Этот метод имеет спецификатор public и поэтому доступен для вызова любыми классами. Значение поля возвращается с помощью оператора return.

Также класс NumberOperator имеет три метода, выполняющих операции сложения и вычитания.

Операция сложения выполняется двумя методами sum (строки 23–25 и 27–29). Эти методы имеют одинаковые имена, однако отличаются типами параметров. В первом методе параметр имеет целый тип int. Во втором методе параметр имеет вещественный тип double. В теле этого метода используется явное приведение типа double к типу int (строка 28). В зависимости от типа параметра будет вызываться один из этих методов.

Метод subtract (строки 31–33) реализует операцию вычитания с параметром целого типа.

```
1  public class NumberOperator {
2
3      public static final int ZERO = 0;
4
5      private int number;
6
7      public NumberOperator() {
8          setNumber(ZERO);
9      }
10
11     public NumberOperator(int number) {
12         setNumber(number);
13     }
14
15     public int getNumber() {
16         return this.number;
17     }
18
19     private final void setNumber(int number) {
20         this.number = number;
21     }
22
23     public int sum(int item) {
24         return this.number + item;
25     }
26
27     public int sum(double item) {
28         return this.number + (int) item;
29     }
```

```
30
31      public int subtract(int subtrahend) {
32      return this.number - subtrahend;
33      }
34  }
```

Класс NumberOperatorRunner имеет метод main (строки 3–12), с которого будет начинаться выполнение программы.

В методе main создается объект класса NumberOperator (строки 9–10), вызываются методы этого объекта sum с параметром типа double (строка 12) и subtract (строка 13), формируется строка с результатом (строка 15) и выводится на экран (строка 17).

```
1   public class NumberOperatorRunner {
2
3       public static void main(String[] args) {
4
5           int sourceNumber = 5;
6           double argForSum = 7.5;
7           int argForSubtract = 2;
8
9           NumberOperator number =
10            new NumberOperator(sourceNumber);
11
12          int sum = number.sum(argForSum);
13          int difference = number.subtract(argForSubtract);
14
15          String result = "Sum:" + sum + "\nDifference:" + difference;
16
17          System.out.println(result);
18      }
19  }
```

На экран будет выведено:

Вывод на экран
Sum: 12 Difference: 3

❑

1.1.5. Поля

Поля класса используются для описания параметров сущностей предметной области или группировки данных под общим именем. Совокупность значений полей характеризует состояние объекта.

Поля могут иметь тип этого или другого класса и тем самым являться ссылками на объект. Присвоить ссылку на объект полю можно с помощью оператора new.

В большинстве случаев прямой доступ к полям из других классов запрещен с помощью спецификатора private. Доступ к полям осуществляется только с помощью методов этого же класса.

1.1.6. Конструкторы

Конструктор — это особый метод, который вызывается при выполнении оператора new, создающего объект. Поэтому в заголовке конструктора отсутствует тип возвращаемого значения.

Конструктор имеет то же имя, что и класс.

Задачей конструктора является инициализация полей создаваемого объекта, т.е. задание начального состояния объекта. Не стоит в конструкторе решать другие задачи, так как они могут прервать выполнение конструктора, и объект не будет создан.

В классе может быть несколько конструкторов. Все они должны отличаться списком параметров.

При вызове в операторе new параметры конструктора должны совпадать с параметрами заголовка конструктора в объявлении класса по числу и типам.

Каждый класс должен иметь конструктор по умолчанию. Конструктор по умолчанию не имеет параметров. Если в объявлении класса отсутствует конструктор по умолчанию, но у суперкласса есть такой конструктор, то конструктор по умолчанию будет создан у класса автоматически. Если у суперкласса определен только конструктор с параметрами, то конструктор по умолчанию становится недоступным для подклассов и для его вызова необходимо явное объявление такого конструктора.

Деструкторы — методы для уничтожения объектов в языке Java отсутствуют. Объекты уничтожаются автоматически, если при выполнении программы не останется ни одной ссылки на этот объект.

1.1.7. Методы

1.1.7.1. Назначение методов

Метод — основной элемент структурированности программы. Каждый метод должен решать только одну задачу.

Если в суперклассе и подклассе методы с одним и тем же заголовком имеют разные реализации, то такие методы в подклассе называются *переопределенными*.

В классе может быть несколько методов с одинаковым именем. Методы должны отличаться друг от друга списком своих параметров. Такие методы называются *перегруженными*.

Если метод имеет тип возвращаемого значения void, то метод не возвращает значение. Если у метода тип возвращаемого значения не является void, то его тело должно содержать не менее одного оператора return.

Как и у конструктора, параметры вызова метода должны совпадать с параметрами заголовка по числу и по типам.

Методы делятся на два типа: команды и запросы. *Команды* не возвращают значения и предназначены для изменения состояния объекта. Примером команды является метод установки значений полей set. *Запросы* возвращают значение, учитывая или не учитывая состояние объекта. Примером запроса является метод получения значений полей get.

1.1.7.2. Методы для получения и установки значений полей

В соответствии с принципом инкапсуляции объектно-ориентированного программирования с каждым полем <имя>, значение которого должно изменяться другими объектами, связаны методы get<имя> и set<имя>. Например, если поле называется field, то методы, связанные с этим полем, принято называть getField и setField. Методы get<имя> и set<имя> добавляются к классу для того, чтобы создать единую точку изменения или получения значений поля только с помощью этих методов.

Доступ к полю через специальные методы сокращает изменение строк программ при изменении порядка присвоения или получения значения поля. Например, при присвоении значения поля необходимо проверить допустимость его значения или при получении значения поля его необходимо скорректировать. Обращение к полю напрямую потребует каждый раз проверять или корректировать значение. При использовании методов set<имя> и get<имя> все эти операции будут описаны в методах. При изменении проверки или

корректировки потребуется только изменить методы, а не искать в программе все обращения к полю и изменять их.

Пример 1-03. Пусть имеется класс Article, описывающий научную статью. Любая статья характеризуется именем автора (author), заголовком (header) и текстом самой статьи (text) (строки 3–5).

При печати статья разбивается на страницы. Страницы могут быть разного размера и могут содержать разное число строк текста, определяемое полем stringsPerPage.

Текст статьи разбивается на страницы специальными символами разрыва страницы. Поэтому getText не просто возвращает значение поля text, но и добавляет к нему разрывы страниц с помощью метода addPageBreaks (строка 10).

При присвоении полю text нового значения необходимо проверить, содержит ли текст символы (не является ли текст пустым) (строка 14). Только после этой проверки полю присваивается новое значение (строка 15). Если текст пуст, то вызывается обработчик ошибки (строки).

```
1  public class Article {
2
3      private String author;
4      private String header;
5      private String text;
6
7      private int stringsPerPage;
8
9      public String getText() {
10         return addPageBreaks(this.text, this.stringsPerPage);
11     }
12
13     public void setText(String newText) {
14         if (!newText.isEmpty()) {
15             this.text = newText;
16         } else {
17             // Error processing
18         }
19     }
20 }
```

Статью можно опубликовать несколькими способами: конвертировать в формат PDF или создать веб-страницы в формате HTML и выложить в сети Интернет или добавить к сборнику статей. Эти способы реализованы в классе ArticlePublisher.

Класс ArticlePublisher:

```
1  public class ArticlePublisher {
2
3      public PDFdoc convertToPDF(Article article) {
4          ...
5          // ! String articleText = article.text;
6          String articleText = article.getText();
7          ...
8      }
9
10     public HTMLpage convertToHTML(Article article) {
11         ...
12         // ! String[] pages = cutPerPages(article.text);
13         String[] pages = cutPerPages(article.getText());
14         ...
15     }
16
17     public void addToBook(Book book, Article article) {
18         ...
19         // ! book.addText(article.text);
20         book.addText(article.getText());
21         ...
22     }
23 }
```

При доступе к тексту статьи в трех методах класса ArticlePublisher использовался доступ к полю text. Предположим, что к тексту статьи необходимо добавить дату ее написания. В случае прямого доступа к полю text придется искать все обращения к этому полю (строки 5, 12, 19; отмечены как программные комментарии) в классе ArticlePublisher и добавлять к ним дату написания. В случае использования метода getText (комментарии в строках 6, 13, 20) изменять класс ArticlePublisher не придется. Необходимо лишь изменить класс Article в одном месте — в методе getText. ❑

Чтобы ограничить доступ к полям только через методы set<имя> и get<имя>, а не напрямую, используется спецификатор доступа private.

1.1.7.3. Метод main

Метод main является точкой входа в программу. Именно с него начинается выполнение консольного или настольного приложения.

Метод должен иметь заголовок:

```
public static void main(String[] args)
```

1.1.8. Ключевое слово this

Ключевое слово this представляет собой ссылку внутри объекта на сам себя. Слово this используется для доступа к полям, конструкторам и методам объекта.

Команда this() вызывает конструктор класса по умолчанию.

Пример 1-04. Метод setAField класса AClass присваивает полю aField новое значение (строки 5–7).

```
1  public class AClass {
2
3      private int aField;
4
5      public void setAField(int aField) {
6          this.aField = aField;
7      }
8  }
```

Слово this (строка 6) обозначает, что слева от операции присваивания (обозначается символом «=» (равно)) находится поле aField, а справа — параметр aField. ❑

Добавление слова this при обращении к элементам класса повышает читаемость программы.

1.1.9. Класс Object

1.1.9.1. Класс Object и его методы

Класс Object является родительским классом для всех классов языка Java. Класс Object имеет базовые методы, реализация которых наследуется всеми классами: стандартными (входящими в Java Development Kit (JDK)) и пользовательскими.

Основные методы класса Object:

Метод	Описание и общий вид
equals	Сравнивает исходный объект и объект anObject. boolean equals(Object anObject)
getClass	Возвращает объект типа Class, который содержит данные об элементах класса (полям, конструкторам и методам) и обеспечивает доступ к ним. Class getClass()
hashCode	Возвращает числовое значение (хэш-код), вычисленное по состоянию объекта. int hashCode()
toString	Возвращает представление объекта в виде строки типа String. Методу toString будет посвящен раздел 2.7.1.6. String toString()

Здесь и далее приведенные методы стандартных классов имеют спецификатор доступа public. Такие методы в учебнике называются методами класса.

1.1.9.2. Методы equals и hashCode для сравнения объектов

Сравнение объектов используется при их поиске и упорядочивании. При сравнении объектов используются методы equals и hashCode.

Метод hashCode возвращает *хэш-код* — числовое значение, полученное на основе состояния объекта. Хэш-код может быть вычислен, например, как сумма значений полей, определяющих состояние объекта.

Функция вычисления хэш-кода должна удовлетворять двум правилам.

1. Если объекты не равны, то и их хэш-коды должны быть не равны.

2. Если объекты равны, то и их хэш-коды должны быть равны.

Однако не во всех случаях можно быстро (в смысле временных затрат) построить такую функцию. Как правило, она удовлетворяет только второму правилу. Поэтому разные объекты могут иметь одинаковые хэш-коды.

Кроме того, метод hashCode возвращает целое число типа int размером 32 бита, имеющее $2^{32} = 4\ 294\ 967\ 296$ различных значений. Если количество различных объектов больше, то не все различные объекты будут иметь различные хэш-коды.

Поэтому если хэш-коды двух объектов равны, то их необходимо дополнительно сравнивать методом equals.

Сравнение объектов при поиске или упорядочивании методами equals и hashCode выполняется по следующему правилу.

1. Вычислить хэш-коды сравниваемых объектов.

2. Если хэш-коды объектов не равны, то объекты не равны, иначе сравнить объекты методом equals.

3. Если метод equals возвращает значение true, то объекты равны, иначе объекты не равны.

1.2. Пакеты

1.2.1. Понятие и назначение пакета

В реально работающих приложениях количество классов может доходить до нескольких тысяч. Для упорядоченного хранения классов используются пакеты. Пакеты могут иметь составные имена, образуя, таким образом, иерархическую структуру.

Разделение классов на пакеты необходимо, как минимум, в следующих случаях.

1. Упорядочивание структуры приложения. Близкие по функциональности или назначению классы помещаются в один пакет, что упрощает поиск классов и понимание структуры приложения.

2. Создание отдельного пространства имен. При распределенной разработке приложения каждый программист может не согласовывать имена классов из своих пакетов с другими разработчиками.

3. Спецификация доступа к элементам класса из классов из того же пакета. Все спецификаторы доступа, кроме спецификатора private, делают доступными элементы класса другим классам в этом же пакете (раздел 1.1.2), даже не являющимся подклассами.

Все файлы с объявлениями классов, содержащиеся в пакете, начинаются с директивы package. Общий вид директивы:

```
package <имя пакета>;
```

1.2.2. Короткие и полные имена. Область видимости

Каждый элемент программы на языке Java (класс, поле, метод, пакет) имеет имя. Именем является любая последовательность строчных и прописных букв английского алфавита, цифр и символов «_» (подчеркивание) и «$» (обозначение доллара). Имена не должны начинаться с цифры.

Полное имя элемента обязательно включает имя пакета и имя класса. Например, полное имя метода aMethod класса AClass, находящегося в пакете mypackage, будет следующим:

```
mypackage.AClass.aMethod().
```

Однако внутри пакета mypackage и внутри класса AClass можно использовать короткое имя aMethod.

Областью видимости объявления элемента языка Java называется фрагмент программы, в котором допускается обращение к этому элементу по короткому имени.

1.2.3. Правила именования пакетов

Каждый пакет должен иметь уникальное имя, чтобы исключить конфликт имен. Поэтому пакеты в языке Java именуются в соответствии со следующим соглашением.

Название пакета начинается с имени сайта разработчика, записанного, начиная с домена первого уровня, например com.site для сайта site.com. Далее разделенные точками могут следовать названия подразделений компании, фамилии разработчиков, имена компьютеров, названия проектов и т.д.

Примеры имен пакетов:

```
ru.rsreu.projectone.datatypes;
com.prutzkow.javamanual.example101.
```

Здесь rsreu.ru — адрес веб-сайта Рязанского государственного радиотехнического университета; prutzkow.com — адрес интернет-ресурса автора этого учебника.

1.2.4. Директива import

Если в программе используются классы, интерфейсы, их методы, поля или константы из других пакетов, то необходимо описывать их полными именами, включающими полное название пакета, класса, метода.

Однако такая запись становится громоздкой и затрудняет чтение строк программы. Чтобы перейти от полных имен к коротким, используется директива import, имеющая общий вид

```
import <имя пакета>.<имя класса или интерфейса>;
```

Если необходимо добавить все классы и интерфейсы пакета, то используется следующая форма директивы import, в которой вместо имени класса или интерфейса используется звездочка:

```
import <имя пакета>.*;
```

Директива добавляется в начало файла с объявлением класса, после директивы package.

Директиву import можно добавить вручную. Однако среды разработки (например, Eclipse) автоматически предлагают добавить эту директиву в случае обнаружения в объявлении класса элемента, находящегося в известном среде пакете.

1.3. Структура программы на языке Java

Программа на языке Java представляет собой совокупность пакетов. В каждый пакет помещаются несколько классов. Классы хранятся в файлах с расширением .java. В каждом файле необходимо размещать объявление только одного класса.

Скомпилированные файлы классов сохраняют свое имя, но имеют расширение .class.

Для настольных приложений один из классов должен иметь метод main, с выполнения которого начинается выполнение всей программы (раздел 1.1.7.3).

2. ЭЛЕМЕНТЫ ЯЗЫКА ПРОГРАММИРОВАНИЯ JAVA

2.1. Java Code Conventions

Соглашения по оформлению программ собраны в документе Java Code Conventions. Соглашения предложены компаниями «Sun» и «Oracle», в которых язык Java был создан и продолжает развиваться. Соблюдение этих соглашений позволит вам написать легко читаемые для других программы и самому читать программы, написанные другими программистами.

2.2. Правила именования элементов программы на языке Java

Имена для элементов программы на языке Java выбираются исходя из их предназначения в программе и записываются латиницей на английском языке. Например, maxValue, RestoreData.

Имена элементов зависят от регистра: name ≠ Name ≠ NAME.

В следующей таблице представлены правила именования элементов программы в соответствии с п. 9 соглашений Java Code Conventions.

Элемент	Размер букв	Первое слово	Второе и последующие слова	Пример
Класс, интерфейс	Строчные	С прописной буквы	С прописной буквы	Runner, ImageSpirit
Метод	Строчные	Со строчной буквы	С прописной буквы	draw(), getValue()
Переменная	Строчные	Со строчной буквы	С прописной буквы	height, cubeWidth
Переменная-константа	Прописные	С прописной буквы	С прописной буквы. Разделить слова знаком подчеркивания	PAGE_NUM, MAX_LIMIT
Пакет	Строчные	Со строчной буквы	Со строчной буквы	datatypes, interfaces

Имя метода обязательно должно начинаться с глагола, описывающего действие, которое он выполняет.

Имена интерфейсов обычно заканчиваются на -able, если имя заканчивается глаголом.

Для временных локальных переменных допускаются однобуквенные имена, как правило совпадающие с первой буквой типа.

Имя	Тип
b	byte
c	char
d	double
f	float
i, j, k	int
l	long
x, y	int (координаты)
o	Object
v	Некоторый тип

2.3. Комментарии в программе

В языке Java существуют три вида комментариев.

1. Однострочные комментарии:

```
// one-line comment
```

Однострочным комментарием считается строка, начинающаяся с двух наклонных черт и заканчивающаяся концом строки.

2. Многострочные комментарии:

```
/*
* multi-line comment
*/
```

Многострочным комментарием считаются строки, начинающиеся с символов /* и заканчивающиеся */.

3. Документация (Javadoc):

```
/**
* documentation
*/
```

Документацией считаются строки, начинающиеся с символов /** и заканчивающиеся */. Документация используется для текстового описания назначения классов, полей, конструкторов и методов, имеющих спецификатор public или protected. Для каждого такого метода также описывается назначение параметров и возвращаемого значения.

2.4. Типы переменных. Переменные и константы. Преобразования типов

2.4.1. Базовые и ссылочные типы

Все типы в языке Java можно разделить на базовые и ссылочные. Главным их отличием является то, что имя переменной *базового* типа хранит значение, а *ссылочного* типа хранит ссылку на объект в памяти с его значениями полей, конструкторами и методами. Получить значение переменной ссылочного типа по имени нельзя. Примерами базовых типов являются целые (byte, short, int, long), вещественные (float, double) и др. (будут рассмотрены в разделе 2.4.2). Примерами ссылочных типов являются массивы (будут рассмотрены в разделе 2.9) и классы (раздел 1.1).

2.4.2. Базовые типы

Базовые (примитивные) типы языка Java и их характеристики приведены в следующей таблице:

Название	Тип	Длина, бит	Диапазон значений	Значение по умолчанию
byte	Целый	8	−128 ... 127	0
short	Целый	16	−32768 ... 32767	0
int	Целый	32	$-2^{31} \dots 2^{31} - 1$	0
long	Целый	64	$-2^{63} \dots 2^{63} - 1$	0L
float	Вещественный с плавающей точкой	32	6–7 значащих десятичных цифр	0.0f
double	Вещественный с плавающей точкой	64	15 значащих десятичных цифр	0.0d
char	Символьный	16		'\u0000'
boolean	Логический	8	false, true	false

Все числовые типы в языке Java знаковые.

Тип char содержит символы в кодировке Unicode. В кодировке Unicode каждый символ кодируется 16 битами.

В отличие от других языков программирования в языке Java длина и диапазон одинаковы на всех аппаратных платформах.

2.4.3. Объявление и инициализация переменных

Объявление переменных может располагаться в любом месте программы и имеет вид

```
<тип> <имя переменной> [= <начальное значение>];
```

При объявлении переменной можно задать ее начальное значение (проинициализировать).

Область видимости переменной начинается со строки, в которой она объявлена, и заканчивается в конце программного блока, в котором это объявление расположено. *Программный блок* — это последовательность операторов, заключенная в фигурные скобки «{», «}» (см. раздел 2.8.3). Объявления переменных следует располагать в начале программного блока.

Переменные можно инициализировать значениями в других системах счисления.

Система счисления	Способ задания	Пример
Шестнадцатеричная	Начинается с 0x	0x34ac
Восьмеричная	Начинается с 0	062
Двоичная	Начинается с 0b	0b101100

Для вещественных значений можно явно указать тип: L в конце для типа long и f в конце для типа float.

Пример 2-01. Переменные $b, j, d1, f$ и x объявлены и проинициализированы, а переменные d и i — только объявлены.

```
1 | int i, j = 10000;
```

```
2  byte b = 0x10;
3  double d;
4  double d1 = 1e-10;
5  float f = 1.234f;
6  int x = j * 3;
7  char c = 'z';
```

Каждую переменную следует объявлять в отдельной строке (п. 6.1 соглашений Java Code Conventions). Объявление переменных в строке 1 сделано не по правилам, а в строках 3, 4 — по правилам.

Переменная *b* проинициализирована шестнадцатеричным значением 0x10 (строка 2).

Вещественные значения могут записываться в форме с фиксированной точкой (строка 5) или в экспоненциальной форме (строка 4). Если в строке 5 не указать в конце значения *f*, то компилятор воспримет как значение типа double и выдаст ошибку компиляции из-за невозможности преобразовать тип при присвоении.

Значение символьного типа заключается в апострофы (строка 7). ❑

2.4.4. Объявление и инициализация констант

В качестве констант в Java используют переменные со спецификатором final, который указывается перед типом переменной.

Пример 2-02. Константы могут инициализироваться при объявлении (строка 1) или операцией присваивания (строка 3).

```
1  final long CONST_LONG = 0x56677bbL;
2  final int CONST_INT;
3  CONST_INT = 10;
```

Попытка присвоения новых значений переменным CONST_LONG и CONST_INT приведет к ошибке компиляции. ❑

Использование констант вместо значений позволяет понять их роль в программе.

Пример 2-03. Пусть необходимо поменять местами значения первого и второго элементов последовательности sequence.

Строка программы без использования констант вызывает следующие вопросы:

```
1  changeValues(sequence, 1, 2);
```


Что означают эти числа? Могут ли эти значения быть заменены другими?

Использование констант снимает эти вопросы.

```
1  final int THE_FIRST = 1;
2  final int THE_SECOND = 2;
3
4  changeValues(sequence, THE_FIRST, THE_SECOND);
```

❑

2.4.5. Преобразования базовых типов

В арифметических выражениях базовые типы автоматически преобразуются от более коротких к более длинным:

byte ⇒ short ⇒ int ⇒ float ⇒ double.

Пример 2-04. Результат сложения переменной типа byte и переменной типа int будет иметь тип int (наибольший по длине), а переменных типов short и float — тип float. ❑

Такое преобразование называется *расширяющим*.

Результат арифметических операций целых типов byte и short всегда будет int.

При делении аргументов вещественных типов тип результата всегда будет вещественным, даже если дробная часть числа нулевая. Если результат деления присваивается переменной целого типа, то результат деления преобразуется к целому типу автоматически путем отбрасывания дробной части.

Расширяющее преобразование является неявным.

Сужающее преобразование от длинных типов к коротким возможно при явном указании типа.

Пример 2-05. Пусть переменной *i* типа int присвоено значение 0x12345678 в шестнадцатеричной системе счисления (строка 1).

```
1  int i = 0x12345678;
2  byte b = (byte) i;
3  System.out.println(Integer.toHexString(b));
4  byte b2 = (byte) (b + 10);
```

На экран будет выведено:

Вывод на экран
78

При присвоении значения переменной *i* переменной *b* типа byte задано явное преобразование к типу byte (строка 2). В результате в строке 3 на экран будет выведено число 78 в шестнадцатеричной системе счисления, которое хранилось в 8 младших (правых) битах (по размеру типа byte) переменной *i*. Метод Integer.toHexString(b) (строка 3) выводит на экран значение переменной *b* в шестнадцатеричной системе счисления.

Несмотря на то что переменные *b* и *b*2 имеют один и тот же тип byte (строки 2 и 4), при математических операциях требуется явное приведение к типу byte (строка 4), так как результат операции *b* + 10 имеет тип int. ❑

2.4.6. Классы-оболочки базовых типов

Для каждого из базовых типов (см. раздел 2.4.2) существует аналог ссылочного типа, называемый *классом-оболочкой*. Объект класса-оболочки «оборачивает» соответствующий базовый тип. Такая «обертка» необходима, например, для работы с коллекциями — наборами данных только ссылочных типов (списками, очередями и др.).

Соответствие между базовыми типами и классами-оболочками:

Базовый класс	Класс-оболочка
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Некоторые методы класса Integer, связанные с преобразованиями типов:

Метод	Описание и общий вид
parseInt	Преобразует строковое представление числа *s* в значение типа int в десятичной системе счисления. static int parseInt(String s) Преобразует строковое представление числа *s* в значение типа int в системе счисления radix. static int parseInt(String s, int radix)
toString	Преобразует значение объекта класса Integer в строку. static String toString() Преобразует значение переменной *i* в строку в десятичной системе счисления. static String toString(int i) Преобразует значение переменной *i* в системе счисления radix в строковое представление. static String toString(int i, int radix)
valueOf	Возвращает объект класса Integer, полученный из значения переменной *i*. static Integer valueOf(int i) Возвращает объект класса Integer, полученный из строкового представления числа *s*. static Integer valueOf(String s) Возвращает объект класса Integer, полученный из строкового представления числа *s* в системе счисления radix. static Integer valueOf(String s, int radix)

Другие классы-оболочки имеют аналогичные методы.

2.4.7. Константный тип

Константный тип является разновидностью ссылочного типа. Его особенностью является то, что после создания и инициализации объекта нельзя изменить значения его полей. Чтобы изменить объект, необходимо создать новый объект и поместить ссылку на него в переменную ссылочного типа. Ссылка на старый объект теряется.

Примерами константных типов являются классы-оболочки и строки String.

2.5. Операции

2.5.1. Операция присваивания

Операция присваивания в языке Java обозначается знаком «=» (равно).

Пример 2-06. Присваивание используется при инициализации переменных (строки 1–2) и копировании значения одной переменной другой переменной (строка 3).

```
1  int i = 3;
2  int j = 1500;
3  i = j;
```

❑

2.5.2. Арифметические операции

В языке Java определены следующие арифметические операции.

Операция	Обозначение	Пример
Сложение	+	a=b+c
Вычитание	-	a=b-c
Умножение	*	a=b*c
Деление (деление нацело для аргументов целых типов)	/	a=b/c
Остаток деления	%	a=b%c
Инкремент	++	a++
Декремент	--	a--

Если слева и справа от знака присваивания (равно) стоят одинаковые переменные, то можно использовать сокращенную форму записи.

Операция	Полная форма	Сокращенная форма
Сложение	a=a+b	a+=b
Вычитание	a=a-b	a-=b
Умножение	a=a*b	a*=b
Деление	a=a/b	a/=b
Остаток деления	a=a%b	a%=b

Пример 2-07. Следующий фрагмент выведет на экран строку «c=2» — целочисленный результат деления 7 на 3.

```
1  int a = 7;
2  int b = 3;
3  int c = a / b;
4  System.out.println ("c=" + c);
```

❑

2.5.3. Инкремент и декремент

Увеличение значения переменной на единицу можно записать несколькими способами:

```
i = i + 1;
i += 1;
i++;
++i;
```

Пример 2-08. Отличие способов, записанных в строках 4 и 5, заключается в следующем.

Запись в строке 4 означает: присвоить значение переменной *i* переменной *a*, а затем увеличить значение переменной *i* на 1.

Запись в строке 5 означает: увеличить значение переменной *j* на 1, а затем присвоить значение переменной *j* переменной *b*.

```
1  int i = 2;
2  int j = 2;
3
4  int a = i++;
5  int b = ++j;
6
7  System.out.println("a="+a+"; b="+b);
8  System.out.println("i="+i+"; j="+j);
```

Инкремент в строке 4 эквивалентен следующим присваиваниям:

```
1  a = i;
2  i = i + 1;
```

Инкремент в строке 5 эквивалентен следующим присваиваниям:

```
1  j = j + 1;
2  b = j;
```

На экран будет выведено:

Вывод на экран
a=2; b=3 i=3; j=3

Переменная $a = 2$ — значению переменной *i* до увеличения на 1, а переменная $b = 3$ — значению переменной *j* после увеличения на 1.

❑

Аналогично четырьмя способами можно записать декремент:

```
1 | i = i - 1;
2 | i -= 1;
3 | i--;
4 | --i;
```

2.5.4. Операции сравнения

Операции сравнения языка Java приведены в следующей таблице.

Операция	Полная форма
==	Равно
!=	Не равно
>	Больше
<	Меньше
>=	Больше или равно
<=	Меньше или равно

Операция сравнения == отличается для базовых и ссылочных типов. Базовые типы сравниваются этой операцией по значению, а ссылочные типы — по адресу памяти, в которой размещен объект. Поэтому для сравнения состояний объектов (переменных ссылочного типа) используется метод equals (см. раздел 1.1.9.2).

2.5.5. Логические операции

Аргументы и результаты логических операций имеют тип boolean.

Оператор	Результат	Пример	Результат примера
||	Дизъюнкция ИЛИ (OR)	(2 > 3) || (3 > 2)	true
&&	Конъюнкция И (AND)	(3 != 2) && (4 == 2)	false
!	Отрицание НЕ (NOT)	!(5 <= 6)	true

Таблица истинности логических операций

A	B	!A	A || B	A && B
false	false	true	false	false
true	false	false	true	false
false	true	true	true	false
true	true	false	true	true

2.5.6. Тернарная операция ?:

Тернарная операция ?: используется для получения значения в зависимости от условия и имеет общий вид

<условие>?<оператор 1>:<оператор 2>.

Если условие истинно, то выполняется оператор 1, иначе выполняется оператор 2.

Пример 2-09. Фрагмент программы с использованием тернарной операции:

```
1 | int a = 2;
2 | int b = 3;
3 | int c = 4;
4 | int d = (c > 2) ? a : b;
5 | System.out.println ("d=" + d);
```

На экран будет выведено:

Вывод на экран
d=2

Условие $c > 2$ истинно, поэтому переменной *d* будет присвоено значение переменной *a*, а именно значение 2. ❑

2.6. Ввод и вывод данных на консоль

2.6.1. Стандартные потоки ввода и вывода языка Java

Входные данные могут быть получены с клавиатуры, из файла, из телекоммуникационной сети или других источников. Выходные данные могут быть выведены на экран, записаны в файл или отправлены получателю по сети. Чтобы унифицировать работу с входными и выходными данными, в языке Java введено понятие потока. *Поток* (stream) — это единый способ получения входных данных из различных источников и записи выходных данных различным получателям (файлы, массивы и др.).

В Java определены три стандартных потока, работающие с консолью:

- System.in — поток ввода данных с консоли (клавиатуры);
- System.out — поток вывода данных на консоль (экран);
- System.err — поток вывода ошибок на консоль (экран).

Программист может создавать другие потоки, необходимые для решения задачи.

2.6.2. Вывод данных на консоль

Для вывода на консоль (экран) используются методы print и println объекта System.out. Их отличием является то, что метод println добавляет автоматически символ перевода строки \n, а метод print — нет.

При выводе данных на консоль можно использовать последовательности символов, заключенные в кавычки. Переменные базовых типов преобразуются в строки автоматически. Для переменных ссылочных типов неявно вызывается метод toString. Две и более строки соединяются операцией конкатенации, обозначаемой знаком «+» (плюс).

Пример 2-10. Вывод значения переменной.

```
1 | final int TOTAL_PAGE = 10;
2 | int currentPage = 5;
3 | System.out.println ("Page "+ currentPage + " of " + TOTAL_PAGE);
```

Вывод на экран
`Page 5 of 10`

❑

Для специальных символов используются следующие обозначения:

Специальный символ	Обозначение
Табуляция	\t
Возврат каретки	\r
Перевод строки	\n
Кавычки	\»
Апостроф	\’

2.6.3. Ввод данных с консоли

Данные с консоли (клавиатуры) можно вводить с помощью класса Scanner. Для работы с классом Scanner необходимо выполнить следующие действия:

1) создать объект класса Scanner и связать его с потоком ввода, например System.in;

2) проверить, доступны ли данные с клавиатуры, методом hasNext, и если данные доступны, считать их методом next;

3) закрыть поток ввода методом close.

Основные методы класса Scanner:

Метод	Описание и общий вид
close	Закрывает поток, связанный с объектом класса Scanner. void close()
hasNext	Возвращает значение true, если в потоке ввода доступны данные любого (для метода hasNext) или заданного типа (для остальных методов этой группы), и false в противном случае. boolean hasNext() boolean hasNextBoolean() boolean hasNextByte() boolean hasNextDouble() boolean hasNextFloat() boolean hasNextInt() boolean hasNextLine() boolean hasNextLong() boolean hasNextShort()
next	Считывает данные из потока ввода и преобразует их к заданному типу. Программа останавливает свое выполнение, пока не будут считаны данные из потока. String next() boolean nextBoolean() byte nextByte() double nextDouble() float nextFloat() int nextInt() String nextLine() long nextLong() short nextShort()

Пример 2-11. Разобьем поток ввода на составляющие, разделенные пробелом, и выведем их с указанием типа.

```
1 | Scanner consoleReader = new Scanner(System.in);
2 | System.out.println("Waiting For Input...");
3 | while (consoleReader.hasNext()) {
4 |     if (consoleReader.hasNextByte()) {
5 |         System.out.println("Byte: " + consoleReader.nextByte());
6 |     } else if (consoleReader.hasNextInt()) {
7 |         System.out.println("Int: " + consoleReader.nextInt());
```

```
8       } else if (consoleReader.hasNextDouble()) {
9           System.out.println("Double: " + consoleReader.nextDouble());
10      } else if (consoleReader.hasNextBoolean()) {
11          System.out.println("Boolean: " + consoleReader.nextBoolean());
12      } else {
13          System.out.println("String: " + consoleReader.next());
14      }
15  }
16  consoleReader.close();
```

При вводе строки

Text 56,89 34567 true 72

на экран будет выведено:

Вывод на экран
`Waiting For Input...`
`Text 56,89 34567 true 72`
`String: Text`
`Double: 56.89`
`Int: 34567`
`Boolean: true`
`Byte: 72`

❑

2.7. Строки

2.7.1. Класс String

2.7.1.1. Особенности строк класса String

Строка представляет последовательность символов (букв, цифр, других символов), каждый из которых имеет в ней свой индекс (позицию).

Особенности строк класса String:

- класс String является ссылочным типом (см. раздел 2.4.1) и имеет методы;
- строка класса String является объектом константного типа (см. раздел 2.4.7);
- каждый символ строки имеет тип char;
- символы кодируются двухбайтовым кодом Unicode;
- первый символ строки имеет индекс 0, а не 1.

Пример 2-12. Символы строки «Just A String» будут иметь следующие индексы:

Индекс	Значение	Индекс	Значение	Индекс	Значение
0	J	5	A	10	i
1	u	6	Пробел	11	n
2	s	7	S	12	g
3	t	8	t		
4	Пробел	9	r		

2.7.1.2. Инициализация строк

Существуют три способа инициализации строк:

1) присваиванием объекту строки, заключенной в кавычки;

2) присваиванием переменной значения другого объекта класса String;

3) созданием строки с помощью конструктора.

Пример 2-13. Три способа инициализации строк в соответствующих строках фрагмента программы:

```
1   String s1 = "String creation";
2   String s2 = s1;
3   String s3 = new String("To Create A String Again");
```

❑

Строка, присваиваемая объекту класса String, заключается в кавычки, тогда как значения символьного типа char — в апострофы.

Пример 2-14. Строка 1 вызовет ошибку, так как переменной *c*1 присваивается значение не символа, а строки, состоящей из одного символа.

```
1   char c1 = "S";
2   char c2 = 'S';
```

Строка 2 ошибку не вызовет. ❑

2.7.1.3. Операции со строками

Операции со строками реализуются методами класса String:

Метод	Описание и общий вид
charAt	Возвращает код символа, находящегося в строке в позиции index. char charAt(int index)

Метод	Описание и общий вид
compareTo	Возвращает значение 0, если исходная строка и строка anotherString равны, значение меньше 0, если исходная строка лексикографически меньше, и значение больше 0, если исходная строка лексикографически больше. Правила лексикографического сравнения приведены в документации к классу String. int compareTo(String anotherString)
contains	Возвращает значение true, если строка содержит подстроку s, и значение false в противном случае. boolean contains(CharSequence s)
endsWith	Возвращает значение true, если строка заканчивается подстрокой suffix, и значение false в противном случае. boolean endsWith(String suffix)
equals	Возвращает значение true, если исходная строка равна строке anObject, и значение false в противном случае. boolean equals(Object anObject)
indexOf	Возвращает позицию первого вхождения символа ch. Если символ ch в строке не встречается, то возвращает значение -1. int indexOf(int ch) Возвращает позицию первого вхождения символа ch, начиная с позиции fromIndex. Если символ ch в строке не встречается, то возвращает значение -1. int indexOf(int ch, int fromIndex) Возвращает позицию первого вхождения подстроки str. Если подстрока str в строке не встречается, то возвращает значение -1. int indexOf(String str) Возвращает позицию первого вхождения подстроки str, начиная с позиции fromIndex. Если подстрока str в строке не встречается, то возвращает значение -1. int indexOf(String str, int fromIndex)
isEmpty	Возвращает значение true, если длина строки нулевая, и значение false в противном случае. boolean isEmpty()

Метод	Описание и общий вид
lastIndexOf	Возвращает позицию последнего вхождения символа ch. Если символ ch в строке не встречается, то возвращает значение -1. int lastIndexOf(int ch) Возвращает позицию последнего вхождения символа ch, начиная с позиции fromIndex. Если символ ch в строке не встречается, то возвращает значение -1. int lastIndexOf(int ch, int fromIndex) Возвращает позицию последнего вхождения подстроки str. Если подстрока str в строке не встречается, то возвращает значение -1. int lastIndexOf(String str) Возвращает позицию последнего вхождения подстроки str, начиная с позиции fromIndex. Если подстрока str в строке не встречается, то возвращает значение -1. int lastIndexOf(String str, int fromIndex)
length	Возвращает количество символов в строке (длину строки). int length()
replace, replaceAll	Возвращает строку, в которой первое вхождение символа target заменено символом replacement. String replace(char target, char replacement) Возвращает строку, в которой первое вхождение подстроки target заменено подстрокой replacement. String replace(CharSequence target, CharSequence replacement) Возвращает строку, в которой все вхождения подстроки, описанной регулярным выражением regex, заменены подстрокой replacement. String replaceAll(String regex, String replacement)
split	Возвращает массив строк типа String, полученных из исходной строки. Разделитель строк задается регулярным выражением regex. String[] split(String regex)
startsWith	Возвращает значение true, если строка заканчивается подстрокой prefix, и значение false в противном случае. boolean startsWith(String prefix)

Метод	Описание и общий вид
substring	Возвращает подстроку исходной строки, состоящую из символов, начиная с позиции beginIndex до конца строки. String substring(int beginIndex) Возвращает подстроку исходной строки, состоящую из символов, начиная с позиции beginIndex и заканчивая позицией endIndex -1 (символ в позиции endIndex в результирующую подстроку не включается). String substring(int beginIndex, int endIndex)
toLowerCase, toUpperCase	Возвращает строку, в которой все буквы исходной строки преобразованы к строчным. String toLowerCase() Возвращает строку, в которой все буквы исходной строки преобразованы к прописным. String toUpperCase()
trim	Возвращает строку, из которой удалены пробелы в начале и в конце строки. String trim()

2.7.1.4. Использование операций над строками

Результат операции со строкой помещается не в исходную строку, а возвращается в качестве значения операции.

Пример 2-15. Рассмотрим применение операций над строками.

В строке 6 вызван метод replace, однако результат его действия не был присвоен ни одной переменной. В строке 7 будет выведена на экран строка sourceString в неизменном виде.

В строке 9 вызван метод replace, заменяющий подстроку This подстрокой It, и результат его работы присвоен строке sourceString.

В строке 12 вызван метод charAt, и его результат присвоен символу symbol. Значение переменной symbol *t* равно второму по счету символу строки sourceString = "It is a string", который находится в позиции 1.

В строке 15 вызван метод substring, и его результат присвоен переменной с тем же именем substring. Результат равен строке, состоящей из символов строки sourceString с позициями от 0 до 4 - 1.

В строке 18 вызывается сразу два метода. Сначала получена подстрока символов с позициями от 0 до 4 - 1, а затем определена длина этой подстроки. В строке 19 выведено значение длины подстроки Length= 4.


```
1  String sourceString = "This is a string";
2  String substring;
3  char symbol;
4  int length;
5
6  sourceString.replace("This","It");
7  System.out.println ("SourceString= " + sourceString);
8
9  sourceString = sourceString.replace("This", "It");
10 System.out.println("SourceString= " + sourceString);
11
12 symbol = sourceString.charAt(1);
13 System.out.println ("Symbol= " + symbol);
14
15 substring = sourceString.substring (0, 4);
16 System.out.println("Substring= " + substring);
17
18 length = sourceString.substring (0, 4).length();
19 System.out.println("Length= " + length);
```

На экран будет выведено:

Вывод на экран

`SourceString= This is a string`
`SourceString= It is a string`
`Symbol= t`
`Substring= It i`
`Length= 4`

❑

2.7.1.5. Сравнение строк

Как уже упоминалось в разделе 2.5.4, объекты, которыми являются строки класса String, сравниваются методом equals, а не операцией сравнения ==.

Пример 2-16. Рассмотрим отличия в использовании операции сравнения == и метода equals для объектов класса String.

```
1  String s1 = "This is a string";
2  String s2 = "This is a string";
3  String s3 = s1.replaceAll(" ", " ");
```

```
4 | System.out.println("s1==s2: " + (s1 == s2));
5 | System.out.println("s1==s3: " + (s1 == s3));
6 | System.out.println("s1 equals s2: " + s1.equals (s2));
7 | System.out.println("s1 equals s3: " + s1.equals (s3));
```

На экран будет выведено:

Вывод на экран
`s1==s2: true`
`s1==s3: false`
`s1 equals s2: true`
`s1 equals s3: true`

Все строковые значения, которые используются в тексте программы, хранятся в одной области памяти. Это сделано для ее более рационального использования. После объявления строковые переменные *s*1 и *s*2 с одинаковыми значениями (строки 1, 2) ссылаются на одну область памяти. Поэтому в строке 4 будет выведено значение true. Если строковое значение будет изменено любой операцией [даже не изменяющей значения символов строки (строка 3; замена всех пробелов на пробелы)], для него будет создан новый объект. Поэтому в строке 5 будет выведено значение false. Однако значения строк остались одинаковыми, поэтому в строке 7 будет выведено значение true. ❑

2.7.1.6. Метод toString

Метод toString определен в классе Object, поэтому он наследуется всеми стандартными и пользовательскими классами. Этот метод преобразует состояние объекта класса в строку, которая используется для вывода на экран.

Заголовок метода toString:

public String toString().

По умолчанию метод toString возвращает строку вида

<имя класса>@<хэш-код>,

где <имя класса> — имя класса объекта, вызвавшего этот метод, а <хэш-код> — значение метода hashCode (см. раздел 1.1.9.2) в шестнадцатеричной системе счисления, вычисленное по значениям всех или части полей объекта.

В некоторых классах этот метод переопределен или перегружен. Например, в классе Arrays, предназначенном для работы с массивами, метод toString имеет параметр Исходный массив и возвращает значения его элементов в квадратных скобках.

При выводе на экран методом print или println объекта System.out метод toString можно не вызывать, так как он вызывается неявно.

Пример 2-17. Объект класса StringBuilder (создан в строке 1) не эквивалентен объекту класса String. Однако для вывода его значения необязательно явно вызывать метод toString.

```
1 | StringBuilder sb = new StringBuilder("A string");
2 | System.out.println(sb);
```

На экран будет выведено:

Вывод на экран
`A string`

❑

2.7.2. Класс StringBuilder

2.7.2.1. Назначение класса StringBuilder

При изменении строки класса String результат изменения записывается в новый объект класса String. Создание нового объекта требует затрат времени и памяти. Этого недостатка лишены объекты класса StringBuilder.

Если метод, вызываемый объектом StringBuilder, изменяет содержимое его строки, то это не приводит к созданию нового объекта, как в случае объекта класса String, а лишь изменяет состояние текущего объекта StringBuilder.

2.7.2.2. Методы класса StringBuilder

Чтобы добавить строку к объекту класса StringBuilder, необходимо вызвать метод append. Этим же методом можно добавлять к строке строковые представления переменных базовых типов и объектов классов.

Пример 2-18. Метод append класса StringBuilder эквивалентен операции соединения строк класса String.

```
1 | String st = "This is ";
2 | st += "a " + "String!";
3 |
```

```
4 | StringBuilder sb = new StringBuilder("This is ");
5 | sb.append("a ").append("String!");
```

❑

Кроме метода append, в классе StringBuilder определен метод insert, вставляющий строковое представление параметра этого метода в определенную позицию строки.

Конструктор класса StringBuilder по умолчанию выделяет память для строки длиной 16 символов. При добавлении новых символов к строке автоматически увеличивается выделенная под нее память. Если максимальная длина строки известна заранее, можно задать ее в конструкторе или с помощью метода ensureCapacity. Размер памяти, выделенной под строку, в символах возвращает метод capacity.

Преобразовать объект класса StringBuilder к объекту класса String можно с помощью метода toString.

Методы append, insert и некоторые другие методы класса StringBuilder приведены в следующей таблице.

Метод	Описание и общий вид
append	Добавляет параметр метода к строке объекта StringBuilder. Параметр метода преобразуется к объекту класса String. При добавлении массива типа char также можно указать позицию первого добавляемого символа index и их количество len. При добавлении объекта, реализующего интерфейс CharSequence, можно указать позиции первого start и последнего end добавляемых символов. StringBuilder append(boolean b) StringBuilder append(char c) StringBuilder append(char[] str) StringBuilder append(char[] str, int index, int len) StringBuilder append(CharSequence s) StringBuilder append(CharSequence s, int start, int end) StringBuilder append(double d) StringBuilder append(float f) StringBuilder append(int i) StringBuilder append(long lng) StringBuilder append(Object obj) StringBuilder append(String str) StringBuilder append(StringBuffer sb)

Метод	Описание и общий вид
insert	Вставляет параметр метода в позицию offset в строку объекта StringBuilder. При вставке массива типа char также можно указать позицию первого вставляемого символа index и их количество len. При вставке объекта, реализующего интерфейс CharSequence, можно указать позиции первого start и последнего end вставляемых символов. StringBuilder insert(int offset, boolean b) StringBuilder insert(int offset, char c) StringBuilder insert(int offset, char[] str) StringBuilder insert(int offset, char[] str, int index, int len) StringBuilder insert(int offset, CharSequence s) StringBuilder insert(int offset, CharSequence s, int start, int end) StringBuilder insert(int offset, double d) StringBuilder insert(int offset, float f) StringBuilder insert(int offset, int i) StringBuilder insert(int offset, long l) StringBuilder insert(int offset, Object obj) StringBuilder insert(int offset, String str)
replace	Заменяет символы, начиная с позиции start и заканчивая позицией end-1, строкой str. StringBuilder replace(int start, int end, String str)
reverse	Обращает строку (первый символ становится последним, второй символ — предпоследним и т.д.). StringBuilder reverse()

В классе StringBuilder определены также методы charAt, indexOf, lastIndexOf, length, substring, аналогичные методам класса String (см. раздел 2.7.1.3).

В классе StringBuilder не переопределены методы equals и hashCode, так как, несмотря на равенство строк, размеры памяти, выделенные под них в объектах класса StringBuilder, могут отличаться. Для сравнения строковых значений объектов sb1 и sb2 класса StringBuilder можно использовать следующий вызов методов:

```
1 | sb1.toString().contentEquals(sb2);
```

2.7.3. Класс StringBuffer

Основным отличием класса StringBuffer от класса StringBuilder является потокобезопасность в многопоточных приложениях. Однако потокобезопасность замедляет скорость обработки.

2.7.4. Форматированный вывод

2.7.4.1. Использование форматированного вывода

В языке Java возможен вывод чисел, дат и строк в заданном формате. Вывод организован точно так же, как и в языке Си: задается строка формата format и список параметров args, значения которых необходимо вывести в определенном виде:

formatOutput(String format, Object... args).

Форматированный вывод используется в следующих классах.

Класс String:

Метод	Описание и общий вид
format	Возвращает строку, отформатированную в соответствии с строкой формата format: static String format(String format, Object... args)

Класс PrintWriter:

Метод	Описание и общий вид
printf	Выводит строку, отформатированную в соответствии с строкой формата format, в поток вывода: PrintWriter printf(String format, Object... args)

Объект System.out (см. раздел 2.6.2) принадлежит классу PrintWriter, поэтому можно использовать метод printf для форматированного вывода на консоль.

Пример 2-19. Вывод переменных типов boolean и int, а также строки класса String в заданном формате.

```
1 | boolean b = true;
2 | int i = 10;
3 | String str = "Text";
4 |
5 | System.out.printf("%B + %3x -> %s", b, i, str);
```


На экран будет выведено:

Вывод на экран
TRUE + a -> Text

Для каждого параметра указан спецификатор формата. Спецификатор формата %B означает, что значение переменной *b* будет выведено как логическое прописными буквами. Спецификатор формата %3x означает, что значение переменной *i* будет выведено как целое число в шестнадцатеричной системе счисления на 3 позициях. Спецификатор формата %s предназначен для вывода строк. Строка формата может содержать и другие символы, используемые при выводе. ❑

2.7.4.2. Строка формата

Строка формата представляет собой последовательность спецификаторов форматов параметров. Строка формата также может содержать и другие символы. По умолчанию вывод выравнивается по правому краю.

Общий вид спецификатора формата:

%[argument_index$][flags][width][.precision]conversion

Спецификатор формата определяет позицию параметра argument_index в списке параметров, флаги модификации вывода flags, общее количество позиций для вывода значения width, количество позиций для вывода вещественной части числа precision и тип формата conversion (например, символ, целое или вещественное число).

Параметр argument_index определяет, к какому параметру из списка параметров относится спецификатор формата. Если параметр опущен, то спецификатор относится к тому же по счету параметру, что и этот спецификатор в строке формата: первый спецификатор относится к первому параметру, второй спецификатор — к второму параметру и т.д.

Пример 2-20. С помощью параметра argument_index значение переменной *f* выводится в различных форматах.

```
1 | float f = 12.39f;
2 | System.out.printf("%f %1$4.3g %1$e", f);
```

Значение переменной *f* выведено с использованием трех форматов:

Вывод на экран
12,390000 12.4 1.239000e+01

В первом спецификаторе формата %f параметр argument_index опущен, так как и без указания этого параметра спецификатор относится к первому параметру. Второй спецификатор формата %1$4.3g. ❑

Флаги модифицируют форматированный вывод. Используемые флаги зависят от типа формата.

Флаг	Общая	Символ	Целые числа	Числа с плавающей точкой	Дата и время	Описание
-	+	+	+	+	+	Вывод будет выравнен по левой границе
#	+[1]	–	+[3]	+	–	Вывод зависит от типа формата
+	–	–	+[4]	+	–	Числа будут всегда иметь знак
пробел	–	–	+[4]	+	–	Положительные числа будут включать начальные пробелы
0	–	–	+	+	–	Числа будут включать начальные пробелы
,	–	–	+[2]	+[5]	–	К целой части числа будут добавлены разделители разрядов
(	–	–	+[4]	+[5]	–	Отрицательные числа будут заключены в скобки

1 Зависит от определения в классе, реализующем интерфейс Formattable.
2 Только для типа формата d.
3 Только для типов формата o, x и X.
4 Только для типа формата d, применяемого для вывода значения типов языка Java: byte и Byte, short и Short, int и Integer, long и Long.
5 Только для типов формата e, E, f, g и G.

Параметр width определяет минимальное общее количество позиций для вывода значения параметра. Это количество включает количество позиций для вывода вещественной части числа. Если для вывода значения требуется большее количество позиций, чем определено параметром width, то значение выводится полностью, а параметр width игнорируется.

Параметр precision определяет количество позиций для вывода вещественной части числа для типов значений *e*, *E* и *f*. Для типов значений *g* и *G* этот параметр определяет количество цифр в выводимом числе. При необходимости число округляется.

Типы формата conversion делятся на следующие категории.

1. Общие — могут использоваться с любыми типами параметров.
2. Символьные — используются с типами и классами, представляющими символы в кодировке Unicode: char, Character, byte, Byte, short, and Short. Эта категория также может использоваться для типов int и Integer, если Character.isValidCodePoint(int) возвращает значение true.
3. Целочисленные — используются с целыми типами языка Java и соответствующими им классами-оболочками: byte и Byte, short и Short, int и Integer, long и Long.
4. Вещественные — используются с вещественными типами и классами-оболочками: float и Float, double и Double.
5. Дата/Время — используются с типами, представляющими дату или время: long и Long, Calendar, Date.
6. Специальные — используются для вывода определенных символов.

Типы формата перечислены в следующей таблице.

Тип формата	Категория	Описание
b, B	Общие	Если значение параметра равно null, то выводится «false». Если тип параметра boolean или Boolean, то выводится возвращаемое значение метода String.valueOf(arg). В остальных случаях выводится «true»
h, H	Общие	Если значение параметра равно null, то выводится «null». В остальных случаях выводится возвращаемое значение метода Integer.toHexString(arg.hashCode())

Тип формата	Категория	Описание
s, S	Общие	Если значение параметра равно null, то выводится «null». Если объект arg реализует интерфейс Formattable, то выводится возвращаемое значение метода arg.formatTo. В остальных случаях выводится возвращаемое значение метода arg.toString()
c, C	Символьные	Вывод символа в кодировке Unicode
d	Целые	Вывод числа в десятичной системе счисления
o	Целые	Вывод числа в восьмеричной системе счисления
x, X	Целые	Вывод числа в шестнадцатеричной системе счисления
e, E	Вещественные	Вывод числа в экспоненциальном виде MeP, где M — мантисса, P — порядок
f	Вещественные	Вывод числа с целой и вещественной частями
g, G	Вещественные	Если модуль значения выводимого числа больше или равен 10^{-4}, но меньше $10^{precision}$, то число выводится в том же формате, что и тип f. Если модуль значения выводимого числа меньше 10^{-4}, но больше или равен $10^{precision}$, то число выводится в том же формате, что и тип e или E
a, A	Вещественные	Вывод вещественного числа в виде 0xMpP, где M — мантисса, P — порядок, в шестнадцатеричной системе счисления
t, T	Дата/Время	Используется для вывода даты и времени в определенном формате
%	Специальные	Вывод символа процента (%)
n	Специальные	Вывод символа перевода строки

Типы формата, обозначаемые прописными буквами, отличаются от типов, обозначаемых строчными буквами, тем, что при форматированном выводе будут использоваться только прописные буквы, что эквивалентно вызову метода toUpperCase класса String.

2.8. Управляющие конструкции

2.8.1. Операторы ветвления

2.8.1.1. Условный оператор

Условный оператор if в зависимости от истинности условия передает управление одним или другим операторам и имеет вид

```
if (<условие>) {
    <операторы 1>;
} [else {
    <операторы 2>;
}]
```

Условие в операторе if представляет собой логическое выражение типа boolean. Если условие истинно (имеет значение true), то выполняются операторы 1, иначе выполняются операторы 2. Ветвь else может отсутствовать. В этом случае при ложности условия никакие действия не выполняются.

Если после ключевого слова else следует другой условный оператор, то эта последовательность операторов оформляется так (п. 7.4 соглашений Java Code Conventions):

```
if (<условие>) {
    <операторы 1>;
} else if {
    <операторы 2>;
}
```

Пример 2-21. В следующем фрагменте программы формируется и выводится на экран сообщение в зависимости от значения переменной *x*.

```
1  String message;
2  int x = 10;
3
4  if (x > 5) {
5      message = x + " is greater than 5";
```

```
6   } else {
7       message = x + " is less or equals 5";
8   }
9
10  System.out.println(message);
```

На экран будет выведено:

Вывод на экран
`10 is greater than 5`

❑

2.8.1.2. Оператор выбора

Если необходимо проверить равенство значения переменной одному из нескольких значений, то используется оператор выбора switch.

Общий вид оператора switch:

```
switch(<переменная>)
{
      case <значение 1>:
        <операторы 1>;
        [break;]
        ...
      case <значение N>:
        <операторы N>;
        [break;]
      [default:
        <операторы N+1>;]
}
```

При равенстве значения переменной одному из значений выполняются следующие за ним операторы. Если в конце операторов стоит оператор break, то выполнение оператора switch заканчивается. Если оператор break отсутствует, то выполняются все последующие операторы до оператора break или до конца оператора switch. Если значение переменной не совпадает ни с одним значением, то выполняется ветвь default. Если ветвь default отсутствует, то никакие действия не выполняются.

Пример 2-22. Если переменная switchParameter имеет значение 1, то на экран будет выведено «12» (строки 4–7). Операторы ветви значения 2 выполнились, так как в ветви значения 1 отсутствует оператор break. Если переменная switchParameter имеет значение 2, то на экран будет выведено «2» (строки 5–7). Ветвь значения 3 не выполнится, так как ветвь значения 2 завершается оператором break. Если переменная switchParameter имеет значение 3, то на экран будет выведено «3» (строка 8).

```
1   int switchParameter;
2   ...
3   switch (switchParameter) {
4       case 1: System.out.print("1");
5       case 2:
6           System.out.print("2");
7           break;
8       case 3: System.out.print("3");
9   }
```

При других значениях переменной switchParameter на экран ничего выведено не будет, так как ветвь default отсутствует. ❑

2.8.2. Операторы цикла

2.8.2.1. Циклы в языке Java

Циклы предназначены для выполнения операторов, составляющих тело цикла. Каждое повторение цикла называется *итерацией*. Цикл заканчивает выполнение итераций в зависимости от условия продолжения (завершения) цикла.

В языке Java существуют четыре вида операторов циклов:

1) цикл с параметром for;
2) цикл с предусловием while;
3) цикл с постусловием do-while;
4) цикл перебора элементов (будет рассмотрен в разделе 2.9.3).

Тело цикла может включать другой цикл любого вида. Циклы внутри других циклов называются *вложенными*.

2.8.2.2. Цикл с параметром

Оператор цикла с параметром for состоит из заголовка, состоящего из трех частей, и тела цикла:

```
for (<начальные значения>; <условие>; <изменение значений>) {
        <операторы>;
}
```

В части <начальные значения> заголовка оператора задается одно или несколько значений переменных цикла — параметров цикла. В части <условие> определяется условие продолжения цикла. Пока это условие истинно, цикл будет выполняться. В части <изменение значений> задается правило изменения параметров цикла на каждой итерации.

Пример 2-23. В цикле с параметром выводятся на экран все целые числа из промежутка от 0 до 9.

```
1  for (int i = 0; i < 10; i++) {
2      System.out.print(i);
3  }
```

На экран будет выведено:

Вывод на экран
0123456789

❑

Шаг изменения параметра цикла можно задавать в части изменения значений.

Пример 2-24. Следующий фрагмент выводит на экран ноль и все четные числа промежутка от 0 до 9.

```
1  for (int i = 0; i < 10; i = i + 2) {
2      System.out.print(i);
3  }
```

На экран будет выведено:

Вывод на экран
02468

❑

2.8.2.3. Операторы цикла с предусловием и постусловием

Оператор цикла с предусловием while имеет вид

```
while (<условие>) {
    <операторы>;
}
```

Оператор цикла с постусловием do-while имеет вид

```
do {
    <операторы>;
}
while (<условие>);
```

Операторы цикла с пред- и постусловием выполняются, пока <условие> истинно. Отличие этих операторов друг от друга заключается в том, что условие в операторе do-while проверятся после тела цикла, а у оператора while — до. Следовательно, у оператора do-while всегда выполнится хотя бы одна итерация, тогда как у оператора while тело цикла может и не выполняться.

Обычно операторы цикла с предусловием и постусловием используются, если неизвестно количество итераций или нет необходимости введения параметра цикла.

Пример 2-25. В цикле while на каждой итерации к строке добавляется символ «*a*».

```
1  String string = "";
2
3  while (string.length() < 6) {
4      string += 'a';
5  }
6
7  System.out.println(string);
```

Цикл завершается, если длина строки станет равной или превысит 6.

В результате будет выведено шесть символов «*a*»:

Вывод на экран
aaaaaa

❑

Пример 2-26. Цикл do-while выполняется пока $x > y$ (строка 9). Как только это условие становится ложным, цикл завершается.

```
1  float x = 2;
2  float delta = 0.5f;
3  double y;
4
5  do {
6      x += delta;
7      y = x * x - 10;
```

```
8       System.out.printf("x=%f; y=%f\n", x, y);
9   } while (x > y);
```

Для вывода на экран использовался форматированный вывод (строка 8), описанный в разделе 2.7.4.

Каждая строка вывода соответствует одной итерации цикла:

Вывод на экран
x=2,500000; y=-3,750000
x=3,000000; y=-1,000000
x=3,500000; y=2,250000
x=4,000000; y=6,000000

❑

2.8.3. Программный блок

Программный блок или *составной оператор* — это последовательность операторов, заключенных в фигурные скобки (операторные скобки). Блоки определяют область видимости переменных. Блоки могут быть вложены один в другой. Блоки используются, чтобы показать тело цикла, метода или класса, ветви оператора условия.

Если в ветви находится только один оператор, то скобки можно не использовать. Однако согласно пп. 7.4–7.7 соглашений Java Code Conventions фигурные скобки необходимо ставить даже при одном операторе в ветви.

2.9. Массивы

2.9.1. Понятие массива

Одним из видов наборов данных в языке Java являются массивы. *Массив* — это структура данных, в которой хранятся значения одного типа. Составляющие массива называются его *элементами*.

Каждый элемент занимает в массиве определенную позицию (индекс). Позиция определяет положение элемента в массиве и позволяет получить доступ к этому элементу. Например, если array — массив, то выражение array[i] соответствует значению *i*-го элемента массива. Элементу массива с указанным индексом можно присваивать значение точно так же, как и переменной.

Пример 2-27. Инициализация и присваивание значения элемента массива переменной var:


```
1   array[i] = 5;
2   var = array[i];
```

❑

Размер или *длина массива* — это количество элементов в массиве. Размер массива задается при создании массива и не может быть изменен. Из массива нельзя удалять или добавлять элементы, но можно изменять их значения. Если решение задачи требует изменения количества элементов, то целесообразней воспользоваться другой структурой данных — списками (будут рассмотрены в разделе 4.4.3). Выражение array.length возвращает размер массива array.

Нумерация позиций элементов массива в языке Java начинается с нуля. Поэтому позиция последнего элемента в массиве всегда на единицу меньше, чем размер массива.

Массивы являются ссылочным типом (см. раздел 2.4.1). Поэтому имя массива представляет собой ссылку на область памяти, в которой этот массив размещен.

2.9.2. Объявление массивов и инициализация их элементов

Будем использовать следующий способ объявления массива с заданным именем и типом:

```
<тип>[] <имя>;
```

Это выражение объявляет массив, но не инициализирует его. Существуют следующие способы объявления и инициализации элементов массива:

```
<тип> [] <имя> = new <тип> [<размер>];
<тип> [] <имя> = {<значения элементов массива через запятую>};
```

Первым способом элементы массива инициализируются значениями по умолчанию для указанного <типа> (см. раздел 2.4.2). Размер массива <размер> может быть задан конкретным значением или значением переменной. Элементы массива могут быть проинициализированы одновременно или отдельно от объявления массива.

Вторым способом элементы массива одновременно объявляются и инициализируются указанными значениями.

Пример 2-28. Массив array1 объявлен и проинициализирован в строке 1. Массив array2 объявлен в строке 3, а проинициализирован в строке 4. Массив array3 объявлен в строке 7, а проинициализирован в строке 8, причем его размер определен значением переменной length.

```
1  int[] array1 = {1, 2, 3, 0};
2
3  char[] array2;
4  array2 = new char [10];
5
6  int arrayLength = 5;
7  double[] array3;
8  array3 = new double [arrayLength];
```

❑

2.9.3. Цикл перебора элементов

Для обработки элементов массивов (и других наборов данных) в языке Java существует, кроме уже перечисленных операторов цикла (см. раздел 2.8.2.2), еще один тип цикла с параметром — цикл перебора элементов некоторого набора данных без применения счетчика. Его общий вид:

```
for (<переменная> : <набор данных>) {
    <операторы>;
}
```

На каждой итерации <переменной> последовательно присваивается каждый элемент <набора данных>, после чего выполняются операторы в блоке.

Пример 2-29. Эквивалентные по результату выполнения цикл перебора элементов (строки 3–5) и цикл с параметром (строки 9–11) выводят на экран элементы массива в одном и том же порядке.

```
1  int[] array = {1, 2, 3, 0};
2
3  for (int element : array) {
4      System.out.print(element);
5  }
6
7  System.out.println();
8
9  for (int i = 0; i < array.length; i++) {
10     System.out.print(array[i]);
11 }
```


И первый, и второй циклы вывели на экран идентичные результаты:

Вывод на экран
1230
1230

❑

Одно отличие цикла с параметром от цикла перебора элементов заключается в том, что цикл с параметром перебирает индексы элементов, а цикл перебора элементов, как следует из названия, перебирает сами элементы массива. Это отличие и определяет область применения цикла перебора элементов — обработку всех элементов набора данных.

Другим отличием цикла перебора элементов является то, что в его теле нельзя изменять элементы массива.

2.9.4. Операции над массивами

2.9.4.1. Операции и циклы

Под *операциями* над массивами понимаются операции над его элементами. Поэтому для этих операций обычно требуется цикл. Количество необходимых циклов определяется, главным образом, размерностью массива. Одномерные массивы требуют, как правило, одного цикла; двумерные массивы — двух циклов, вложенных один в другой, и т.д.

В разделах 2.9.4.2–2.9.4.5 приведены фрагменты программы, реализующие типовые операции над массивами: суммирование элементов массива; поиск минимального и максимального элементов массива; копирование элементов массива. Примеры сортировки и вывода элементов массива приведены в разделе 2.9.5.3.

2.9.4.2. Суммирование элементов массива

Пример 2-30. Суммирование реализовано двумя способами: с помощью цикла с параметром и с помощью цикла перебора элементов.

```
1  int[] array = {1, 2, 3, 0};
2  int summ;
3
4  summ = 0;
5  for (int element : array) {
```

```
6       summ += element;
7   }
8
9   summ = 0;
10  for (int i = 0; i < array.length; i++) {
11      summ += array[i];
12  }
```

❑

2.9.4.3. Поиск минимального значения элементов массива

Пример 2-31. Переменная minIndex хранит позицию минимального значения, а не его значение. Получить минимальное значение можно с помощью выражения array[minIndex] (строки 6, 11).

```
1   int[] array = {1, 2, 3, 0};
2   int minIndex;
3
4   minIndex = 0;
5   for (int i = 1; i < array.length; i++) {
6       if (array[minIndex] > array[i]) {
7           minIndex = i;
8       }
9   }
10
11  System.out.println ("Min= " + array[minIndex]);
```

Перед началом цикла за минимальный принимается первый элемент массива (с индексом 0, строка 4). Поэтому переменной цикла i присваивается значение 1 (строка 5), чтобы не сравнивать значение первого элемента массива с ним самим.

На экран будет выведено значение минимального элемента:

Вывод на экран
`Min= 0`

❑

2.9.4.4. Поиск максимального значения элементов массива

Пример 2-32. Поиск максимального значения элементов массива аналогичен поиску минимального значения.


```
1   int[] array = {1, 2, 3, 0};
2   int maxIndex;
3
4   maxIndex = 0;
5   for (int i = 1; i < array.length; i++) {
6       if (array[maxIndex] < array[i]) {
7           maxIndex = i;
8       }
9   }
10
11  System.out.println ("Max= " + array[maxIndex]);
```

На экран будет выведено значение максимального элемента:

Вывод на экран
`Max= 3`

❑

2.9.4.5. Копирование значений элементов массива

Массивы являются ссылочным типом. Поэтому присваивание одного массива другому создает не копию значений элементов массива, а копию ссылки на исходный массив.

Пример 2-33. Объявим и проинициализируем массив array и присвоим его значение массиву arrayImage (строка 2).

```
1   int[] array = {1, 5, 3, 9};
2   int[] arrayImage = array;
3
4   System.out.printf("The 2nd element in array: %d\n"
5       + "The 2nd element in arrayImage: %d\n", array[1],
6       arrayImage[1]);
7
8   array[1] = 7;
9
10  System.out.printf("The 2nd element has been changed to 7\n"
11      + "The 2nd element in array: %d\n"
12      + "The 2nd element in arrayImage: %d\n", array[1],
13      arrayImage[1]);
```

На экран будет выведено:

Вывод на экран
The 2nd element in array: 5 The 2nd element in arrayImage: 5 The 2nd element has been changed to 7 The 2nd element in array: 7 The 2nd element in arrayImage: 7

При изменении значения второго элемента (с индексом 1) массива array его значение изменилось и в массиве arrayImage. Массив arrayImage содержит копию ссылки на массив array, но не копии значений его элементов. ❑

Скопировать массив можно путем копирования значений его элементов в другой массив в цикле перебора элементов. Однако класс System содержит метод arraycopy для копирования элементов массива.

Метод	Описание и общий вид
arraycopy	Копирует length значений массива src, начиная с позиции srcPos, в массив dest, начиная с позиции destPos. static void (Object src, int srcPos, Object dest, int destPos, int length)

Пример 2-34. Скопируем все значения элементов массива array в массив arrayImage (строка 4).

```
1   int[] array = {1, 5, 3, 9};
    int length = array.length;
2   int[] arrayImage = new int[length];
3
4   System.arraycopy(array, 0, arrayImage, 0, length);
5
6   System.out.printf("The 2nd element in array: %d\n"
7       + "The 2nd element in arrayImage: %d\n", array[1],
8       arrayImage[1]);
9
10  array[1] = 7;
11
12  System.out.printf("The 2nd element has been changed to 7\n"
13      + "The 2nd element in array: %d\n"
14      + "The 2nd element in arrayImage: %d\n", array[1],
15      arrayImage[1]);
```

На экран будет выведено:

Вывод на экран
The 2nd element in array: 5 The 2nd element in arrayImage: 5 The 2nd element has been changed to 7 The 2nd element in array: 7 The 2nd element in arrayImage: 5

Значение второго элемента в массиве arrayImage не изменилось после того, как значение второго элемента было изменено в массиве array. Массив arrayImage содержит не копию ссылки на массив array, а копии значений элементов массива array. ❑

Также скопировать элементы массива можно методом copyOf класса Arrays (будет рассмотрен в разделе 2.9.5.2).

2.9.5. Класс Arrays

2.9.5.1. Назначение класса Arrays

Класс Arrays представляет собой совокупность методов для выполнения наиболее часто используемых операций над массивами. Все методы класса Arrays являются статическими. Поэтому методы можно вызывать, не создавая объект класса, а просто указав перед ними имя класса, например Arrays.binarySearch(...).

2.9.5.2. Методы класса Arrays

Операции над массивами, реализуемые методами класса Arrays:

Метод	Описание и общий вид
binarySearch	Возвращает позицию элемента, равного key, в массиве *a*. Если элемент не найден, то возвращается значение -1. Перед вызовом метода binarySearch необходимо отсортировать массив, вызвав метод sort (см. далее). Если массив не отсортирован, то результат работы метода не определен. Если массив содержит несколько значений, равных key, то неизвестно, позиция какого по счету элемента массива, равного key, будет возвращена. static int binarySearch(byte[] a, byte key) static int binarySearch(char[] a, char key)

Метод	Описание и общий вид
	static int binarySearch(double[] a, double key) static int binarySearch(float[] a, float key) static int binarySearch(int[] a, int key) static int binarySearch(long[] a, long key) static int binarySearch(Object[] a, Object key) static int binarySearch(short[] a, short key) <T>int binarySearch(T[] a, T key, Comparator<? super T> c)
copyOf	Возвращает массив, полученный копированием первых newLength значений элементов массива original. Если длина массива original меньше чем параметр newLength, то дополнительные значения скопированного массива имеют значения по умолчанию. static boolean[] copyOf(boolean[] original, int newLength) static byte[] copyOf(byte[] original, int newLength) static char[] copyOf(char[] original, int newLength) static double[] copyOf(double[] original, int newLength) static float[] copyOf(float[] original, int newLength) static int[] copyOf(int[] original, int newLength) static long[] copyOf(long[] original, int newLength) static short[] copyOf(short[] original, int newLength) static <T> T[] copyOf(T[] original, int newLength) static <T,U> T[] copyOf(U[] original, int newLength, Class<? extends T[]> newType)
equals	Возвращает значение true, если массивы *a* и *a*2 содержат одинаковое число элементов с одинаковыми значениями, расположенных в одинаковом порядке, и значение false в противном случае. static boolean equals(boolean[] a, boolean[] a2) static boolean equals(byte[] a, byte[] a2) static boolean equals(char[] a, char[] a2) static boolean equals(double[] a, double[] a2) static boolean equals(float[] a, float[] a2) static boolean equals(int[] a, int[] a2) static boolean equals(long[] a, long[] a2) static boolean equals(Object[] a, Object[] a2) static boolean equals(short[] a, short[] a2)
fill	Присваивает значение val всем элементам массива *a*. static void fill(boolean[] a, boolean val) static void fill(byte[] a, byte val)

Метод	Описание и общий вид
	static void fill(char[] a, char val) static void fill(double[] a, double val) static void fill(float[] a, float val) static void fill(int[] a, int val) static void fill(long[] a, long val) static void fill(Object[] a, Object val) static void fill(short[] a, short val) Присваивает значение val элементам массива *a*, находящимся в позициях с fromIndex по toIndex – 1. static void fill(boolean[] a, int fromIndex, int toIndex, boolean val) static void fill(byte[] a, int fromIndex, int toIndex, byte val) static void fill(char[] a, int fromIndex, int toIndex, char val) static void fill(double[] a, int fromIndex, int toIndex, double val) static void fill(float[] a, int fromIndex, int toIndex, float val) static void fill(int[] a, int fromIndex, int toIndex, int val) static void fill(long[] a, int fromIndex, int toIndex, long val) static void fill(Object[] a, int fromIndex, int toIndex, Object val) static void fill(short[] a, int fromIndex, int toIndex, short val)
sort	Сортирует по возрастанию (кроме метода с компаратором *c*) элементы массива a методом быстрой сортировки. static void sort(byte[] a) static void sort(char[] a) static void sort(double[] a) static void sort(float[] a) static void sort(int[] a) static void sort(long[] a) static void sort(Object[] a) static void sort(short[] a) static <T>void sort(T[] a, Comparator<? super T> c) Сортирует по возрастанию (кроме метода с компаратором *c*) элементы массива *a*, находящимся в позициях с fromIndex по toIndex - 1, методом быстрой сортировки. static void sort(byte[] a, int fromIndex, int toIndex) static void sort(char[] a, int fromIndex, int toIndex)

Метод	Описание и общий вид
	static void sort(double[] a, int fromIndex, int toIndex) static void sort(float[] a, int fromIndex, int toIndex) static void sort(int[] a, int fromIndex, int toIndex) static void sort(long[] a, int fromIndex, int toIndex) static void sort(Object[] a, int fromIndex, int toIndex) static void sort(short[] a, int fromIndex, int toIndex) static <T>void sort(T[] a, int fromIndex, int toIndex, Comparator<? super T> c)
toString	Возвращает строковое представление массива, состоящее в перечислении через запятую значений элементов массива, заключенном в квадратные скобки. static String toString(boolean[] a) static String toString(byte[] a) static String toString(char[] a) static String toString(double[] a) static String toString(float[] a) static String toString(int[] a) static String toString(long[] a) static String toString(Object[] a) static String toString(short[] a)

2.9.5.3. Сравнение объектов для сортировки и бинарного поиска. Интерфейсы Comparable и Comparator

В классе Arrays определен метод sort, упорядочивающий элементы массива в определенном порядке. Переменные базовых типов сортируются по возрастанию. Порядок упорядочивания объектов классов можно задать, реализовав интерфейсы Comparable и Comparator. При сравнении объектов могут использоваться методы equals и hashCode (см. раздел 1.1.9.2).

Интерфейс Comparable включает метод compareTo, который должен быть определен в классе, объекты которого необходимо сортировать.

Метод	Описание и общий вид
compareTo	Сравнивает текущий объект и объект obj между собой. Подробнее о методе compareTo см. раздел 2.7.1.3. int compareTo(T obj)

Интерфейс Comparator включает два метода, которые должны быть определены в классе, реализующем компаратор. *Компаратор* — это объект, методы которого используются для сравнения объектов некоторого класса.

Метод	Описание и общий вид
compare	Сравнивает объекты o1 и o2 между собой. Возвращает значение 0, если объекты равны; значение меньше 0, если первый объект меньше второго; значение больше 0, если первый объект больше второго. int compare(T o1, T o2)
equals	Выполняет сравнение элементов так же, как метод equals класса Object (см. раздел 1.1.9). boolean equals(Object obj)

2.9.5.4. Применение методов класса Arrays

Пример 2-35. Отсортируем значения элементов массива по убыванию. Стандартный метод sort сортирует значения по возрастанию. Поэтому необходимо задать правило сравнения элементов с помощью интерфейса Comparator. В строке 1 указывается класс объектов, которые будут сравниваться в методе compare. В нашем случае это класс Integer.

```
1  public class DescendingOrderer implements Comparator<Integer> {
2
3      @Override
4      public int compare(Integer arg0, Integer arg1) {
5          return -(arg0 - arg1);
6      }
7  }
```

Следующий фрагмент программы сортирует значения элементов массива по убыванию с помощью компаратора comporator (строка 10) и ищет позицию в массиве — первое вхождение элемента со значением key (строка 14). Для реализации интерфейса элементы массива должны быть объектами. Поэтому элементы массива являются объектами класса-оболочки Integer, а не базового типа int. Для получения строкового представления массива используется метод toString класса Arrays (строки 8, 9). Результирующее сообщение накапливаются в объекте класса StringBuilder с использованием форматированного вывода.

```
1  Integer[] array = { 1, 5, 3, 8 };
2  int keyIndex;
3  int key = 5;
4  Comparator<Integer> comparator = new DescendingOrderer();
5
6  StringBuilder resultString = new StringBuilder();
7  String resultString = String.format("Original array\n%s\n",
8      Arrays.toString(array));
9
10 Arrays.sort(array, comparator);
11 resultString.append(String.format("Sorted array\n%s\n",
12     Arrays.toString(array)));
13
14 keyIndex = Arrays.binarySearch(array, key, comparator);
15 if (keyIndex > -1) {
16     resultString.append(String.format("Index of value %d: %d\n", key,
17         keyIndex));
18 } else {
19     resultString.append(String.format(
20         "Element with value %d isn\'t found\n", key));
21 }
22
23 System.out.print(resultString);
```

В результате элементы отсортированы по убыванию. Элемент со значением 5 находится в первой позиции:

Вывод на экран

`Original array`
`[1, 5, 3, 8]`
`Sorted array`
`[8, 5, 3, 1]`
`Index of value 5: 1`

Если удалить из списка параметров метода sort компаратора comporator (строка 10), то будет выполнена сортировка по умолчанию (по возрастанию) и будут получены следующие результаты:

Вывод на экран

`Original array`
`[1, 5, 3, 8]`
`Sorted array`
`[1, 3, 5, 8]`
`Index of value 5: 2`

❑

2.9.6. Многомерные массивы

В языке Java не существует многомерных массивов как единого объекта. Чтобы создать, например, двумерный массив, необходимо создать одномерный массив одномерных массивов.

Пример 2-36. Объявим двумерные массивы и проинициализируем их.

```
1  int[][] initializedArray = { { 1, 2 }, { 2, 3 }, { 3, 4 } };
2  byte[][] onlyDeclaredArray;
3
4  double[][] one2twoDimArray = new double[3][];
5  one2twoDimArray[0] = new double[1];
6  one2twoDimArray[1] = new double[5];
7  one2twoDimArray[2] = new double[2];
8
9  String outputResult = "";
10 for (int[] array : initializedArray) {
11     outputResult += Arrays.toString(array);
12 }
13 System.out.println(outputResult);
```

В строке 1 объявлен и проинициализирован массив initializedArray размером 2 × 3 явным указанием значений элементов.

В строке 2 объявлен, но не проинициализирован массив onlyDeclaredArray.

В строках 4–7 объявлен двумерный массив one2twoDimArray. В строке 4 объявлено только одно измерение массива. Строки массива инициализируются в строках 5–7. Массив one2twoDimArray не является прямоугольным, как массив initializedArray. В его первой строке — один элемент (строка 5), во второй — 5 (строка 6), а в третьей — 2 (строка 7).

Если в метод toString класса Arrays передать многомерный массив, то будет выведен его хэш-код, но не его элементы. Поэтому для второго и последующих измерений массива нужны отдельные циклы обработки (строки 10–12). В строке 13 на экран будут выведены строки массива:

Вывод на экран
`[1, 2][2, 3][3, 4]`

❑

3. ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

3.1. Основные принципы объектно-ориентированного программирования

3.1.1. Принципы и их назначение

Основными принципами объектно-ориентированного программирования являются инкапсуляция, наследование и полиморфизм. Все эти принципы взаимосвязаны между собой.

Основные принципы определяют подходы к разработке программ с сокращенным дублированием строк программы. Сокращение дублирования позволяет сократить время разработки и тестирования программы, что удешевляет стоимость ее разработки и дальнейшего сопровождения.

Рассмотрим эти принципы и то, как они позволяют сократить дублирование строк программы.

3.1.2. Инкапсуляция

Инкапсуляция — это принцип, определяющий, что чтение и изменение значений полей доступно только через методы для получения и установки значений полей (см. раздел 1.1.7.2). Такое ограничение создает единую точку доступа к полям, что позволяет задавать логику доступа к полям только в этих методах.

Методы получения и установки значений поля с именем <имя> обычно называются get<имя> и set<имя> соответственно.

Инкапсуляция сокращает дублирование строк программы за счет того, что правила обращения к полю реализованы в методах get<имя> и set<имя>, а не при каждом обращении к полю (см. пример 1-03).

3.1.3. Наследование

3.1.3.1. Виды взаимосвязей классов

Можно выделить три типа взаимосвязей между классами: наследование, включение и использование.

Наследование реализует отношение IS-A. Если Y IS-A X, то Y может делать все, что может делать X.

Более подробно наследование будет рассмотрено в разделе 3.1.3.2.

Включение реализует отношение HAS-A. Если X HAS-A Y, то X может использовать все действия Y. Включение реализуется через использование объекта класса Y как поля класса X.

Использование применяется, когда необходимо динамически менять состояние объекта класса X. Для этого в метод класса X передается в качестве параметра объект класса Y, реализующий новое поведение объекта класса X. Объект класса Y необходим классу X только для решения одной задачи, но не является его частью, в отличие от включения.

Пример 3-01. Класс Robot определяет поведение робота. Робот имеет левую руку [поле leftHand (строка 3)] и может двигаться [реализуется объектом robotMover (строка 4).

```
1  public class Robot {
2
3      private Hand leftHand;
4      private RobotMover robotMover;
5
6      public Robot(Hand leftHand, RobotMover robotMover) {
7          this.leftHand = leftHand;
8          this. robotMover = robotMover;
9      }
10
11     public RobotMover getRobotMover() {
12         return this. robotMover;
13     }
14
15     public void catchByLeftHand() {
16         this.leftHand.catch();
17     }
18
19     public void move(Coord x, Coord y) {
20         this.robotMover.move(x, y);
21     }
22 }
```


Обратите внимание, что методы отделены пустой строкой (п. 6.4 соглашений Java Code Conventions) в этом, предыдущих и последующих примерах.

Робот-андроид, описываемый классом Android, отличается от стандартного робота, описываемого классом Robot, тем, что он ходит, а не просто передвигается. Отличия реализованы в методе move. В остальном робот-андроид ничем не отличается от стандартных роботов.

```
1  public class Android extends Robot {
2
3      public void move(Coord x, Coord y) {
4          super.robotMover.walk(x, y);
5      }
6  }
```

Таким образом, реализовано наследование методов класса Robot классом Android. Отличия подкласса от суперкласса реализованы в методе move.

Класс Robot содержит поле класса Hand, реализуя тем самым включение.

```
1  public class Hand {
2
3      public void useHandTool(Tool tool) {
4          tool.execute();
5      }
6
7      public void catch() {
8          // Catch realization
9      }
10 }
```

Робот может использовать ручной инструмент. Такая возможность реализована в методе useHandTool класса Hand (строки 3–5). Параметром метода useHandTool является объект tool класса Tool. Объект tool используется внутри класса useHandTool, однако не является его частью. В данном случае реализуется использование. ❑

Способ взаимодействия между классами выбирается так, чтобы сократить дублирование строк программы.

3.1.3.2. Понятие наследования

Наследование — это способ взаимосвязи классов, при котором объект одного класса может иметь поля и поведение объекта другого класса, а также добавлять к ним характерные только для него поля и поведение.

Наследование позволяет использовать методы суперкласса в подклассах. За счет этого сокращается дублирование строк программы.

Чтобы объявить класс подклассом, необходимо в его заголовке после имени класса добавить директиву extends с именем суперкласса.

У каждого суперкласса может быть несколько подклассов. Но у каждого подкласса может быть только один суперкласс. Множественное наследование классов в языке Java невозможно.

3.1.3.3. Ключевое слово super

Поля, конструкторы или методы суперкласса доступны в его подклассах через ключевое слово super. Ключевое слово super используется в подклассах для обращения к полям или вызова методов суперкласса, например:

```
super.aField;
super.aMethod();
```

Ссылка super() вызывает конструктор суперкласса по умолчанию.

С помощью ключевого слова super можно получить доступ только к непосредственному суперклассу, но не к другим родительским классам.

Пример 3-02. Пусть у класса А есть подкласс Б, а у класса Б — подкласс В. Класс В может получить доступ с помощью ключевого слова super только к составным элементам класса Б, но не класса А. ❑

3.1.3.4. Пример наследования

Пример 3-03. Класс Circle описывает круг. Метод getDiameter возвращает значение диаметра (строки 5–7), а метод getArea — значение площади круга (строки 9–11).

```
1  public class Circle {
2
3      private double radius;
4
5      public double getDiameter() {
6          return this.radius * 2;
7      }
8
9      public double getArea() {
10         return Math.PI * Math.pow(this.radius, 2);
11     }
12 }
```

Класс Cylinder описывает цилиндр и является подклассом класса Circle (строка 1). Класс Cylinder имеет поле height (строка 3). Площадь цилиндра вычисляется по иной формуле, чем у круга. Поэтому метод getArea в классе Cylinder переопределен (строки 5–8), однако площадь рассчитывается с использованием метода getArea суперкласса (строки 6–7). Метод getDiameter наследуется от суперкласса без изменений.

```
1  public class Cylinder extends Circle {
2
3      private double height;
4
5      public double getArea() {
6          return (super.getArea() * 2) + (Math.PI * super.getDiameter()
7              * this.height);
8      }
9  }
```

Таким образом, наследование позволило сократить количество методов в подклассе Cylinder и использовать методы суперкласса Circle при переопределении методов. ❑

3.1.4. Полиморфизм

Полиморфизм заключается в разработке группы классов, имеющих общее поведение. Эти классы имеют методы с одинаковым заголовком, но с различной реализацией.

Полиморфизм сокращает дублирование строк программы за счет использования методов суперкласса и его подклассов с одинаковыми заголовками без необходимости написания метода определения класса, в котором он объявлен. Эти действия выполняются компилятором автоматически.

Таким образом, чтобы пользоваться преимуществами полиморфизма, необходимо добиться того, чтобы заголовки методов в суперклассе и его подклассах были одинаковыми.

Пример 3-04. Пусть имеется иерархия классов, возводящих число в степени 2, 3 и 6.

Класс Power2 имеет единственный метод raise. Метод raise возводит в квадрат параметр arg.

```
1  public class Power2 {
2
3      public long raise(long arg) {
4          return arg * arg;
5      }
6  }
```

Класс Power3 является подклассом класса Power2. В классе Power3 переопределен метод raise, возвращающий куб параметра arg.

```
1  public class Power3 extends Power2 {
2      @Override
3      public long raise(long arg) {
4          return super.raise(arg) * arg;
5      }
6  }
```

Класс Power6 является подклассом класса Power3. Переопределенный метод raise возводит параметр arg в шестую степень.

```
1  public class Power6 extends Power3 {
2      @Override
3      public long raise(long arg) {
4          return super.raise(arg) * super.raise(arg);
5      }
6  }
```

В методе main класса PowerHierarchyDemoRunner выполняются следующие действия. Массив powers имеет тип Power2 и содержит три элемента (строки 9–12): объекты классов Power2, Power3 и Power6. У каждого элемента массива powers в цикле вызывается метод raise, возводящий параметр arg последовательно во вторую, третью и четвертую степени (строки 14–17).


```
1  public class PowerHierarchyDemoRunner {
2
3      public static void main(String[] args) {
4
5          long arg = 2;
6          int powerCount = 3;
7          int[] powerBases = {2, 3, 6};
8
9          Power2[] powers = new Power2[powerCount];
10         powers[0] = new Power2();
11         powers[1] = new Power3();
12         powers[2] = new Power4();
13
14         for (int i = 0; i < powerCount; i++) {
15           System.out.printf("Power %d of %d is %d\n",
16               powerBases[i], arg, powers[i].raise(arg));
17         }
18     }
19 }
```

На экран будет выведено:

Вывод на экран
`Power 2 of 2 is 4`
`Power 3 of 2 is 8`
`Power 6 of 2 is 64`

Хотя элементы массива powers представляют объекты разных классов, благодаря полиморфизму нет необходимости дополнительно проверять, объектом какого класса является элемент массива powers.

Также для сокращения дублирования строк программы используется наследование. Например, шестая степень в методе raise класса Power6 вычисляется через третью степень с помощью метода raise в классе Power3, которая в свою очередь вычисляется через квадрат с помощью метода raise в классе Power2. ❑

Задать общее поведение можно не только с помощью иерархии классов, но и с помощью интерфейсов (будут рассмотрены в разделе 3.6).

3.2. Возвращаясь к классам

3.2.1. Классы-данные и классы-утилиты

В зависимости от функционального назначения все классы можно разделить на классы-данные и классы-утилиты.

Классы-данные описывают сущности предметной области (книга в библиотеке, автомобиль в дорожном движении и т.п.) и их параметры. Как правило, классы-данные включают только поля и методы для доступа к ним. Примером класса-данного является класс Article, описывающий научную статью (см. раздел 1.1.7.2).

Классы-утилиты представляют собой совокупность методов, предназначенных для решения определенных задач. Как правило, классы-утилиты не имеют полей. Примерами классов-утилит являются класс ArticlePublisher (см. раздел 1.1.7.2), содержащий операции с научными статьями — объектами класса Article, и класс Arrays (см. раздел 2.9.5).

3.2.2. Методы и конструкторы

3.2.2.1. Параметры передаются в методы и конструкторы только по значению

Переменные базовых и ссылочных типов передаются в методы и конструкторы через параметры только по значению. Это означает, что создается копия переменной-параметра, которая передается в метод или конструктор.

Имени переменной базового типа поставлено в соответствие значение этой переменной. Поэтому метод получает не копию переменной, а копию значения. При изменении значения переменной внутри метода это значение после окончания выполнения метода не сохраняется.

Имени переменной ссылочного типа поставлена в соответствие ссылка на объект. Поэтому метод работает не с самой ссылкой, а с ее копией, которая ссылается на ту же ячейку памяти. Изменение значения переменной ссылочного типа внутри метода сохраняется после окончания выполнения метода. Однако если попытаться изменить значение объекта константного типа (например, строки типа String), то вместо него будет создан объект с новым значением, ссылка на объект изменится и новое значение не сохранится после окончания работы метода.

Чтобы запретить изменение значения параметра внутри метода, необходимо добавить к его описанию в заголовке метода спецификатор final.

Пример 3-05. Метод changeParameters класса ParameterPasser изменяет свои параметры (строки 6–7) и выводит их значения на экран (строки 9–10). Метод имеет два параметра: параметр i базового типа int и параметр *array* ссылочного типа.

```
1  public class ParameterPasser {
2
3      public final static String FORMAT = "%s\ni=%d; array=%s\n\n";
4
5      public void changeParameters(int i, int[] array) {
6          i = 4;
7          array[0] = 3;
8
9          System.out.printf(ParameterPasser.FORMAT,
10             "In the method", i, Arrays.toString(array));
11     }
12 }
```

Класс ParameterPassingDemoRunner содержит точку входа — метод main. В методе main создается объект класса ParameterPasser (строка 10) и объявляются и инициализируются переменная *i* базового типа (строка 7) и переменная *array* ссылочного типа (строка 8).

Значения переменных *i* и *array* выводятся на экран (строки 10–11), затем вызывается метод changeParameters объекта parameterPasser (строка 13) и значения переменных *i* и *array* вновь выводятся на экран (строки 15–16).

```
1  public class ParameterPassingDemoRunner {
2
3      public static void main(String[] args) {
4
5          int i = 7;
6          int[] array = { 6 };
7
8          ParameterPasser parameterPasser = new ParameterPasser();
9
```

```
10          System.out.printf(ParameterPasser.FORMAT,
11          "Before method call", i, Arrays.toString(array));
12
13          parameterPasser.changeParameters(i, array);
14
15          System.out.printf(ParameterPasser.FORMAT,
16            "After method call", i, Arrays.toString(array));
17      }
18  }
```

Значение переменной *i* базового типа было изменено в методе changeParameters с 7 на 4. Однако после выполнения метода значение осталось без изменений равным 7.

Значение переменной array ссылочного типа было изменено в методе changeParameters. Измененное значение сохранилось и после окончания выполнения метода.

На экран будет выведено:

Вывод на экран
Before method call i=7; array=[6] In the method i=4; array=[3] After method call i=7; array=[3]

❑

3.2.2.2. Методы и конструкторы с неизвестным количеством параметров

Методы и конструкторы классов могут иметь заранее неизвестное количество параметров. В этом случае параметры будут передаваться в метод или конструктор в массиве.

Чтобы реализовать метод или конструктор с неизвестным количеством параметров, в списке параметров метода или конструктора необходимо указать тип неизвестных параметров и название массива, в который будут помещены параметры (обычно он называется args).

Общий вид метода с неизвестным количеством параметров:

[<спецификаторы>] <имя метода> (<тип>... <имя массива>);

Аналогично объявляются конструкторы с неизвестным количеством параметров.

При объявлении методов и конструкторов с неизвестным количеством параметров необходимо соблюдать два правила:

1) если среди параметров есть фиксированные параметры и список параметров с неизвестным количеством, то этот список должен стоять обязательно в конце;

2) в списке параметров может быть только одна последовательность с неизвестным количеством параметров.

Пример 3-06. Метод writeParameters класса ParameterWriter формирует строку с заголовком header и со списком своих параметров из массива args (строки 4—10).

```
1   public class ParameterWriter {
2       public static final String LINE_BREAKER = "\n";
3
4       public String writeParameters(String header, String... args) {
5           String resultString = header + LINE_BREAKER;
6           for (String arg : args) {
7             resultString += arg + LINE_BREAKER;
8           }
9           return resultString;
10      }
11  }
```

В следующем фрагменте создается объект класса ParameterWriter (строка 1) и результат вызова метода writeParameters этого объекта выводится на экран (строки 2—3).

```
1   ParameterWriter parameterWriter = new ParameterWriter();
2   System.out.println(parameterWriter.writeParameters
3       ("Parameter List:", "1", "5", "99"));
```

На экран будет выведено:

Вывод на экран
Parameter List: 1 5 99

❑

3.2.2.3. Возвращаемые значения методов

Результат работы метода должен быть возвращен в то место программы, в котором этот метод был вызван. Возвращаемое значение указывается после оператора return:

return <значение>;

Возвращаемое значение должно иметь тот же тип, что и тип, указанный в заголовке метода.

Оператор return должен быть последним выполняемым оператором метода. Если после оператора return находятся другие операторы, то это вызовет ошибку компиляции.

Если метод не возвращает значение, то тип возвращаемого значения в заголовке метода обозначается словом void. В этом случае оператор return использовать не нужно.

Если необходимо возвратить несколько значений (например, координаты точки в многомерном пространстве), то для этой цели используются массивы или классы с несколькими полями.

3.3. Абстрактные классы и методы

Абстрактный класс описывает состояние и поведение обобщенной сущности предметной области. Поведение задается с помощью методов, в том числе абстрактных.

У абстрактного метода нет реализации. Его объявление содержит спецификатор abstract и заканчивается точкой с запятой после списка параметров в скобках. Наличие абстрактного метода у суперкласса обязывает реализовывать его в подклассах.

Абстрактный метод можно не реализовывать в подклассе, если объявить его абстрактным.

Если класс содержит хотя бы один абстрактный метод, то он должен иметь спецификатор abstract.

3.4. Статические методы и поля

Поля и методы делают статическими, если поля не описывают состояние объекта, а методы не обращаются к полям, характеризующим состояние объекта. Например, для вычисления тригонометрической функции синус нужно знать только значение угла, которое можно передать через параметр метода. Поэтому в стандартном классе Math метод вычисления синуса sin является статическим. Статические поля и методы имеют спецификатор static.

Статический метод может вызываться с помощью имени класса или ссылки на объект, например StaticClass.staticMethod() или staticClassObject.staticMethod(). Однако согласно п. 10.2 соглашений Java Code Conventions обращаться к статическому методу необходимо только через имя класса.

Статические методы не связаны с конкретным объектом класса, а только с классом. Поэтому статические методы не имеют доступа к нестатическим полям и методам своего класса (в том числе через ключевое слово this), но имеют доступ к статическим полям.

Если класс содержит только статические методы и необходимо запретить создание экземпляров класса, то к заголовку конструктора добавляется спецификатор private, что запрещает доступ к конструктору из других классов.

Статические поля существуют для всех объектов класса в единственном экземпляре. Изменение этого поля в одном объекте приведет к изменению поля во всех объектах класса. Как правило, статические поля используются как константы со спецификатором final.

3.5. Классы, методы и поля со спецификатором final

Действие спецификатора final зависит от элемента, перед которым он записан.

Поля со спецификатором final являются константами и должны быть проинициализированы при объявлении, в конструкторе или логическом блоке.

Параметр метода со спецификатором final не может быть изменен в методе.

Метод со спецификатором final не может быть переопределен в подклассах.

Класс со спецификатором final не может иметь подклассов. Все его методы также считаются имеющими спецификатор final, даже если это явно не указано.

3.6. Интерфейсы

3.6.1. Назначение и общий вид интерфейсов

Интерфейсы задают поведение, которое должны иметь классы, реализующие этот интерфейс.

В отличие от абстрактных классов, которые описывают обобщенную сущность предметной области, интерфейсы задают общее поведение классов, описывающих различные сущности.

Все объявленные в интерфейсе методы рассматриваются как имеющие спецификаторы public и abstract, а все поля — как имеющие спецификаторы public, static и final, даже если это не указано явно.

Общий вид интерфейса:

```
[public] interface <имя> [extends <имя1>, <имя2>, ..., <имяN>] {
    // Final fields and method header declarations
},
```

где <имя1>, <имя2>, ..., <имяN> — имена наследуемых интерфейсов.

Чтобы указать, что класс реализует интерфейс, необходимо в заголовке класса после его имени указать директиву implements и имя интерфейса. Класс может реализовать более одного интерфейса. В этом случае их имена перечисляются после директивы implements.

Проверить возможность реализации интерфейса объектом некоторого класса можно операцией instanceof. Операция возвращает значение true типа boolean, если интерфейс реализуется объектом, и значение false в ином случае. Например:

```
someObject instanceof someInterface
```

Пример 3-07. Пусть необходимо сохранять полное состояние графического редактора, чтобы пользователь мог продолжить редактирование на том месте, на котором он его закончил. Для сохранения состояния необходимо, чтобы основные классы графического редактора сохраняли значения своих полей некоторым способом.

Потребуем, чтобы все основные классы графического редактора имели методы для сохранения и восстановления своего состояния в некотором виде. Это требование объявим в виде интерфейса StateStoreable. Все классы, реализующие этот интерфейс, должны иметь два метода: метод storeState для сохранения состояния объекта класса (строка 2) и метод restoreState для восстановления состояния объекта класса (строка 3).

```
1 | public interface StateStoreable {
2 |     void storeState();
3 |     void restoreState();
4 | }
```


Графическое изображение, создаваемое в редакторе, состоит из геометрических фигур. Абстрактный класс Figure является суперклассом для всех фигур, которые можно использовать в редакторе, и описывает их поведение. С помощью директивы implements StateStoreable (строка 1) обязываем все подклассы реализовывать этот интерфейс. В классе Figure нет необходимости реализовывать методы storeState и restoreState, и поэтому эти методы объявлены абстрактными (строки 6 и 9).

```
1  | public abstract class Figure implements StateStoreable {
2  |
3  |     // Fields, constructors, methods
4  |
5  |     @Override
6  |     public abstract void storeState();
7  |
8  |     @Override
9  |     public abstract void restoreState();
10 | }
```

В заголовке класса Rectangle директива implements StateStoreable отсутствует (строка 1). Однако, являясь подклассом класса Figure, класс Rectangle наследует требование реализации методов интерфейса StateStoreable.

Методы storeState и restoreState (строки 6–8 и 11–13) в классе Rectangle не являются абстрактными, поэтому реализация их в классе обязательна.

```
1  | public class Rectangle extends Figure {
2  |
3  |     // Fields, constructors, methods
4  |
5  |     @Override
6  |     public void storeState() {
7  |         // Realization
8  |     }
9  |
10 |     @Override
11 |     public void restoreState() {
12 |         // Realization
```

```
13      }
14  }
```

Кроме изображения, необходимо сохранять настройки пользователя, собранные в классе Preferences. Класс Preferences не связан с классом Figure, поэтому в заголовке класса Preferences необходимо указать требование реализации интерфейса StateStoreable директивой implements StateStoreable (строка 1).

```
1   public class Preferences implements StateStoreable {
2
3       // Fields, constructors, methods
4
5       @Override
6       public void storeState() {
7           // Realization
8       }
9
10      @Override
11      public void restoreState() {
12          // Realization
13      }
14  }
```

❑

3.6.2. Наследование, суперинтерфейсы и подынтерфейсы

По аналогии с суперклассами и подклассами будем называть *суперинтерфейсом* родительский интерфейс, а *подынтерфейсами* — его интерфейсы-наследники.

Интерфейсы, как и классы, могут наследоваться от других интерфейсов. При этом подынтерфейс может иметь несколько суперинтерфейсов. Таким образом, для интерфейсов разрешено множественное наследование, так как это не вызывает неразрешимых ситуаций для наследуемых методов.

Пример 3-08. Пусть имеются интерфейсы, определяющие методы печати объектов.

Интерфейс DisplayPrintable определяет возможность вывода на экран. Также в интерфейсе определена константа ASPECT_RATIO.


```
1   public interface DisplayPrintable {
2       float ASPECT_RATIO = 1.3f;
3
4       void print();
5   }
```

Интерфейс PaperPrintable определяет возможность вывода на бумагу через принтер. В этом интерфейсе, как и в интерфейсе DisplayPrintable, определена константа ASPECT_RATIO. Кроме константы ASPECT_RATIO, также определена константа DENSITY.

```
1   public interface PaperPrintable {
2       float ASPECT_RATIO = 1.3f;
3       int DENSITY = 300;
4
5       void print();
6   }
```

Интерфейс AnywherePrintable является подынтерфейсом интерфейсов DisplayPrintable и PaperPrintable.

```
1   public interface AnywherePrintable
2       extends DisplayPrintable, PaperPrintable {
3   }
```

Класс InterfaceImplementer реализует интерфейс AnywherePrintable.

```
1   public class InterfaceImplementer implements AnywherePrintable {
2
3       @Override
4       public void print() {
5           System.out.println(AnywherePrintable.DENSITY);
6       }
7   }
```

В методе print выводится значение константы AnywherePrintable.DENSITY, которая наследуется от интерфейса PaperPrintable.

Попытка получить значение константы AnywherePrintable.ASPECT_RATIO приведет к ошибке компиляции, так как константа с таким именем определена в обоих суперинтерфейсах DisplayPrintable и PaperPrintable. Чтобы получить ее значение, необ-

ходимо указать имя суперинтерфейса, например PaperPrintable.ASPECT_RATIO.

Следующий фрагмент программы создает объект класса InterfaceImplementer (строка 1), вызывает его метод print (строка 2) и проверяет реализацию созданным объектом суперинтерфейсов (строки 4–5).

```
1 | InterfaceImplementer interfaceImplementer = new InterfaceImplementer();
2 | interfaceImplementer.print();
3 |
4 | System.out.println(interfaceImplementer instanceof DisplayPrintable);
5 | System.out.println(interfaceImplementer instanceof PaperPrintable);
```

Два значения true, выведенные на экран, означают, что объект interfaceImplementer реализует оба интерфейса DisplayPrintable и PaperPrintable:

Вывод на экран
300 true true

❑

3.6.3. Интерфейсы-маркеры

Некоторые интерфейсы не содержат внутри заголовков методов, т.е. являются пустыми. Примерами пустых интерфейсов являются интерфейсы Cloneable, определяющий, можно ли клонировать объекты класса, реализующего этот интерфейс, и Serializable, определяющий, можно ли записать или считать объекты класса, реализующего этот интерфейс, в поток или из потока ввода и вывода.

Такие интерфейсы называются *интерфейсами-маркерами*. Интерфейсы-маркеры не задают поведение класса, а определяют у объектов класса, реализующего этот интерфейс, наличие некоторых свойств.

3.7. Пример проектирования иерархии классов и применения наследования, абстрактных классов, интерфейсов и рефакторинга

Пример 3-09. При проектировании иерархии классов необходимо придерживаться двух правил:

1) выбирать описание иерархии так, чтобы сократить дублирование строк программы;

2) классы должны быть спроектированы так, чтобы изменение одних классов не вызывало изменения других классов (классы как можно меньше «знают» друг о друге).

I. Пусть имеются некоторые объекты, у которых есть одна общая характеристика — площадь. Необходимо спроектировать иерархию классов, в которых вычисляется площадь этих объектов.

Опишем объект круг. Круг характеризуется координатами центра и радиусом. Эти параметры будут являться полями класса Circle. Также класс будет иметь конструктор, который присваивает значения параметров полям, и метод вычисления площади getArea.

```
1  | public class Circle {
2  |
3  |     private double xCenter;
4  |     private double yCenter;
5  |
6  |     private double radius;
7  |
8  |     public Circle(double xCenter, double yCenter, double radius) {
9  |         this.xCenter = xCenter;
10 |         this.yCenter = yCenter;
11 |         this.radius = radius;
12 |     }
13 |
14 |     public double getArea() {
15 |         return Math.PI * Math.pow(this.radius, 2);
16 |     }
17 | }
```

Методы этого и следующих классов, которые не относятся к данному примеру, опущены.

II. Пусть также необходимо описать круг, внутри которого находится отверстие в виде квадрата. Центры круга и квадрата совпадают. Квадрат характеризуется длиной диагонали, проходящей через центр круга.

Эта фигура, помимо характеристик круга, также определяется диагональю квадрата и ее углом относительно оси абсцисс.

Площадь круга без квадрата равна разности площади круга и площади квадрата.

Эта фигура использует все поля и методы круга, поэтому для сокращения дублирования строк программы целесообразно использовать наследование, что соответствует отношению IS-A (см. раздел 3.1.3.1).

Класс QuadrateInCircle будет подклассом класса Circle.

```
1  public class QuadrateInCircle extends Circle {
2
3      private double diagonal;
4      private double angle;
5
6      public QuadrateInCircle(double xCenter, double yCenter,
7          double radius, double diagonal, double angle) {
8          super(xCenter, yCenter, radius);
9          this.angle = angle;
10         this.diagonal = diagonal;
11     }
12
13     @Override
14     public double getArea() {
15         return super.getArea() - Math.pow(this.diagonal, 2) / 2;
16     }
17 }
```

Метод getArea класса QuadrateInCircle переопределяет метод getArea класса Circle, изменяя поведение суперкласса.

В классе QuadrateInCircle вызывались конструктор и метод getArea суперкласса Circle. С одной стороны, за счет этого написано меньше строк программы, с другой — при изменении конструктора или метода getArea класса Circle не придется изменять эти же элементы в классе QuadrateInCircle.

III. Добавим к иерархии класс, описывающий геометрическую фигуру квадрат. Квадрат характеризуется также координатами центра, диагональю и ее углом относительно оси абсцисс.

Площадь квадрата вычисляется по формулам, отличным от формул вычисления площади круга или круга без квадрата.

Наследование не приведет к сокращению дублирования строк программы, так как все поля и реализацию метода getArea придется писать заново. Поэтому класс Quadrate не стоит делать подклассом класса Circle или QuadrateInCircle.

Класс Quadrate имеет следующее объявление:

```
1  public class Quadrate {
2
3      private double xCenter;
4      private double yCenter;
5
6      private double diagonal;
7      private double angle;
8
9      public Quadrate(double xCenter, double yCenter,
10         double diagonal, double angle) {
11         super();
12         this.xCenter = xCenter;
13         this.yCenter = yCenter;
14         this.diagonal = diagonal;
15         this.angle = angle;
16     }
17
18     public double getArea() {
19         return Math.pow(this.diagonal, 2) / 2;
20     }
21 }
```

IV. Однако теперь класс QuadrateInCircle содержит строки, повторяющиеся в методе getArea класса Quadrate. Необходимо переписать класс QuadrateInCircle. Этот процесс называется *рефакторингом*. Рефакторинг не означает, что программа неверно работает. Рефакторинг необходим, чтобы обновить классы из-за добавления, удаления или изменения других классов и чтобы соответствовать правилам, приведенным в начале этого раздела.

Класс QuadrateInCircle должен быть подклассом классов Circle и Quadrate. Однако множественное наследование для классов невозможно. Сделаем объекты классов Circle и Quadrate полями класса QuadrateInCircle, что соответствует включению — отношению HAS-A (см. раздел 3.1.3.1). Площадь круга без квадрата вычислим с помощью методов getArea классов Circle и Quadrate.

Класс QuadrateInCircle после рефакторинга:

```
1  public class QuadrateInCircle {
2
3      private Circle outerCircle;
4      private Quadrate innerQuadrate;
5
6      public QuadrateInCircle(double xCenter, double yCenter,
7              double radius, double diagonal, double angle) {
8          outerCircle = new Circle(xCenter, yCenter, radius);
9          innerQuadrate = new Quadrate
10             (xCenter, yCenter, diagonal, angle);
11     }
12
13     public double getArea() {
14         return outerCircle.getArea() — innerQuadrate.getArea();
15     }
16 }
```

Теперь весь класс QuadrateInCircle основан на объектах и методах классов Circle и Quadrate и не содержит дублирования строк программы.

V. Любую иерархию классов разрабатывает и расширяет не один программист, а целая группа разработчиков. Поэтому необходимо описать правила, которым должны соответствовать новые классы, добавляемые в иерархию.

Для рассматриваемой иерархии правило одно: каждый класс должен иметь метод, возвращающий значение площади объекта. Этот метод должен называться именно getArea, но не getObjectArea или getFigureSquare и т.п.

Для классов, описывающих одинаковые сущности — фигуры, это правило опишем в виде абстрактного класса Shape.

```
1  public abstract class Shape {
2
3      public abstract double getArea();
4  }
```

Класс Shape и метод getArea имеют спецификатор abstract. Наличие абстрактного метода getArea означает, что неизвестно, какую реализацию будет иметь этот метод в подклассах, но каждый подкласс должен иметь такой метод и его реализовывать (или вновь быть абстрактным, если класс абстрактный).

Классы Circle, Quadrate, QuadrateInCircle должны быть объявлены подклассами класса Shape с помощью инструкции extends Shape, которую необходимо добавить к заголовкам этих классов.

VI. Не только фигуры могут иметь площадь, но и страны мира. Пусть необходимо в иерархии классов также определять площади стран.

Страны мира описываются объектами класса Country.

```
1  public class Country {
2
3      private int population;
4      private double area;
5      private String capital;
6
7      public int getPopulation() {
8          return population;
9      }
10
11     public double getArea () {
12         return area;
13     }
14
15     public String getCapital() {
16         return capital;
17     }
18 }
```

Класс Country также имеет метод getArea, как и подклассы класса Shape. Страна и геометрическая фигура являются разными сущностями, поэтому класс Country не может быть подклассом класса Shape. Класс Country реализует иное поведение, чем класс Shape. Эти классы объединяет только наличие метода getArea. Поэтому использовать наследование нельзя.

Предположим, что к иерархии будут добавляться и другие сущности, для которых вычисляется их площадь. Необходимо описать правило для рассматриваемой иерархии классов: каждый класс должен иметь метод getArea.

Абстрактный класс является шаблоном для подклассов, описывающих вариации одной сущности. Для задания общего поведения классов, описывающих разные сущности, используют интерфейсы.

Определим интерфейс Areable, обязывающий все классы, которые его реализуют, иметь в описании метод getArea.

```
1  public interface Areable {
2
3      double getArea();
4  }
```

Необходимо добавить к заголовкам классов Shape и Country, описанным в этом разделе, инструкцию implements Areable. Классы Circle, Quadrate и QuadrateInCircle будут обязаны реализовывать этот метод, так как они являются подклассами класса Shape, у которого метод getArea является абстрактным. После этого отсутствие метода getArea в классе с такой инструкцией в заголовке приведет к ошибке компиляции. ❑

3.8. Перечисления Enum

3.8.1. Особенности перечислений Enum

Некоторые переменные должны иметь небольшое и конечное число возможных значений, например дни недели (понедельник, вторник и т.д.), стороны света (север, запад, юг, восток). Для ограничения числа значений переменной используют перечисления.

Перечисление описывается так же, как и класс, только вместо ключевого слова class используется ключевое слово enum. В теле перечисления описываются константы — значения, которые может принимать объект этого перечисления.

Объекты перечисления создаются без использования оператора new как переменные базовых типов.

3.8.2. Перечисления как совокупность констант

Пример 3-10. В одном из способов формирования цвета изображения используются составляющие красного, зеленого и синего цветов. Этот способ имеет название RGB, образованное по первым буквам английских названий этих цветов. Опишем эти цвета в виде перечисления.

```
1  public enum RGBColorsSimple {
2      RED, GREEN, BLUE;
3  }
```

Типичный фрагмент работы с перечислением состоит из определения значения переменной типа перечисления и оператора выбора switch, вызывающего операторы в зависимости от этого значения.

```
1  RGBColorsSimple rgbColorsSimple = RGBColorsSimple.BLUE;
2
3  switch (rgbColorsSimple) {
4  case RED: {
5      // Actions for Red
6  }
7  case GREEN: {
8      // Actions for Green
9  }
10 case BLUE: {
11     // Actions for Blue
12 }
13 }
```

❑

Использование оператора switch затрудняет расширение программы при появлении новых констант в перечислении. Поэтому вместо оператора switch при работе с перечислениями необходимо использовать перечисления как суперкласс для подклассов констант.

3.8.3. Перечисления как суперкласс для подклассов констант

Перечисление может иметь поля, конструкторы и методы, задающие поведение как всего перечисления, так и отдельных констант. Также перечисления могут реализовывать интерфейсы.

Пример 3-11. Пусть необходимо в зависимости от значения константы цвета RGB выполнять определенные действия, а также хранить для каждой константы дополнительные параметры.

Перечисление RGBColorsComplicated имеет следующее описание:

```
1  public enum RGBColorsComplicated {
2      RED(4) {
3          public void someAbstractMethod() {
4            // Realization
5            // Call special method
6            someMethodSpeciallyForRed();
7          }
8
9          public void someMethodSpeciallyForRed() {
10           // Realization
11         }
12     },
13     GREEN(2) {
14         public void someAbstractMethod() {
15           // Realization
16         }
17     },
18     BLUE(1) {
19         public void someAbstractMethod() {
20           // Realization
21         }
22     };
23
24     private int colorCode;
25
26     RGBColorsComplicated(int colorCode) {
27         this.colorCode = colorCode;
28     }
29
30     public int getColorCode() {
31         return this.colorCode;
32     }
33
34     public void someCommonMethod() {
35         // Realization
36     }
37
38     public abstract void someAbstractMethod();
39 }
```

Значения, записанные после констант (4, 2, 1) (строки 2, 13, 18), являются параметром конструктора (строки 26–28). В конструкторе значение параметра присваивается полю colorCode (строка 27). Значение поля colorCode не является общим для перечисления, а свое для каждой константы.

В строке 38 объявлен абстрактный метод someAbstractMethod. Этот метод должен быть реализован для каждой из констант (строки 3–7, 14–16, 19–21).

Также перечисления могут иметь общие для всех констант методы, например метод someCommonMethod (строки 34–36).

Фрагмент программы с использованием переменных и констант перечисления и вызова методов:

```
1  RGBColorsComplicated rgbColorsComplicated =
2      RGBColorsComplicated.GREEN;
3  rgbColorsComplicated.someAbstractMethod();
4  RGBColorsComplicated.RED.someCommonMethod();
5  System.out.println(RGBColorsComplicated.BLUE.getColorCode());
6  System.out.println(Arrays.toString(RGBColorsComplicated.values()));
```

Методы вызываются как через объекты перечисления (строка 3), так и через указание перечисления и константу (строки 4–5).

При вызове метода someAbstractMethod (строка 3) произойдет обращение к реализации этого метода в константе GREEN, так как объект rgbColorsComplicated имеет значение этой константы. ❑

Перечисление представляет собой суперкласс, подклассами которого являются константы перечисления. Количество подклассов ограничено числом констант. Другие подклассы без расширения перечисления создать нельзя.

Конструкторы вызываются при инициализации любого из элементов. Конструктор не может быть объявлен со спецификаторами public или protected, так как не вызывается явно, и перечисление не может быть суперклассом для подклассов, которые не являются его константами.

При инициализации хотя бы одной константы перечисления инициализируются и все остальные константы.

3.8.4. Методы перечислений Enum

Некоторые стандартные методы перечислений Enum:

Метод	Описание и общий вид
compareTo	Сравнивает текущий объект перечисления с объектом этого перечисления obj. int compareTo(E obj)
ordinal	Возвращает порядковый номер константы в перечислении, начиная с нуля. int ordinal()
toString	Возвращает строковое значение константы. String toString()
valueOf	Возвращает объект, значение которого равно константе, заданной параметром name. enumType valueOf(String name) Возвращает объект, значение которого равно константе перечисления T, заданной параметром name. <T extends Enum<T>> T valueOf(Class<T> enumType, String name)
values	Возвращает массив, содержащий константы перечисления в порядке их объявления. enumType[] values()

4. СТАНДАРТНЫЕ КЛАССЫ И БИБЛИОТЕКИ

4.1. Создание многоязычных интерфейсов пользователя

4.1.1. Класс Locale

Класс Locale позволяет изменять форматы вывода даты, чисел в зависимости от языка или языка и страны.

Объект класса Locale создается конструктором, например:

```
Locale russianLanguage = new Locale("ru");
Locale usEnglish = new Locale("en", "US");
```

или присвоением константы, определенной в этом же классе, например:

```
Locale french = Locale.FRENCH;
Locale localeForFrance = Locale.FRANCE;
```

С помощью методов getDefault и setDefault можно получить или установить объект класса Locale, используемый по умолчанию.

4.1.2. Файлы ресурсов

При разработке многоязычного интерфейса пользователя необходимо хранить тексты сообщений, надписи в окнах приложения и другие текстовые константы на разных языках.

Для хранения текстовых констант на разных языках необходимо в папке src проекта приложения создать новую папку, например с именем resources. Для каждого языка необходимо создать в этой папке отдельный текстовый файл ресурсов. Имя файла ресурсов имеет следующие форматы.

Для ресурса по умолчанию:

<имя ресурса>.properties

Для ресурса для определенного языка:

<имя ресурса>_<обозначение языка>.properties

Для ресурса для определенного языка и страны:

<имя ресурса>_<обозначение языка>_<обозначение страны>.properties

Имя ресурса выбирается произвольно, например text.

Обозначение языка представляет собой двухбуквенное сочетание, записанное строчными буквами, например ru — для русского языка, en — для английского, de – для немецкого.

Обозначение страны представляет собой двухбуквенное сочетание, записанное прописными буквами, например RU — для России, US — для США, CH — для Швейцарии.

Файл ресурсов содержит пары названий ключей и их значений через знак «=», например:

```
mainform.button.close.caption = Close
```

Рекомендуется создавать иерархические имена ключей, соответствующие их назначению.

Также в файле ресурсов кроме текстовых констант можно хранить массивы и объекты.

4.1.3. Класс ResourceBundle

Доступ к файлам ресурсов осуществляется с помощью объекта класса ResourceBundle.

Объект класса ResourceBundle создается статическим методом getBundle этого класса с указанием только имени ресурса или также объекта класса Locale. Например:

```
ResourceBundle rb1 = ResourceBundle("text");
ResourceBundle rb2 = ResourceBundle("resources.text", Locale.UK);
```

Название resources.text означает, что файлы ресурсов с именем text находятся в папке resources.

Получить значение текстовой константы из файлов ресурсов можно с помощью метода getString, параметром которого является имя ключа. Например, вызов этого метода

```
rb1.getString("mainform.button.close.caption");
```

вернет значение ключа «Close» (см. раздел 4.1.2).

4.1.4. Пример использования классов Locale и ResourceBundle

Пример 4-01. Текстовые константы должны быть доступны из различных методов и объектов приложения. Чтобы не создавать в каждом классе или методе объект класса ResourceBundle для доступа

к файлам ресурсов, создадим класс со статическими методами, доступными в любом месте программы. Метод getString класса ResourceBundle не является статическим, поэтому «обернем» класс ResourceBundle классом со статическими методами и полями. Объект класса ResourceBundle внутри класса-«обертки» является единой точкой доступа к файлам ресурсов. Поэтому он должен быть создан в единственном экземпляре. Для этого используется шаблон проектирования «Одиночка» (Singleton).

Класс Resourcer имеет следующее описание:

```
1  public class Resourcer {
2      private static final String DEFAULT_PROPERTY_NAME =
3          "resources.text";
4      private static String basename = null;
5      private static ResourceBundle resources = null;
6
7      private Resourcer() {
8      }
9
10     public static void createResourcer() {
11         Resourcer.createResourcer(null);
12     }
13
14     public static void setBasename(String basename) {
15         if (basename == null) {
16             if (Resourcer.basename == null) {
17                 Resourcer.basename =
18                         Resourcer.DEFAULT_PROPERTY_NAME;
19             }
20         } else {
21             Resourcer.basename = basename;
22         }
23     }
24
25     public static void createResourcer(String basename) {
26         if (Resourcer.resources == null) {
27             Resourcer.setBasename(basename);
28             Resourcer.resources = ResourceBundle.getBundle(
29                     Resourcer.basename, Locale.getDefault());
```

```
30          } else {
31              Locale systemLocale = Locale.getDefault();
32              Locale resourcerLocale = Resourcer.resources.getLocale();
33              if (!(resourcerLocale.equals(systemLocale))) {
34                      Resourcer.resources = ResourceBundle.getBundle(
35                      Resourcer.basename, systemLocale);
36              }
37          }
38      }
39
40      public static String getString(String parameter) {
41          Resourcer.createResourcer();
42          return Resourcer.resources.getString(parameter);
43      }
44  }
```

Все методы класса Resourcer являются статическими и не требуют создания объекта этого класса. Объект класса ResourceBundle создается при получении значения ключа или при смене настроек языка по умолчанию.

Следующие файлы ресурсов находятся в папке resources.

Файл ресурсов по умолчанию text.properties. Ключ string.default.locale (строка 2) не содержится в других файлах ресурсов, поэтому его значение будет всегда браться из этого файла.

```
1  string.default.locale = Default locale:
2  string.test = English
```

Файл ресурсов для русского языка в России в файле text_ru_RU.properties. Значение ключа string.test, равное строке «Русский», записано кодами символов в кодировке Unicode (строка 1).

```
1  string.test = \u0420\u0443\u0441\u0441\u043A\u0438\u0439
```

В следующем фрагменте программы выводятся строковые константы для текущего языка и английского языка.

```
1  String result;
2
3  result = Resourcer.getString("string.default.locale")
4      + Locale.getDefault().toString() + "\n";
5  result += Resourcer.getString("string.test");
6  System.out.println(result);
7
8  Locale.setDefault(Locale.US);
9
10 result = Resourcer.getString("string.default.locale")
11     + Locale.getDefault().toString() + "\n";
12 result += Resourcer.getString("string.test");
13 System.out.println(result);
```

По умолчанию в системе установлены настройки для русского языка в России. Поэтому значение ключа string.test взято из файла text_ru_RU.properties. После смены языка по умолчанию на английский язык для США (строка 8) значение ключа string.test получено из файла text.properties.

Вывод на экран

`Default locale: ru_RU`
`Русский`
`Default locale: en_US`
`English`

❑

4.2. Исключения

4.2.1. Зачем нужны исключения

Исключение (исключительная ситуация) — это ситуация, вызывающая отклонение от нормальной работы программы. Примерами исключений могут быть открытие несуществующего файла или неверный формат входных данных.

Возникновение исключения вызывает аварийное завершение программы. Поэтому исключения необходимо обрабатывать, т.е. отойти от нормальной работы программы.

Обычно исключение обрабатывается одним из двух способов:

1) устранить ошибку, вызвавшую исключение; например, если необходимые данные не могут быть получены, то можно заменить их значениями по умолчанию;

2) выдать сообщение пользователю или записать его в журнал работы приложения.

Исключения представляют собой сообщения об ошибке при работе программы, на которые программа должна прореагировать.

Исключения в языке Java являются объектами. Классы исключений образуют иерархию. Все исключения являются подклассами класса Throwable.

Если стандартные классы исключений не соответствуют некоторой исключительной ситуации, то необходимо создать описывающий ее класс исключения.

4.2.2. Конструкция try-catch-finally

Основная конструкция языка Java для работы с исключениями — это конструкция try-catch-finally. В части try размещается фрагмент программы, который может вызвать исключение, в части catch — обработка исключения, а в части finally, которая может отсутствовать, — строки программы, выполняющиеся вне зависимости от того, было вызвано исключение или нет. Часть finally обычно используется для закрытия потоков ввода и вывода данных (будут рассмотрены в разделе 4.3).

Общий вид конструкции try-catch-finally:

```
try {
        // Операторы, которые могут вызвать исключение
} catch (<Класс исключения 1> <переменная 1>) {
        // Обработка исключения 1
} [catch (<Класс исключения 2> <переменная 2>) {
        // Обработка исключения 2
}]
...
[finally {
        // Действия, которые должны быть выполнены
        // вне зависимости от того, имело место исключение или нет
}]
```

При возникновении исключительной ситуации в некотором фрагменте программы происходит поиск конструкции try-catch-finally во всех методах, прямо или косвенно вызвавших этот фрагмент. В случае отсутствия такой конструкции сообщение об ошибке выводится на консоль через поток System.err. В случае наличия нескольких частей catch выполняется первая из них, а остальные игнорируются.

В любой части try, catch или finally может быть вложена другая конструкция try-catch-finally.

4.2.3. Оператор throw

В некоторых случаях (например, при невозможности дальнейшего выполнения программы) требуется специально вызвать исключение, чтобы быстро передать управление в ту часть программы, в которой обрабатываются исключения. Это действие выполняется оператором throw.

Общий вид оператора throw с уже созданным объектом исключения:

throw <Объект класса исключения>,

или с созданием такого объекта:

throw new <Класс исключения>.

Объект исключения должен быть подклассом класса Throwable.

4.2.4. Директива throws

Если в методе может возникнуть исключение, но оно не может быть обработано в нем, то к заголовку метода необходимо добавить директиву throws.

Общий вид директивы throws:

<заголовок метода> throws <Класс исключения>.

4.2.5. Где обрабатывать исключения

Предположим, метод А вызывает метод Б, который в свою очередь вызывает метод В. В методе В может возникнуть исключение. В каком методе необходимо обрабатывать исключение конструкцией try-catch-finally? В том методе, в котором это исключение можно обработать.

Если в методе В можно исправить ошибку, вызвавшую исключение, то конструкцию try-catch-finally следует разместить в методе В. Если ошибку исправить не удается (например, файл с данными не найден), а диалог с пользователем ведется в методе А, то конструкция try-catch-finally помещается в метод А. В методе А пользователю сообщается о случившейся ошибке.

4.3. Работа с файлами

4.3.1. Байтовые и символьные потоки

Кроме стандартных потоков in, out и err (см. раздел 2.6), предназначенных для работы с консолью, в языке Java определены потоки

ввода и вывода данных, например из файлов на внешних запоминающих устройствах.

Файл представляет собой именованную последовательность составляющих его байт. Для чтения и записи данных в виде байт в языке Java существуют байтовые потоки. Однако последовательности байт могут кодировать символы, составляющие текст. В этом случае для чтения и записи применяются символьные потоки.

Создать, удалить, переименовать и выполнить другие операции с файлами и папками можно с помощью методов класса File. Объекты класса File также являются объектно-ориентированным описанием файла и служат параметрами конструкторов классов байтовых или символьных потоков.

В некоторых случаях работа приложения будет остановлена до тех пор, пока не будут считаны или записаны данные в поток.

При работе с потоками ввода или вывода данных программа использует ресурсы памяти и процессора компьютерной системы. Поэтому после окончания работы с потоком необходимо его обязательно закрыть методом close, что освобождает ресурсы.

4.3.2. Класс File

4.3.2.1. Назначение класса File

Класс File является объектно-ориентированным представлением файлов и папок файловой подсистемы операционной системы компьютера и операций с ними.

Объекты класса File являются параметрами конструкторов и методов других классов для работы с файлами и потоками ввода и вывода (например, классы RandomAccessFile, FileOutputStream, FileReader).

4.3.2.2. Конструкторы класса File

Конструкторы позволяют создать объект, используя в качестве входных данных полный путь к файлу или другие объекты класса File, описывающие папки, в которых необходимо создать новый файл или другую папку.

Конструкторы класса File:

Описание и общий вид
Путь к файлу определяется на основе пути parent и пути child. `File(File parent, String child)`
Путь к файлу задается строкой pathname. `File(String pathname)`
Путь к файлу определяется на основе строк parent и child. `File(String parent, String child)`
Путь к файлу преобразуется из ссылки, начинающейся с file. `File(URI uri)`

4.3.2.3. Основные методы класса File

Метод	Описание и общий вид
createNewFile	Создает пустой файл. Возвращает значение true, если файл был создан успешно, и значение false, если файл уже существует. `boolean createNewFile()`
delete	Удаляет файл или папку. Папка перед вызовом этого метода должна быть пустой. Возвращает значение true, если файл или папка были удалены успешно, и значение false в противном случае. `boolean delete()`
exists	Проверяет, существует ли файл или папка. Возвращает значение true, если файл или папка существует, и значение false в противном случае. `boolean exists()`
getAbsolutePath	Возвращает полный путь к файлу (имя диска, папки и имя файла). `String getAbsolutePath()`
mkdir	Создает папку. Возвращает значение true, если папка была создана успешно, и значение false в противном случае. `boolean mkdir()`
mkdirs	Создает папку, а также родительские папки, если они не существуют. Возвращает значение true, если папка была создана успешно, и значение false в противном случае. `boolean mkdirs()`

Метод	Описание и общий вид
renameTo	Переименовывает файл. Файл будет иметь имя dest. Возвращает значение true, если файл был переименован успешно, и значение false в противном случае. Не предназначен для переноса файла из одной папки в другую. boolean renameTo(File dest)

4.3.2.4. Константное поле File.separator

Вид разделителя элементов пути к файлу или папке зависит от операционной системы. Например, в операционной системе семейства Microsoft Windows таким разделителем является наклонная черта «\». Чтобы сделать программу независящей от операционной системы, необходимо использовать константное поле File.separator при формировании пути к файлу или папке.

Пример 4-02. Метод formPath класса FilePathFormer возвращает путь из имен вложенных друг в друга папок и имени файла. Между именами вставляется разделитель separator. Для формирования пути используется метод join класса StringJoiner, накапливающий строку и добавляющий разделитель между исходными строками.

Класс StringJoiner имеет следующее описание:

```
1  public class StringJoiner {
2      private StringJoiner() {}
3
4      public static String join(String separator, String... args) {
5          if (args.length == 0) {
6              return "";
7          }
8
9          if (args.length == 1) {
10             return args[0];
11         }
12
13         StringBuilder resultString = new StringBuilder();
14
15         for (int i = 0; i < args.length - 1; i++) {
16             resultString.append(args[i]);
17             if (!args[i].endsWith(separator)) {
18                 resultString.append(separator);
19             }
20         }
21         resultString.append(args[args.length - 1]);
22
23         return resultString.toString();
24     }
25 }
```

Статический метод join (строки 4–24) является методом с неизвестным количеством параметров. Для накопления строки используется объект resultString класса StringBuilder (создан в строке 13).

Класс FilePathFormer в своем основном методе formPath вызывает метод join класса StringJoiner:

```
1  public class FilePathFormer {
2      private FilePathFormer() {}
3
4      public static String formPath(String separator,
5              String... pathElements) {
6          String path = StringJoiner.join(separator, pathElements);
7          return path;
8      }
9  }
```

При вызове метода formPath создается путь, заданный в его параметрах.

```
1  String path = FilePathFormer.formPath
2      (File.separator, "Folder1\\", "Folder2", "file.txt");
3  System.out.println(path);
```

На экран будет выведен сформированный путь к файлу:

Вывод на экран
`Folder1\Folder2\file.txt`

❑

4.3.3. Байтовые потоки

4.3.3.1. Родительские классы байтовых потоков

Все классы байтовых потоков являются прямыми или косвенными наследниками абстрактного класса потока ввода InputStream и абстрактного класса потока вывода OutputStream.

4.3.3.2. Класс FileInputStream

Класс FileInputStream создает байтовый поток чтения данных из файла.

Методы класса FileInputStream:

Метод	Описание и общий вид
available	Возвращает количество доступных для чтения из потока байт. int available()
close	Закрывает поток. void close()
read	Возвращает считанный байт из потока или значение -1, если достигнут конец файла. int read() Читает в этот массив *b* из потока количество байт, равное размеру этого массива. Возвращает количество считанных из потока байт или значение -1, если достигнут конец файла. int read(byte[] b) Читает из потока len байт, начиная с байта с номером off, в массив *b*. Возвращает количество считанных из потока байт или значение -1, если достигнут конец файла. int read(byte[] b, int off, int len)

Методы read блокируют выполнение программы до тех пор, пока не станут доступными данные для чтения.

4.3.3.3. Класс FileOutputStream

Класс FileOutputStream создает байтовый поток записи данных в файл. Может дописать данные в конец файла или создать вместо существующего новый файл.

Некоторые методы класса FileOutputStream:

Метод	Описание и общий вид
close	Закрывает поток. void close()
write	Записывает элементы массива *b* в поток. void write(byte[] b) Записывает в поток len байт, начиная с байта с номером off, из массива *b*. void write(byte[] b, int off, int len) Записывает значение *b* в поток. void write(int b)

4.3.3.4. Обзор классов байтовых потоков

Кроме классов FileInputStream и FileOutputStream, существуют и другие классы байтовых потоков. Источником данных для них могут быть массивы, совокупность файлов и др. Перечислим некоторые из классов байтовых потоков.

Классы ByteArrayInputStream и ByteArrayOutputStream предназначены для чтения и записи значений соответственно в массив значений типа byte в виде потока данных. Данные для чтения и записи хранятся в буфере. При чтении специальный счетчик хранит позицию следующего считываемого значения. При записи буфер автоматически увеличивается для записи новых значений. Записанные в поток данные могут быть преобразованы в массив значений типа byte.

Классы DataInputStream и DataOutputStream предназначены для чтения и записи переменных базовых типов в поток ввода или вывода.

Классы CheckedInputStream и CheckedOutputStream позволяют вычислить проверочную сумму для считанных или записываемых данных для проверки того, были ли они искажены во время хранения или передачи по каналам связи.

Классы ObjectInputStream и ObjectOutputStream позволяют считывать и записывать внутреннее представление объектов и переменных базовых типов в виде последовательности байт. Процессы записи и чтения объектов называются *сериализацией* и *десериализацией* соответственно. Состояние программы можно охарактеризовать совокупностью созданных в ней объектов. Поэтому сериализацию и десериализацию используют для сохранения состояния программы.

Классы PipedInputStream и PipedOutputStream используются вместе друг с другом для передачи данных между потоками выпол-

нения (threads; связаны с многопоточностью, а не с рассматриваемыми в этом разделе потоками ввода и вывода (streams)) в одной программе. Сначала создается один поток, а затем создается другой, связываясь с первым. Через эту пару потоков данные передаются из одного метода в другой метод.

Класс PushbackInputStream имеет метод unread, возвращающий прочитанные данные обратно в поток. Предположим, что разделитель некоторых последовательностей данных должен всегда быть в начале последовательности. При получении этого разделителя цикл чтения данных может вернуть его обратно в поток и перейти к следующей итерации. На новой итерации разделитель будет считан в начале новой последовательности данных.

Класс SequenceInputStream обеспечивает чтение из нескольких потоков ввода так, как будто это один источник данных. Например, если объекту этого класса переданы два потока чтения данных из двух файлов, то сначала будут прочитаны полностью данные из первого файла, а после их окончания — полностью данные из второго файла. При этом переключение между файлами не остановит ввода данных.

Классы ZipInputStream и ZipOutputStream предназначены для извлечения и добавления файлов в архивы формата ZIP. При добавлении файлов можно задать степень сжатия.

Описание других классов байтовых потоков можно найти в документации к языку Java.

4.3.4. Символьные потоки

4.3.4.1. Родительские классы символьных потоков

Классы, описывающие символьные потоки ввода и вывода данных, являются наследниками абстрактных классов Reader и Writer соответственно.

4.3.4.2. Класс FileReader

Класс FileReader создает символьный поток чтения данных из файла.

Некоторые методы класса FileReader:

Метод	Описание и общий вид
close	Закрывает поток. void close()

Метод	Описание и общий вид
read	Возвращает считанный символ из потока или значение -1, если достигнут конец файла. int read() Читает из потока length символов в массив cbuf, начиная с символа offset. Возвращает количество считанных из потока символов или значение -1, если достигнут конец файла. int read(char[] cbuf, int offset, int length)
ready	Возвращает значение true, если доступны данные для чтения и метод read не заблокирует поток, и значение false в остальных случаях. boolean ready()

4.3.4.3. Класс FileWriter

Класс FileWriter создает символьный поток записи данных из файла.

Некоторые методы класса FileWriter:

Метод	Описание и общий вид
close	Закрывает поток. void close()
write	Записывает символ в поток. void write(int c) Записывает в поток length символов в массив cbuf, начиная с символа offset. void write (char[] cbuf, int offset, int length)

4.3.4.4. Буферизированные символьные потоки

Для ускорения операций чтения и записи данных, а следовательно, работы с потоками ввода и вывода используется промежуточный буфер.

Для организации буферизированных потоков используются классы BufferedReader и BufferedWriter. Эти классы используют те же методы для чтения и записи данных, что и классы FileReader и FileWriter.

Классы BufferedReader и BufferedWriter являются «обертками» для классов FileReader и FileWriter, изменяя их поведение (шаблон проектирования «Декоратор»):

```
BufferedReader in = new BufferedReader(new FileReader("inpdata.txt"));
BufferedWriter out = new BufferedWriter(new FileWriter("outpdata.txt"));
```

4.3.4.5. Обзор классов символьных потоков

Классы символьных потоков так же разнообразны по источникам данных, как и байтовые потоки. Рассмотрим некоторые из них.

Классы символьных потоков CharArrayReader, CharArrayWriter, PipedReader, PipedWriter, PushbackReader являются аналогами классов байтовых потоков (см. раздел 4.3.3.4).

Класс InputStreamReader преобразует байты, полученные с помощью байтового потока ввода, переданного через конструктор, в символы в определенной кодировке. Класс OutputStreamWriter выполняет обратные действия: преобразует символы в определенной кодировке в последовательность байт и записывает их с помощью байтового потока вывода.

Класс LineNumberReader, кроме буферизированного чтения символов, подсчитывает количество считанных строк. Разделителями строк считаются символ перевода строки («\n»), символ возврата каретки («\r») или их комбинация.

Класс PrintWriter предоставляет методы print, printf и println для вывода текста. Объект System.out, предназначенный для вывода текста на консоль (см. раздел 2.6.2), принадлежит классу PrintWriter.

Классы StringReader и StringWriter считывают символы из строки, как из потока ввода или вывода.

4.3.5. Пример чтения данных из текстового файла

Пример 4-03. Фрагмент чтения из текстового файла в строку textFromFile класса StringBuilder с обработкой исключений. Для чтения используется символьный поток FileReader, буферизированный потоком BufferedReader.

```
1  FileReader preReader = null;
2  BufferedReader reader = null;
3  String filename = "source.dat";
4  StringBuilder textFromFile = new StringBuilder("Text from file:\n");
5  int character;
6
7  try {
8      preReader = new FileReader(filename);
```



```
9      reader = new BufferedReader(preReader);
10     while ((character = reader.read()) != -1) {
11         textFromFile.append((char) character);
12     }
13 } catch (FileNotFoundException e) {
14     e.printStackTrace();
15 } catch (IOException e) {
16     e.printStackTrace();
17 } finally {
18     if (reader != null) {
19         try {
20           reader.close();
21         } catch (IOException e1) {
22           e1.printStackTrace();
23         }
24     }
25 }
26
27 String text = textFromFile.toString();
```

❑

4.3.6. Пример записи данных в бинарный файл

Пример 4-04. Фрагмент записи массива байт в файл. Для записи используется байтовый поток FileOutputStream.

```
1  String filename = "d:\\mydocuments\\123.txt";
2  byte[] dataForWriting = { 1, 2, 3, 4, 5, 6 };
3  FileOutputStream out = null;
4
5  try {
6      out = new FileOutputStream(filename);
7      out.write(dataForWriting);
8  } catch (FileNotFoundException e) {
9      e.printStackTrace();
10 } catch (IOException e) {
11     e.printStackTrace();
12 } finally {
13     if (out != null) {
```

```
14          try {
15            out.close();
16          } catch (IOException e1) {
17            e1.printStackTrace();
18          }
19      }
20  }
```

❑

4.4. Коллекции

4.4.1. Коллекции в языке Java и их виды

Массивы как наборы данных обладают рядом недостатков. Из массива невозможно удалить элемент, сдвинув оставшиеся элементы и уменьшив длину массива. Массив невозможно в процессе работы программы увеличивать, добавляя новые элементы, так как длина массива фиксирована и задается при его создании.

Чтобы устранить эти недостатки, в языке Java помимо массивов существуют наборы данных, называемые *коллекциями*. Основными видами коллекций являются следующие.

Список. Интерфейс List включает методы добавления, удаления и перебора элементов списка.

Множество. Интерфейс Set описывает математическое множество. Математическое множество — это набор неповторяющихся элементов.

Карта отображения. Классы, реализующие интерфейс Map, предназначены для хранения ключей и их значений. *Ключ* — это уникальный элемент, который позволяет найти соответствующее ему значение. Этим и определяется область применения карт отображений.

Интерфейсы List и Set являются подынтерфейсами интерфейса Collection.

Классы ArrayList, HashSet и HashMap реализуют интерфейсы List, Set и Map соответственно. Эти интерфейсы реализуют и другие классы, отличающиеся от перечисленных способом хранения наборов данных.

Коллекции позволяют хранить и обрабатывать объекты классов, но не значения переменных базовых типов. Для хранения значений

базовых типов используются соответствующие им классы-оболочки (см. раздел 2.4.6).

Нумерация элементов коллекций, как и массивов, начинается с нуля. Количество элементов в коллекции можно получить с помощью метода size, а не length, как у массивов.

Для сортировки, сравнения и поиска элементов коллекций может возникнуть необходимость переопределения в них методов equals и hashCode и реализации интерфейса Comparable.

4.4.2. Параметризация

Чтобы указать, объекты какого класса будут храниться в коллекции, необходимо указать этот класс при создании объекта коллекции.

Пример 4-05. Список класса ArrayList для хранения строк класса String создается следующим образом:

```
1 | List<String> text = new ArrayList<String>();
```

Обратите внимание, что класс ArrayList, реализующий интерфейс List, используется только при создании списка. Во всех остальных случаях необходимо использовать интерфейс (в данном случае List). Используя это правило, вы можете легко изменить реализацию списка другим классом в одной строке программы. ❑

В коллекции также можно хранить все подклассы класса, указанного при создании объекта коллекции.

В записи параметризованных типов используется метасимвол «?».

Запись List<?> означает, что этому типу соответствуют коллекции, параметризованные любым классом.

Запись List<? extends SomeClass> означает, что этому типу соответствуют коллекции, параметризованные классом SomeClass и любым его подклассом.

Запись List<? super SomeClass> означает, что этому типу соответствуют коллекции класса SomeClass или любого из его суперклассов, а также любых его подклассов.

4.4.3. Списки и интерфейс List

Интерфейс List задает поведение массива с возможностью добавления и удаления элементов.

Некоторые методы интерфейса List:

Метод	Описание и общий вид
add	Добавляет объект element в конец списка. Возвращает значение true, если объект был добавлен успешно, и значение false в противном случае. boolean add(E element) Добавляет объект element в позицию списка index. void add(int index, E element)
get	Возвращает элемент списка, находящийся в позиции index. E get(int index)
indexOf	Ищет первое вхождение объекта element в списке. Возвращает позицию объекта в списке или –1, если объект не был найден. int indexOf(Object element)
remove	Удаляет элемент в позиции index из списка. Возвращает удаляемый элемент. E remove(int index) Удаляет одно из вхождений элемента списка, равного объекту element. Возвращает значение true, если объект содержался в списке, и значение false в противном случае. boolean remove(Object element)
set	Заменяет значение элемента списка в позиции index объектом element. E set(int index, E element)
size	Возвращает количество элементов в списке. int size()

Интерфейс List реализуют классы ArrayList, LinkedList и др. Класс ArrayList реализует операции над списком с помощью массива ссылок, а класс LinkedList — с помощью двунаправленного списка. Операции добавления в конец, в начало или в середину, удаления произвольного элемента, доступа к элементу по его индексу для массива и двунаправленного списка требуют различных временных затрат (временной сложности). Это и определяет выбор класса списка.

4.4.4. Множества и интерфейс Set

Интерфейс Set включает методы, реализующие операции с элементами математического множества.

Некоторые методы интерфейса Set:

Метод	Описание и общий вид
add	Добавляет объект element в множество, если он там отсутствовал. Возвращает значение true, если объект отсутствовал в множестве, и значение false в противном случае. boolean add(E element)
contains	Возвращает значение true, если объект element содержится в множестве, и значение false в противном случае. boolean contains(Object element)
remove	Удаляет элемент множества, равный объекту element. Возвращает значение true, если объект содержался в множестве, и значение false в противном случае. boolean remove(Object element)
size	Возвращает количество элементов в множестве. int size()

Интерфейс Set реализуют классы HashSet, TreeSet и др., описывающие множество на основе различных структур данных.

4.4.5. Итератор

Для обхода элементов списка или множества можно использовать цикл перебора (см. раздел 2.9.3). Однако для решения этой задачи для коллекций существует специальный объект — *итератор*. Итератор обеспечивает доступ к элементам коллекции и их последовательный обход.

Чтобы получить итератор, необходимо вызвать метод iterator() у объекта списка или множества.

Пример 4-06. Итератор для перебора элементов списка может быть получен только после создания списка (строка 2).

```
1 | List<String> strings = new ArrayList<String>();
2 | Iterator<String> iterator = strings.iterator();
```

Для получения итератора для перебора элементов множества необходимо также вызвать метод iterator().

```
1 | Set<Integer> numbers = new HashSet<Integer>();
2 | Iterator<Integer> iterator = numbers.iterator();
```

❑

Интерфейс Iterator параметризуется тем же классом, что и список или множество.

Методы интерфейса Iterator:

Метод	Описание и общий вид
hasNext	Возвращает значение true, если существует элемент для следующей итерации, и значение false в противном случае. boolean hasNext()
next	Возвращает следующий элемент списка или множества. E next()
remove	Удаляет элемент списка или множества, который был последним возвращен итератором. void remove()

Пример 4-07. Типовой перебор элементов списка с помощью итератора (с выводом на консоль) имеет вид

```
1  List<String> text = new ArrayList<String>();
2
3  // text list initialization
4
5  Iterator<String> iterator = text.iterator();
6  while (iterator.hasNext()) {
7      String string = iterator.next();
8      System.out.println(string);
9  }
```

❑

4.4.6. Карты отображений и интерфейс Map

Интерфейс Map реализует карту отображения, элементы которой состоят из двух частей: ключа и соответствующего ему значения. Ключи элементов уникальны, а значения могут повторяться.

Объект класса, реализующего интерфейс Map, параметризуется двумя типами, в отличие от уже рассмотренных коллекций. Первый тип определяет тип ключа, а второй — тип значений.

Пример 4-08. Карта отображения на основе класса HashMap с ключом типа Integer и значением типа String.


```
1  Map<Integer, String> map = new HashMap<Integer, String>();
```

❑

Некоторые методы интерфейса Map:

Метод	Описание и общий вид
containsKey	Возвращает значение true, если карта отображения содержит ключ key, и значение false в противном случае. boolean containsKey(Object key)
containsValue	Возвращает значение true, если карта отображения содержит одно или более значений value, и значение false в противном случае. boolean containsValue(Object value)
get	Возвращает значение, соответствующее ключу key, или null, если ключ key отсутствует в карте отображения. Значение null также может быть возвращено, если оно соответствует ключу key. Чтобы определить, какой из случаев имеет место, используйте метод containsValue. V get(Object key)
keySet	Возвращает множество всех ключей карты отображения. Set<K> keySet()
put	Добавляет элемент с ключом key и значением value. Если элемент с таким ключом существовал, то его значение замещается новым. Возвращает предыдущее значение, соответствующее ключу, если элемент с таким ключом существовал в карте отображения, или null в противном случае. По поводу возвращаемого значения null см. также описание метода get. V put(K key, V value)
size	Возвращает количество элементов в карте отображения. int size()
values	Возвращает коллекцию значений карты отображения. Collection<V> values()

Интерфейс Map реализуют классы HashMap, TreeMap и др.

4.4.7. Класс Collections и его методы

Класс Arrays (см. раздел 2.9.5) содержит служебные методы для обработки массивов. Служебные методы для обработки коллекций собраны в классе Collections. Все методы класса Collections являются статическими.

Некоторые методы класса Collections:

Метод	Описание и общий вид
binarySearch	Возвращает номер позиции элемента element в списке list. Если элемент отсутствует в списке, то метод возвращает значение –1. Перед вызовом метода список должен быть предварительно отсортирован. static <T> int binarySearch(List<? extends Comparable<? super T>> list, T key) Аналогичен предыдущему методу. Сравнение элементов может осуществляться методами класса, реализующего интерфейс Comparator. static <T> int binarySearch(List<? extends T> list, T element, Comparator<? super T> c)
copy	Копирует все элементы из списка src в список dest. static <T> void copy(List<? super T> dest, List<? extends T> src)
disjoint	Возвращает значение true, если коллекции c1 и c2 не имеют одинаковых значений элементов, и значение false в противном случае. static boolean disjoint(Collection<?> c1, Collection<?> c2)
fill	Заменяет все элементы списка list значением элемента obj. static <T> void fill(List<? super T> list, T obj)
max	Возвращает значение максимального элемента коллекции coll. static <T extends Object & Comparable<? super T>> T max(Collection<? extends T> coll) Возвращает значение максимального элемента коллекции coll. Сравнение выполняется методами объекта comp, реализующего интерфейс Comparator. static <T> T max(Collection<? extends T> coll, Comparator<? super T> comp)

Метод	Описание и общий вид
min	Возвращает значение минимального элемента коллекции coll. static <T extends Object & Comparable<? super T>> T min(Collection<? extends T> coll) Возвращает значение минимального элемента коллекции coll. Сравнение выполняется методами объекта comp, реализующего интерфейс Comparator. static <T> T min(Collection<? extends T> coll, Comparator<? super T> comp)
reverse	Обращает последовательность элементов списка. static void reverse(List<?> list)
sort	Сортирует элементы списка в порядке по возрастанию. static <T extends Comparable<? super T>> void sort(List<T> list) Сортирует элементы списка. Порядок сортировки определяется методами объекта comp, реализующего интерфейс Comparator. static <T> void sort(List<T> list, Comparator<? super T> comp)

4.4.8. Преобразования массивов в коллекции и обратно

4.4.8.1. Преобразования коллекций в массивы

В интерфейсе Collection, наследниками которого являются интерфейсы List и Set, определен метод toArray, имеющий две перегруженные разновидности.

Метод	Описание и общий вид
toArray	Возвращает элементы коллекции в виде массива. Элементы массива являются объектами класса Object. Object[] toArray() Аналогичен предыдущему методу. Элементы массива являются объектами того же класса T, что и элементы коллекции. Массив array должен быть создан конструктором new. Размер массива array может быть любым, в том числе и равным нулю. Метод увеличивает размер массива array, если он меньше размера коллекции. Если размер массива array больше размера коллекции, то пустые элементы массива array заполняются значением null. <T> T[] toArray(T[] array)

Пример 4-09. Создадим список и преобразуем его в массив.

```
1  List<String> stringList = new ArrayList<String>();
2  stringList.add("The First");
3  stringList.add("The Second");
4  stringList.add("The Third");
5  System.out.println("List: " + stringList);
6
7  String[] stringArray = new String[stringList.size()];
8  stringArray = stringList.toArray(stringArray);
9  System.out.println("Array: " + Arrays.toString(stringArray));
```

На экран будут выведены значения списка и массива:

Вывод на экран
`List: [The First, The Second, The Third]`
`Array: [The First, The Second, The Third]`

❑

4.4.8.2. Преобразования массивов в коллекции

Для преобразования массивов в коллекции используется метод asList класса Arrays.

Метод	Описание и общий вид
asList	Преобразует массив a в список. Является методом с неизвестным количеством параметров (см. раздел 3.2.2.2). Поэтому с помощью этого метода можно создать список из последовательности объектов. `static <T> List<T> asList(T... a)`

Пример 4-10. Создадим массив и преобразуем его в список.

```
1  String[] stringArray = { "The First", "The Second", "The Third" };
2  System.out.println("Array: " + Arrays.toString(stringArray));
3
4  List<String> listFromArray = Arrays.asList(stringArray);
5  System.out.println("List from array: " + listFromArray);
6
7  List<String> listFromValues = Arrays.asList("The First", "The Second",
8      "The Third");
9  System.out.println("List from values: " + listFromValues);
```


Список может быть создан как из массива (строки 4–5), так и из последовательности объектов (строки 7–9).

На экран будет выведено:

Вывод на экран
`Array: [The First, The Second, The Third]`
`List from array: [The First, The Second, The Third]`
`List from values: [The First, The Second, The Third]`

❑

5. ПРАКТИЧЕСКИЕ РАБОТЫ

5.1. Практическая работа 1. Знакомство со средой разработки программ на языке Java

5.1.1. Цель работы

Знакомство со средой разработки программ на языке Java и получение основных навыков написания программ, редактирования их текста, сохранения, импорта, экспорта и запуска на выполнение проектов.

5.1.2. Задание

Прочитать главу 6 «Разработка программ в среде Eclipse» и выполнить приведенные в ней действия в среде разработки Eclipse.

5.2. Практическая работа 2. Строки

5.2.1. Цель работы

Изучение и получение навыков работы со следующими понятиями языка Java и приемами программирования:

- строки в Java;
- класс String и его методы;
- управляющие конструкции: условный оператор, операторы циклов;
- проектирование классов и методов;
- ввод данных с консоли; класс Scanner.

5.2.2. Задание

I. Выполнить ввод строк с клавиатуры и реализовать для каждой из них вариант задания. Другие входные данные также ввести с клавиатуры.

II. Вариант задания обработки строки оформить в виде отдельного метода. Входные параметры и выходные результаты должны иметь тип String, в том числе в вариантах, связанных с выводом строк на экран.

III. Метод main должен выполнять следующие действия:

- вводить строки с консоли, используя следующий фрагмент кода:

```
1  Scanner in = new Scanner(System.in);
2
3  System.out.print("Enter number of strings: n = ");
4  int stringCount = Integer.parseInt(in.next());
5  in.nextLine();
6
7  String[] stringArray = new String[stringCount];
8  for (int i = 0; i < stringArray.length; i++) {
9      System.out.printf("Enter %d. string:\n", i+1)
10     stringArray[i] = in.nextLine();
11 }
12
13 // Task realization
14
15 in.close();
```

- вызывать метод, реализующий вариант задания (см. п. II);
- выводить результаты работы метода или сообщения об ошибках.

IV. Программа должна выдавать результат или информационные сообщения при любых значениях входных данных. Для этого выполнить следующие проверки:

- является ли входная строка пустой;
- является ли слово пустым (в вариантах заданий, связанных с обработкой слов в слове).

Если проверка не была пройдена, то вывести соответствующее сообщение.

V. Каждое задание содержит условие, выполнение которого влечет определенные в задании действия. Если условие не было выполнено, то вывести соответствующее сообщение.

VI. Словом считать последовательность любых символов, кроме пробела. Разделителем слов считать пробел.

VII. В каждом варианте использовать метод trim.

VIII. Данные на экран выводить только в методе main.

Для выполнения практической работы прочитать разделы 1.1–2.8.

5.2.3. Варианты заданий

1. Если строка начинается с подстроки startSubstr, то удвоить в строке подстроку, начиная с позиции beginIndex длиной substrLength.
2. Удалить все двойные подряд символы в строке. Тройные, четверные и т.д. символы оставить без изменений.
3. Удалить символы левее первого пробела и правее последнего пробела.
4. Добавить после entry-го вхождения подстроки oldSubstr подстроку newSubstr.
5. Записать символы четных по счету слов строки в обратном порядке (обратить слова).
6. В каждом слове заменить все вхождения подстроки oldSubstr подстрокой newSubstr в случае, если слово заканчивается подстрокой endSubstr.
7. Вывести на экран слова, первая буква которых встречается в них еще хотя бы один раз.
8. Входная строка имеет формат <Фамилия><Пробел><Имя>. Преобразовать ее в формат <Инициал><Пробел><Фамилия>. В случае неверности формата входных данных выдать соответствующее сообщение.
9. Заменить два и более стоящих подряд одинаковых символов одним символом.
10. Вывести на экран слова, которые начинаются и заканчиваются одной подстрокой длиной substrLength, и количество таких слов.
11. У каждого слова в строке первую букву сделать прописной, а остальные буквы — строчными.
12. Выделить скобками слова, начинающиеся с той же буквы, что и предпоследнее слово в строке.
13. Удалить из строки все повторения последнего слова.
14. Вывести на экран слова, состоящие из одинаковых символов.
15. Вывести на экран содержащиеся в строке целые числа, состоящие только из четных цифр.
16. Для каждого слова вывести количество вхождений во всех введенных строках. Уже выведенные слова не повторять.
17. Вывести на экран слова, встречающиеся в строке один раз.
18. Напечатать все слова максимальной и минимальной длины. Повторы слов не выводить.

19. Разделить строку на части длиной не более substrLength с помощью символа перевода строки. Строку делить на части только по разделителю слов.
20. Дано слово word. Проверить, встречаются ли символы, составляющие слово word, в строке string более двух раз.
21. В строке записать в обратном порядке (обратить) каждое слово.

5.3. Практическая работа 3. Одномерные массивы

5.3.1. Цель работы

Изучение и получение навыков работы со следующими понятиями языка Java:

- управляющие конструкции: условный оператор, операторы циклов;
- одномерные массивы, их создание и обработка;
- проектирование классов и методов;
- методы с неизвестным количеством параметров;
- файлы ресурсов;
- метод toString класса Object.

5.3.2. Задание

I. Создать класс А, соответствующий следующим требованиям:

- содержать метод или методы, реализующие вариант задания;
- параметрами конструктора класса должны быть элементы массива, количество которых заранее неизвестно;
- тип элементов массива должен быть int;
- содержать переопределенный метод toString;
- не содержать вызовы методов ввода и вывода данных.

II. Создать класс Б, соответствующий следующим требованиям:

- содержать метод main;
- создавать объект класса А; значения элементов массива задать в качестве параметров конструктора;
- вызывать метод объекта класса А, реализующий вариант задания;
- выводить исходный массив;
- выводить результаты обработки массива.

III. Классы, поля, методы, переменные должны иметь названия, соответствующие их назначению в программе.

IV. Данные на экран выводить только в методе main класса Б.

V. Все текстовые константы хранить в файлах ресурсов. Доступ к файлам ресурсов осуществлять с помощью класса Resourcer (см. раздел 4.1.4).

VI. При накоплении строк использовать класс StringBuilder.

Для выполнения практической работы прочитать разделы 2.9, 4.1.

5.3.3. Пример выполнения задания

Задание. Определить сумму элементов одномерного массива с положительными значениями.

Класс OneDimArrayExample, реализующий задание:

```
1  package com.prutzkow.onedimarrayexample;
2
3  import java.util.Arrays;
4
5  public class OneDimArrayExample {
6      private int[] arrayBody;
7
8      public OneDimArrayExample(int... args) {
9          this.arrayBody = new int [args.length];
10         setArrayElements(args);
11     }
12
13     public final void setArrayElements(int... args) {
14         int length = args.length;
15         for (int i = 0; i < length; i++) {
16           this.arrayBody[i] = args[i];
17         }
18     }
19
20     public int sumArrayPositiveElements() {
21         int summ = 0;
22         int length = this.arrayBody.length;
23         for (int i = 0; i < length; i++) {
24           if (this.arrayBody[i]>0) {
25           summ += this.arrayBody[i];
26           }
27         }
28         return summ;
29     }
30
31     @Override
32     public String toString() {
33         return Arrays.toString(this.arrayBody);
34     }
35 }
```

Класс OneDimArrayRunner, создающий объект класса OneDimArrayExample и вызывающий его методы:

```
1  package com.prutzkow.onedimarrayrunner;
2
3  import com.prutzkow.onedimarrayexample.OneDimArrayExample;
4
5  public class OneDimArrayRunner {
6      public static void main(String[] args) {
7          OneDimArrayExample arrayObject =
8            new OneDimArrayExample(1, 2, 3);
9          String result = "Origin Array\n";
10         result += arrayObject + "\n";
11         result += "Summ of Array Positive Elements = "
12           + arrayObject.sumArrayPositiveElements();
13         System.out.println(result);
14     }
15 }
```

5.3.4. Варианты заданий

1. Найти первый элемент массива с нечетным значением и последний элемент с четным значением, вывести их индексы и поменять их значения местами. В случае отсутствия нечетных или четных элементов вывести соответствующее сообщение.

2. Поменять местами одну и другую половины массива. При нечетном размере массива средний элемент оставить на месте.

3. Проверить в массиве наличие элементов с четными значениями. Если они присутствуют в массиве, то значения COUNT таких

последних элементов умножить на MULTIPLICATOR. Если таких значений меньше COUNT, то вывести соответствующее сообщение.

4. Поменять местами в массиве значения 1-го и 2-го, 3-го и 4-го и т.д. элементов. Подсчитать, в скольких парах первоначально левый элемент был меньше правого.

5. Найти MAX_NUM-е абсолютное значение среди элементов массива. Значение MAX_NUM = 1 — максимальное значение элементов массива, MAX_NUM = 2 — максимальное значение среди элементов со значениями, меньшими максимального, и т.д. В случае отсутствия такого элемента вывести соответствующее сообщение.

6. Определить, является ли последовательность значений элементов массива возрастающей или убывающей. Вывести соответствующее сообщение.

7. Найти сумму положительных значений элементов, расположенных левее элемента с минимальным значением. Если такие элементы в массиве отсутствуют, вывести соответствующее сообщение.

8. Вычесть значение SUBTRAHEND из отрицательных значений элементов, находящихся правее ZERO_NUM-го элемента с нулевым значением.

9. Вывести на экран значения элементов массива. Повторы уже выведенных значений на экран не выводить.

10. Вычислить среднее арифметическое элементов с ненулевыми значениями, имеющих нечетные индексы. Привести среднее арифметическое к целому типу любым способом. Вычесть полученное целое значение из всех элементов массива с четными значениями.

11. Подсчитать сумму элементов, значение которых превышает удвоенное значение минимума. При реализации задачи обязательно использовать операторы цикла for и while или do-while.

12. Подсчитать количество элементов, значения которых больше значений каждого из следующих за ним COUNT элементов. Элементы, за которыми не следуют COUNT элементов, не обрабатывать.

13. Подсчитать сумму элементов с четными значениями и произведение элементов с нечетными значениями. Если сумма элементов с четными значениями больше, то обнулить элементы с положительными значениями. Если произведение элементов с нечетными значениями больше, то обнулить элементы с отрицательными значениями. В случае равенства суммы и произведения вывести соответствующее сообщение.

14. Подсчитать количество элементов, значения которых лежат в диапазоне от LOW_LIMIT до HI_LIMIT. Если среднее арифметическое значений элементов, которые лежат в этом диапазоне, превышает количество таких элементов, то обнулить элементы с положительными значениями.

15. Присвоить всем элементам, которые находятся между элементами с максимальным и минимальным значениями, значение разности максимального и минимального значений. В случае непосредственного соседства элементов с минимальным и максимальным значениями вывести соответствующие сообщения. Все элементы массива имеют различные значения.

16. Удалить из массива элемент с индексом INDEX, считая, что первый элемент имеет индекс 1. Если значение INDEX — четное, то сдвинуть элементы справа от удаленного влево на одну позицию. Если значение INDEX — нечетное, то сдвинуть элементы слева от удаленного вправо на одну позицию. Дополнить массив значением ADDING_VALUE. В случае некорректного значения INDEX выдать соответствующее сообщение.

17. Изменить порядок следования значений элементов на обратный, если число элементов с положительным значением больше числа элементов с отрицательными и нулевыми значениями.

18. Подсчитать произведение значений элементов, находящихся между 1-м и 2-м отрицательными элементами. Если отрицательных чисел в массиве меньше двух или первый и второй отрицательные элементы находятся рядом, то вывести соответствующие сообщения.

19. Найти среди значений элементов массива, которые встречаются более TIMES раз, максимальное значение и обнулить все его вхождения в массиве.

20. Проверить, равна ли последовательность значений элементов массива при их переборе с начала к концу такой же последовательности при переборе с конца к началу. Пример такой последовательности: 2, 7, 9, 7, 2.

5.4. Практическая работа 4. Классы. Двумерные массивы

5.4.1. Цель работы

Изучение и получение навыков работы со следующими понятиями языка Java и приемами программирования:

- двумерные массивы с переменной длиной строк;
- наследование;
- переопределение и перегрузка методов;

- проектирование классов и методов;
- ключевые слова this и super;
- подготовка программной документации;
- метод toString класса Object.

5.4.2. Задание

I. Создать проект для выполнения задания по варианту.

II. Подключить к проекту библиотеку arrayutils.jar (предоставляется преподавателем) с исходными классами по работе с массивами и документацию к ним, предоставленные преподавателем. Исходные классы представлены в разделе 5.4.3. Последовательность действий по подключению библиотеки к проекту приведена в разделе 5.8.2.

III. Создать класс А, удовлетворяющий следующим требованиям:

- должен являться подклассом суперкласса, указанного в следующей таблице в соответствии с номером варианта;

Номера вариантов	Суперкласс
1, 2, 5, 6, 7, 12, 13, 18	Matrix
3, 4, 8, 9, 10, 11, 14, 15, 16, 17, 18, 19, 20, 21	TwoDimArray

- один из методов класса А должен реализовывать обработку двумерного массива в соответствии с вариантом задания. В случае целесообразности оформить решение варианта задания в виде нескольких методов;
- перегрузить метод fill класса TwoDimArray для заполнения поля arrayBody случайными числами с помощью метода Math.random. Метод fill должен иметь два параметра: нижнюю и верхнюю границы диапазона значений генерируемых случайных чисел;
- переопределить метод toString, добавив к выводу элементов массива данные из следующей таблицы в соответствии с вариантом задания. Данные снабдить поясняющей надписью.

Номера вариантов	Добавляемые к выводу данные
1, 6, 11, 16, 21	Количество элементов в массиве
2, 7, 12, 17	Сумма первого и последнего элементов массива
3, 8, 13, 18	Количество элементов первой строки
4, 9, 14, 19	Количество элементов первого столбца
5, 10, 15, 20	Количество элементов последней строки

IV. Создать класс Б, соответствующий следующим требованиям:

- содержать метод main;
- создавать объект класса А;
- инициализировать элементы массива класса А случайными значениями с помощью перегруженного метода fill (см. п. V);
- вызывать метод объекта класса А, реализующий вариант задания;
- выводить исходный массив;
- выводить результаты обработки массива;
- включать константы, значения которых содержат исходные данные, необходимые для работы метода, реализующего вариант задания.

V. Классы, поля, методы, переменные должны иметь названия, соответствующие их назначению в программе.

VI. Все текстовые константы хранить в файлах ресурсов. Для доступа к файлам ресурсов использовать класс Resourcer.

VII. При накоплении строк использовать класс StringBuilder.

VIII. Добавить к константным полям, методам и конструкторам комментарии JavaDoc.

Для выполнения практической работы прочитать разделы 3.1—3.2.

5.4.3. Исходные классы для выполнения задания

Исходный класс TwoDimArray:

```
1   package com.prutzkow.twodimarray;
2
3   import java.util.Arrays;
4
5   public class TwoDimArray {
6
7       public final String FORMAT = "%-5d";
8
9       protected int arrayBody[][] = null;
10
11      public TwoDimArray(int... sizes) throws IllegalArgumentException {
12          this.createArrayBody(sizes);
13      }
14
15      public int getRowCount() {
16          return arrayBody.length;
17      }
18
```

```
    public int getRowLength(int row) {
        return arrayBody[row].length;
    }

    public void createArrayBody(int... sizes)
        throws IllegalArgumentException {
        this.arrayBody = new int[sizes.length][];
        for (int i = 0; i < this.getRowCount(); i++) {
          this.arrayBody[i] = new int[sizes[i]];
        }
        this.fill(0);
    }

    public void fill(int constant) {
        for (int i = 0; i < this.getRowCount(); i++) {
          Arrays.fill(this.arrayBody[i], constant);
        }
    }

    @Override
    public String toString() {
        String s = "";
        for (int i = 0; i < this.getRowCount(); i++) {
          for (int j = 0; j < this.getRowLength(i); j++) {
                s += String.format(this.FORMAT, this.arrayBody[i][j]);
          }
          s += "\n";
        }
        return s;
    }
}
```

Исходный класс Matrix — подкласс класса TwoDimArray:

```
package com.prutzkow.twodimarray;

public class Matrix extends TwoDimArray {

    public Matrix(int sizeX, int sizeY) throws IllegalArgumentException {
        int[] sizes = new int[sizeX];
        Arrays.fill(sizes, sizeY);
        super.createArrayBody(sizes);
    }

    public int getColCount() {
        return arrayBody[0].length;
    }
}
```

5.4.4. *Пример выполнения задания*

Задание. Найти максимальное значение элемента в двумерном массиве.

```
package com.prutzkow.twodimarray;

public class MaxElementArray extends TwoDimArray {

    public MaxElementArray(int... sizes)
        throws IllegalArgumentException {
        super(sizes);
    }

    public int getMaxValue() {
        int max = this.arrayBody[0][0];
        for (int i = 0; i < super.getRowCount(); i++) {
          for (int j = 0; j < super.getRowLength(i); j++) {
            if (max < this.arrayBody[i][j]) {
              max = this.arrayBody[i][j];
            }
          }
        }
        return max;
    }
}
```

5.4.5. Варианты заданий

1. Заменить элементы главной диагонали квадратной матрицы суммами элементов столбцов.

2. Отсортировать элементы тех столбцов, в которых элементов с отрицательными значениями больше, чем элементов с положительными значениями.

3. Вывести номера строк, в которых элементы с максимальным значением этой строки встречаются более одного раза.

4. Вывести номера строк, в которых встречается самая длинная последовательность элементов с одинаковыми значениями.

5. Подсчитать суммы значений, находящихся выше и ниже главной диагонали квадратной матрицы. Вычесть из большего значения суммы меньшее значение. Если суммы значений равны, то вывести соответствующее сообщение.

6. Найти в квадратной матрице строку с номером num, элементы которой равны элементам столбца с тем же номером num. Вывести номера таких строк или сообщение об их отсутствии.

7. Переставить элементы столбца COL_1 и COL_2. Столбцы с одинаковыми значениями не менять.

8. Переставить элементы строк ROW_1 и ROW_2. Если размеры строк не равны между собой, то лишние элементы опустить, а недостающие заполнить константой MISSING_VALUE.

9. Преобразовать элементы строки массива так, чтобы сначала располагались элементы с ненулевыми значениями без изменения их порядка, а затем — с нулевыми.

10. Вывести номер строки (или номера строк) с максимальной суммой значений элементов, расположенных между первым и вторым элементами со значением CONST_VALUE. Если в строке количество элементов со значением CONST_VALUE меньше двух, то считать сумму элементов равной нулю.

11. Записать в строку ROW_1 значения элементов, встречающихся одновременно в строках ROW_2 и ROW_3. Остальные элементы строки ROW_1 обнулить. ROW_1 ≠ ROW_2; ROW_1 ≠ ROW_3.

12. Записать в столбец COL_1 значения элементов, встречающихся только в одном из столбцов COL_2 и COL_3. Остальные элементы столбца COL_1 обнулить. COL_1 ≠ COL_2; COL_1 ≠ COL_3.

13. Среди всех элементов с минимальными значениями массива определить координаты минимума, находящегося ближе всех к элементу, находящемуся в последней строке последнего столбца. При выводе массива на экран вывести также номера строк и столбцов.

14. Заполнить строку и столбец, содержащие первый элемент с максимальным значением всего массива, значениями ROW_COL_VALUE. Если в строке отсутствует элемент необходимого столбца, то строку не обрабатывать.

15. Подсчитать сумму элементов строк, в которых содержится максимальное значение массива.

16. Проверить, встречаются ли элементы строки ROW в других строках массива в том же порядке, но не обязательно подряд. Например: элементы строки 5, 3, 2, 7, 9 содержатся в строке 5, 2, 3, 1, 1, 2, 7, 8, 8, 9. Если такие строки существуют, то утроить значения элементов, которые не содержатся в строке ROW, иначе — вывести сообщение.

17. В каждой строке массива изменить на противоположный знак элемент строки, у которого разница суммы элементов, расположенных левее этого элемента, и суммы элементов, расположенных правее, минимальна. Рассматривать все элементы строки, кроме первого и последнего. Например, для строки 2, 5, 3, 9, 7, 4 необходимо изменить знак у элемента со значением 9.

18. Найти столбцы, в которых максимальное и минимальное значения элементов отличаются от среднего арифметического элементов этого столбца не более чем на PERCENT_DIFFERENCE процентов.

19. В каждой строке массива заменить значения, расположенные между первым и вторым по значению минимальными элементами, значением REPLACING_VALUE. Если в строке несколько одинаковых минимальных по значению элементов, то взять первый и второй минимальные элементы по порядку.

20. Заменить элементы строки ROW количеством вхождений этих значений в массив.

21. В каждом столбце записать элементы с четными значениями в порядке убывания. В элементы с нечетными значениями записать значение REPLACING_VALUE.

5.5. Практическая работа 5. Иерархии классов

5.5.1. Цель работы

Изучение и получение навыков работы со следующими понятиями языка Java и приемами программирования:

- проектирование классов и методов;
- проектирование иерархии классов;

- отношения между классами: наследование и включение;
- методы класса Arrays: sort и binarySearch;
- определение порядка сортировки.

5.5.2. Задание

I. Спроектировать в заданной предметной области иерархию классов. В иерархии должен быть базовый абстрактный класс и не менее трех его подклассов второго уровня. В базовом классе должно быть не менее двух полей, не менее двух методов и конструктор. Одно из полей должно быть объектом пользовательского (нестандартного) класса. В методах выполнить проверку параметров на принадлежность диапазону допустимых значений.

II. При сортировке использовать метод compareTo.

III. Проинициализировать все объекты и переменные, необходимые для работы программы, значениями в отдельном классе. Все текстовые сообщения хранить в файлах ресурсов. Доступ к файлам ресурсов осуществлять с помощью класса Resourcer (см. раздел 4.1.4).

Для выполнения практической работы прочитать разделы 3.3–3.6.2.

5.5.3. Варианты заданий

Взяты варианты заданий из учебного пособия [2] с некоторыми изменениями.

1. Цветочница. Определить иерархию цветов. Создать несколько объектов-цветов. Собрать букет (используя аксессуары) с определением его стоимости. Провести сортировку цветов в букете на основе уровня свежести. Найти цветок в букете.

2. Новогодний подарок. Определить иерархию конфет и прочих сладостей. Создать несколько объектов-конфет. Собрать детский подарок с определением его веса. Провести сортировку конфет в подарке на основе одного из параметров. Найти конфету в подарке.

3. Домашние электроприборы. Определить иерархию электроприборов. Включить некоторые в розетку. Подсчитать потребляемую мощность. Провести сортировку приборов в квартире на основе мощности. Найти прибор в квартире.

4. Шеф-повар. Определить иерархию овощей. Сделать салат. Подсчитать калорийность. Провести сортировку овощей для салата на основе одного из параметров. Найти овощи в салате.

5. Звукозапись. Определить иерархию музыкальных композиций. Записать на диск сборку. Подсчитать продолжительность. Провести перестановку композиций диска на основе принадлежности к стилю. Найти композицию.

6. Камни. Определить иерархию драгоценных и полудрагоценных камней. Отобрать камни для ожерелья. Подсчитать общий вес (в каратах) и стоимость. Провести сортировку камней ожерелья на основе ценности. Найти камни в ожерелье.

7. Мотоциклист. Определить иерархию амуниции. Экипировать мотоциклиста. Подсчитать стоимость. Провести сортировку амуниции на основе веса. Найти элементы амуниции.

8. Железнодорожный транспорт. Определить иерархию подвижного состава (пассажирских, грузовых, почтовых вагонов и локомотивов). Создать пассажирский поезд. Подсчитать общую вместимость пассажиров и багажа. Провести сортировку вагонов поезда на основе уровня комфортности. Найти в поезде определенный вагон.

9. Авиакомпания. Определить иерархию самолетов и вертолетов (пассажирских и грузовых). Создать авиакомпанию. Проверить, можно ли перевести заданное количество пассажиров и груза флотом авиакомпании. Провести сортировку самолетов компании по дальности полета. Найти определенный самолет в компании.

10. Таксопарк. Определить иерархию такси (грузовых и легковых автомобилей, малых, средних и больших автобусов). Создать таксопарк. Подсчитать стоимость автопарка. Провести сортировку автомобилей парка по расходу топлива. Найти в таксопарке определенный автомобиль.

11. Страхование. Определить иерархию страховых обязательств. Собрать из обязательств дериватив. Подсчитать стоимость. Провести сортировку обязательств в деривативе на основе уменьшения степени риска. Найти обязательство в деривативе.

12. Мобильная связь. Определить иерархию тарифов мобильной компании. Создать список тарифов компании. Подсчитать общую численность клиентов. Провести сортировку тарифов на основе размера абонентской платы. Найти тариф среди тарифов компании.

13. Фургон кофе. Загрузить фургон определенного объема грузом на определенную сумму из различных сортов кофе, находящихся к тому же в разных физических состояниях (зерно, молотый, растворимый в банках и пакетиках). Учитывать объем кофе вместе с упаковкой. Провести сортировку товаров на основе соотношения цены и веса. Найти в фургоне товар.

14. Игровая комната. Подготовить игровую комнату для детей разных возрастных групп. Игрушек должно быть фиксированное количество в пределах выделенной суммы денег. Должны встречаться

игрушки родственных групп: маленькие, средние и большие машины, куклы, мячи, кубики. Провести сортировку игрушек в комнате по одному из параметров. Найти игрушки в комнате.

15. Налоги. Определить виды налогов: налоговые выплаты физического лица за год с учетом доходов с основного и дополнительного мест работы, авторских вознаграждений, продажи имущества, получения в подарок денежных сумм и имущества, переводов из-за границы, льгот на детей и материальной помощи. Подсчитать сумму. Провести сортировку налогов по сумме. Найти налог среди вмененных налогов.

16. Счета. Клиент может иметь несколько счетов в банке. Учитывать возможность блокировки/разблокировки счета. Реализовать поиск и сортировку счетов. Вычисление общей суммы по счетам. Вычисление суммы по всем счетам, имеющим положительный и отрицательный балансы отдельно.

17. Туристические путевки. Сформировать набор предложений клиенту по выбору туристической путевки различного типа (отдых, экскурсии, лечение, шопинг, круиз и т. д.) для оптимального выбора. Учитывать возможность выбора транспорта, питания и числа дней. Реализовать сортировку путевок. Найти определенную путевку среди всех путевок.

18. Кредиты. Сформировать набор предложений клиенту по целевым кредитам различных банков для оптимального выбора. Учитывать возможность досрочного погашения кредита и/или увеличения кредитной линии. Реализовать выбор и поиск кредита.

19. Гостиница. Определить иерархию гостиничных номеров. Создать гостиницу. Подсчитать общую вместимость гостиницы. Провести сортировку номеров по стоимости проживания. Найти гостиничный номер.

20. Мебельный магазин. Определить иерархию мебели. Создать мебельный магазин. Подсчитать общую стоимость мебели. Провести сортировку мебели по ее типу и цвету. Найти предметы мебели.

5.6. Практическая работа 6. Потоки ввода и вывода. Исключения

5.6.1. Цель работы

Изучение и получение навыков работы со следующими понятиями языка Java и приемами программирования:

- исключения и способы их обработки;

- символьные и байтовые потоки ввода и вывода;
- работа с файловой системой;
- форматированный вывод;
- перечисления;
- ввод данных с консоли; класс Scanner.

5.6.2. Задание

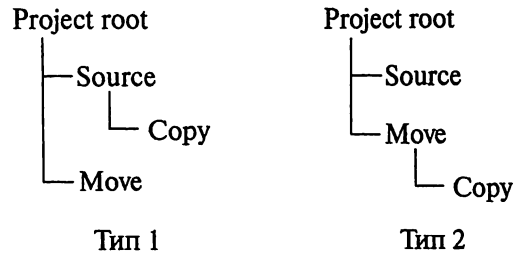
I. Создать класс A, который описывает сущность, имеющую три параметра согласно варианту задания. Для описания третьего параметра использовать перечисление. Класс A должен содержать переопределенные методы toString, equals, hashCode, compareTo.

II. Программа должна выполнить следующие действия:

1) создать структуру папок в папке программного проекта в соответствии с номером варианта;

Номера вариантов	Тип структуры папок
1, 3, 5, 7, 9, 11, 13, 15, 17, 19	1
2, 4, 6, 8, 10, 12, 14, 16, 18	2

Типы структур папок:



Тип 1 Тип 2

2) создать массив объектов класса A и его проинициализировать; инициализацию элементов массива поместить в отдельный метод класса-инициализатора; первый параметр сущности, описываемой классом A, считать уникальным; значения второго и третьего параметров сущности обязательно должны повторяться;

3) создать файл в папке Source и заполнить его данными из массива, указанного в п. 2; формат файла продумать самостоятельно;

4) создать файл в папке Copy и скопировать в него данные из файла, указанного в п. 3; имя создаваемого файла должно быть таким же, как и у файла, указанного в п. 3; расширение создаваемого файла должно быть «.bak»;

5) переместить файл из папки Source в папку Move; перед перемещением выдать запрос пользователю на выполнение этого действия;

6) прочитать содержимое файлов, указанных в пп. 4 и 5, и записать их в массивы объектов класса A;

7) вывести на экран массивы из пп. 2 и 6 в таблице, используя форматированный вывод; выровнять элементы в ячейках таблицы по левому краю;

8) сравнить значения элементов массивов из пп. 2 и 6 и вывести результаты сравнения.

III. Программа, реализующая вариант задания, должна обрабатывать и по возможности исправлять все ошибочные ситуации, возникающие при ее работе. Для обработки ошибочных ситуаций использовать исключения. При возникновении исключения выводить текстовое сообщение об ошибке; метод printStackTrace не вызывать. В случае возникновения ошибки работы с файлом в сообщении должны быть указаны полный путь и имя файла. Обработчик исключений должен быть размещен в одном методе.

IV. После создания файлов и папок выводить на консоль надпись с названием выполненной операции и с полными абсолютными именами файлов и папок, например «File C:\Java\data.txt has been created». Накопление строк при выводе на консоль не использовать.

V. Метод копирования файлов сделать универсальным для любых типов файлов.

VI. Все текстовые константы хранить в файлах ресурсов. В файле ресурсов определить значения следующих ключей:

- files.folder.source.name = Source — имя папки Source;
- files.folder.copy.name = Copy — имя папки Copy;
- files.folder.move.name = Move — имя папки Move;
- files.file.data.name — имя файла из п. II.3;
- files.file.backup.extension = bak — расширение файла из п. II.4.

Для выполнения практической работы прочитать разделы 2.7.4, 3.8, 4.2, 4.3.

5.6.3. Варианты заданий

Первый параметр считать уникальным. Значения второго и третьего параметров должны повторяться.

1. Методические указания: номер; автор; дисциплина.
2. Интернет-ресурсы: адрес в Интернете; страна — место размещения серверов; категория.
3. Моряки: ФИО; название корабля; должность.

4. Стипендиальная ведомость: ФИО; размер стипендии; номер группы.
5. Бытовая техника: марка; модель; тип устройства.
6. Пассажирские поезда: номер; исходный пункт; пункт назначения.
7. Командная спартакиада: номер спортсмена; количество очков; команда.
8. Книги: код ISBN; автор; жанр.
9. Маршрутные такси: государственный регистрационный номер; владелец (физическое или юридическое лицо); номер маршрута.
10. Местные FM-радиостанции: частота вещания; название радиостанции; жанр программ.
11. Флора: латинское название; семейство; род.
12. Офисные принадлежности: инвентарный номер; наименование; отдел, к которому принадлежность приписана.
13. Транспортная компания: номер груза; отправитель; тип транспортного средства, перевозящего груз.
14. Гостиница: номер комнаты; количество мест; класс номера (экономный, бизнес-класс, президентский).
15. Классы языка Java: название класса; имя и фамилия разработчика; название проекта.
16. Мебель на выставке: название модели; номер зала размещения; вид (диван, шкаф и т.п.).
17. Коммерческие предприятия: название; населенный пункт, в котором предприятие зарегистрировано; форма собственности (акционерное общество (АО), публичное АО (ПАО), общество с ограниченной ответственностью (ООО)).
18. Сетевое оборудование: MAC-адрес; описание (например, «Сервер в комнате 16»); тип.
19. Периодическое издание: название; учредитель; вид.

5.7. Практическая работа 7. Коллекции

5.7.1. Цель работы

Изучение и получение навыков работы со следующими понятиями языка Java и приемами программирования:

- коллекции;
- методы класса Object: toString, equals, hashCode;
- определение порядка сортировки;

- форматированный вывод;
- перечисления.

5.7.2. Задание

I. Использовать класс A из предыдущей практической работы (см. раздел 5.6.3).

II. Выполнить следующие действия.

1. Создать список объектов класса A и отсортировать его двумя способами:

- по умолчанию;
- по второму и третьему параметрам одновременно.

При сортировке использовать интерфейс Comparable.

2. Вывести список значений второго параметра без повторов.

3. Удалить элементы с заданным значением третьего параметра из множества объектов с помощью методов итератора.

4. Найти элемент коллекции по значению первого параметра. Если элемент с таким значением в коллекции отсутствует, то вывести сообщение; найти элементы по значению, которое присутствует в коллекции, и по значению, которое отсутствует в коллекции.

III. Цикл с параметром применять, только если использование цикла перебора элементов for или итератора невозможно.

IV. Список выводить в таблице; использовать методы, разработанные вами при выполнении практической работы из раздела 5.6.

При выполнении задания обязательно использовать интерфейсы List, Set, Map и реализующих их классы.

Для выполнения практической работы прочитать раздел 4.4.

5.7.3. Варианты заданий

Сущность, описываемую классом A, взять из предыдущей практической работы (см. раздел 5.6.3).

5.8. Приложение к практическим работам

5.8.1. Памятка при выполнении практических работ

I. Соблюдайте принципы проектирования программы.

Пиши меньше строк программы, максимально используя возможности Java!

Добавляй, но не изменяй!

II. Соблюдайте принципы проектирования классов и методов.

Каждый класс — одна задача!

Каждый метод — одно действие!

III. Делите программу на как можно меньшие части. Это упрощает ее отладку и понимание.

IV. Ускорьте работу в среде Eclipse с помощью сочетаний клавиш:

- ALT + SHIFT + R — переименовать имена проектов, пакетов, классов, методов, полей, переменных;
- CTRL + / — закомментировать/раскомментировать строку программы;
- CTRL + SHIFT + F — отформатировать текст класса или выделенного фрагмента.

V. Name code elements only in English, but not transliteratsiej. Русский язык используйте только в комментариях, but not in input-output messages and javadocs.

VI. Создавайте в среде Eclipse методы и конструкторы автоматически с помощью пункта контекстного меню *Source*.

VII. Самый лучший знаток языка Java и приемов программирования — это любая поисковая система в сети Интернет.

5.8.2. Как включить пользовательскую библиотеку в проект

При разработке приложений требуется добавлять к проекту пользовательские библиотеки, реализующие определенную функциональность. Использование дополнительных библиотек сокращает время разработки приложения.

Подключить библиотеку к проекту можно несколькими способами.

I. Подключить библиотеку в окне *Java Build Path* настроек проекта. В этом случае в конфигурационный файл проекта подключается ссылка на файл с пользовательской библиотекой, но не сам файл. При развертывании проекта на другом компьютере связь с библиотекой может быть потеряна и ее необходимо будет восстанавливать вручную.

II. Изменить системную переменную, хранящую пути к библиотекам, или добавить библиотеку в одну из папок, пути к которым записаны в эту переменную. Этот способ также требует ручных действий при каждом новом развертывании проекта.

III. Включить библиотеку в проект. Этот способ позволяет хранить файл с пользовательской библиотекой внутри архива проекта и является наиболее предпочтительным.

Чтобы включить пользовательскую библиотеку в проект, выполните следующие действия.

1. В среде разработки Eclipse выберите пункт меню *File | New | Java Project*, чтобы создать новый проект.

2. Выберите созданный проект щелчком мыши и выберите пункт меню *File | New | Folder*. В результате появится окно *New Folder*.

3. В поле *Folder Name* введите имя папки, например lib, и нажмите кнопку *Finish*. В результате в проекте появится новая папка lib.

4. Откройте Проводник операционной системы *Windows* и найдите нужную пользовательскую библиотеку.

5. В Проводнике щелкните правой кнопкой мыши на найденной пользовательской библиотеке. В результате появится контекстное меню.

6. В контекстном меню выберите пункт *Копировать* (*Copy*). В результате пользовательская библиотека будет скопирована в буфер обмена Windows.

7. Вернитесь в среду Eclipse.

8. Щелкните правой кнопкой мыши на созданной папке lib. В результате появится контекстное меню.

9. В контекстном меню выберите пункт *Paste*. В результате пользовательская библиотека будет вставлена в папку lib.

10. Щелкните дважды левой кнопкой мыши по папке lib. В результате раскроется ее содержимое вместе с добавленной пользовательской библиотекой.

11. Щелкните правой кнопкой мыши на пользовательской библиотеке. В результате появится контекстное меню.

12. В контекстном меню выберите пункт *Build Path | Add to Build Path*. В результате в проекте появится папка Referenced Libraries с пользовательской библиотекой.

В результате библиотека будет включена внутрь проекта.

6. РАЗРАБОТКА ПРОГРАММ В СРЕДЕ ECLIPSE

6.1. Основные сведения о среде разработки Eclipse

Мы будем использовать Eclipse как среду для разработки приложений на языке Java. Для этого среда разработки Eclipse имеет следующие возможности:

- создание, отладка, экспорт и импорт проектов приложений;
- компиляция и запуск приложений с помощью установленной на компьютере среды выполнения Java;
- дополнительные возможности при редактировании кода: подсветка синтаксиса; автодополнение; настройка форматирования кода;
- расширение возможностей среды с помощью подключаемых модулей.

Eclipse — не просто среда разработки, но и платформа для создания сред разработки приложений на различных языках программирования. На момент написания этих методических указаний существовали в Eclipse среды разработки приложений на языках C/C++, Python, PHP, Ruby и др.

Среда разработки Eclipse бесплатна для скачивания и использования.

6.2. Начало работы в среде разработки Eclipse

6.2.1. Порядок установки на компьютер

Здесь и далее рассматривается среда Eclipse версии Indigo Service Release 2 для разработки приложений на языке программирования Java.

Чтобы установить среду Eclipse для разработки приложений, выполните следующие действия.

1. Скачайте и установите Java Development Kit (JDK) — бесплатно распространяемый компанией «Oracle» комплект разработчика приложений на языке Java. Рекомендуется установить JDK той же раз-

рядности (32-битный или 64-битный), что и операционная система, установленная на вашем компьютере.

ЗАЧЕМ ЭТО НУЖНО?

Java Development Kit является основным для разработчика компонентом среды Java, включающим в себя компилятор Java (javac), стандартные библиотеки классов Java, примеры, документацию, различные утилиты и среду выполнения Java (JRE), необходимые для компиляции, отладки и выполнения приложений на языке программирования Java.

2. Скачайте среду разработки Eclipse. При скачивании выберите версию Eclipse, предназначенную для операционной системы, установленной на вашем компьютере, и имеющую такую же разрядность, что и JDK. При скачивании архива с расширением zip разархивируйте его в папку C:\eclipse. При скачивании исполняемого файла (с расширением exe) следуйте инструкциям мастера и установите Eclipse в ту же папку C:\eclipse.

6.2.2. Запуск

Чтобы запустить на выполнение среду разработки Eclipse, выполните следующие действия.

1. Перейдите в папку C:\eclipse в файловом обозревателе операционной системы, установленной на вашем компьютере. В операционной системе Windows файловым обозревателем является Проводник.

2. Запустите исполняемый файл eclipse.exe. В результате при первом запуске среды разработки Eclipse появится окно выбора рабочей области *Workspace Launcher* (рис. 6.1). При задании определенных настроек это окно может и не появляться при следующих запусках.

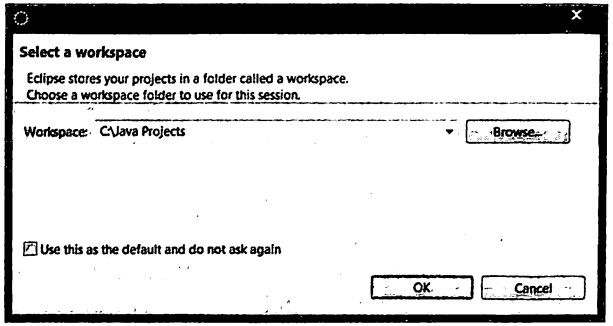


Рис. 6.1. Окно выбора рабочей области

ЗАЧЕМ ЭТО НУЖНО?

Рабочая область — это совокупность нескольких проектов, хранящихся в одном место. В среде разработки Eclipse можно создать несколько рабочих областей. Например, одну рабочую область можно выделить для учебных или рабочих проектов, другую — для своих собственных.

3. Нажмите кнопку *OK*. В результате появится приветственное сообщение.

4. Закройте приветственное сообщение. В результате появится основное окно среды разработки *Java — Eclipse*.

Чтобы перейти в другую рабочую область, выберите пункт меню *File | Switch Workspace*.

6.2.3. Основное окно и его элементы

Основными элементами окна среды разработки Eclipse являются следующие (рис. 6.2).

1. Панель *Package Explorer* со списком проектов и их структурой (пакетами и классами).

2. Редактор кода и текста с автодополнением и подсветкой синтаксиса.

3. Окно просмотра (консоли вывода результата работы приложения, отображаемых ошибок, описания Javadoc, комментариев TODO и т.п.).

В верхней части окна находится главное меню и панель инструментов.

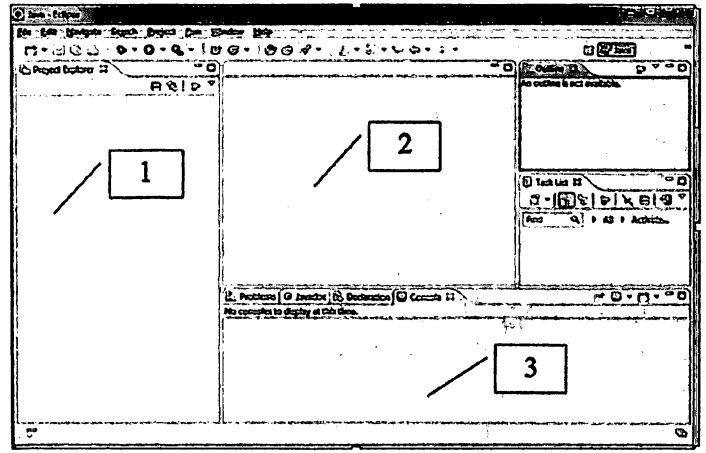


Рис. 6.2. Основное окно Eclipse

4. Выберите в списке *Installed JRE Types* пункт *Standard VM* и нажмите кнопку *Next*. В результате появится окно *JRE Definition*.

5. Нажмите кнопку *Directory...*, в открывшемся окне *Browse For Folder* укажите папку, в которой установлен JDK, и нажмите кнопку *OK* (например, C:\Program Files (x86)\Java\jdk1.6.0_45).

6. В результате в поле *JRE name:* будет указано имя JDK.

7. Нажмите кнопку *Finish*. В списке *Installed JRE* окна *Preferences* появится добавленный JDK.

8. Установите флажок у добавляемого JDK.

9. Нажмите кнопку *OK*.

В результате JDK установлена в качестве среды выполнения.

6.3.4. Подключение исходного кода стандартных классов

При написании проектов или при оптимизации кода по времени выполнения у программиста может возникнуть необходимость посмотреть исходный код классов, входящих в Java Standard Edition (Java SE). Для этого необходимо подключить их в среду разработки.

Чтобы подключить исходный код стандартных классов в среду разработки Eclipse, выполните следующие действия.

1. Выберите пункт меню *Window* | *Preferences*. В результате появится окно *Preferences*.

2. В списке слева выберите пункт меню *Java* | *Installed JREs*. В результате справа появится список установленных сред выполнения.

3. Выберите текущую среду выполнения (у которой установлен флажок) и нажмите кнопку *Edit...*. В результате появится окно *Edit JRE*.

4. В списке *JRE system libraries:* выберите библиотеку rt.jar.

5. Нажмите кнопку *Source attachment*. В результате появится окно *Source Attachment Configuration* (рис. 6.4).

6. Нажмите кнопку *External File...*. В результате появится окно выбора файла.

7. Выберите файл src.zip. Его можно найти в папке, в которой установлен JDK (например, C:\Program Files (x86)\Java\jdk1.6.0_45\src.zip).

8. Нажмите кнопку *OK* в окне *Source Attachment Configuration*.

9. Нажмите кнопку *Finish* в окне *Edit JRE*.

10. Нажмите кнопку *OK* в окне *Preferences*.

В результате исходные коды стандартных классов будут подключены к среде разработки Eclipse.

Далее будет приведена последовательность действий для просмотра исходного кода стандартных классов.

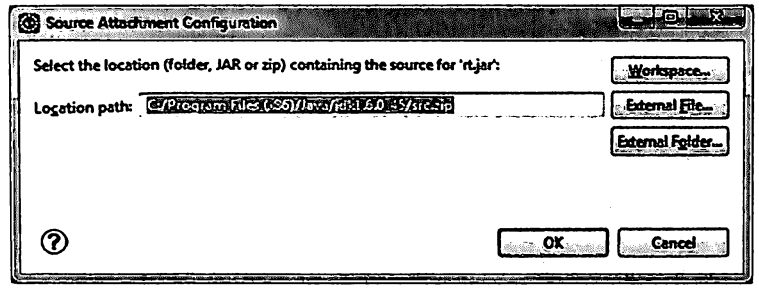


Рис. 6.4. Окно подключения исходных кодов стандартных классов Java Standard Edition

6.4. Наиболее часто выполняемые действия в среде разработки Eclipse

6.4.1. Создание проекта. Метод main

Выполним типовые действия в среде Eclipse на примере разработки учебного приложения. Приложение предназначено для сложения двух чисел и называется TestJava.

Чтобы создать проект с именем TestJava, выполните следующие действия.

1. Выберите пункт меню *File* | *New* | *Java Project*. В результате появится окно *New Java Project* (рис. 6.5).

2. В поле *Project name* введите название проекта TestJava и нажмите кнопку *Finish*.

В результате на панели *Package Explorer* появится созданный проект (рис. 6.6).

6.4.2. Создание класса

Чтобы создать в проекте TestJava класс с именем Runner и методом main, выполните следующие действия.

1. Щелкните правой кнопкой мыши по проекту TestJava на панели *Package Explorer* (справа в главном окне). В результате появится контекстное меню.

2. Выберите пункт контекстного меню *New* | *Class*. В результате появится окно *New Java Class* (рис. 6.7).

3. Введите в поле *Package:* название пакета ru.rsreu.testjava.

4. Введите в поле *Name:* имя класса Runner.

5. Установите флажок *public static void main(String[] args)*.

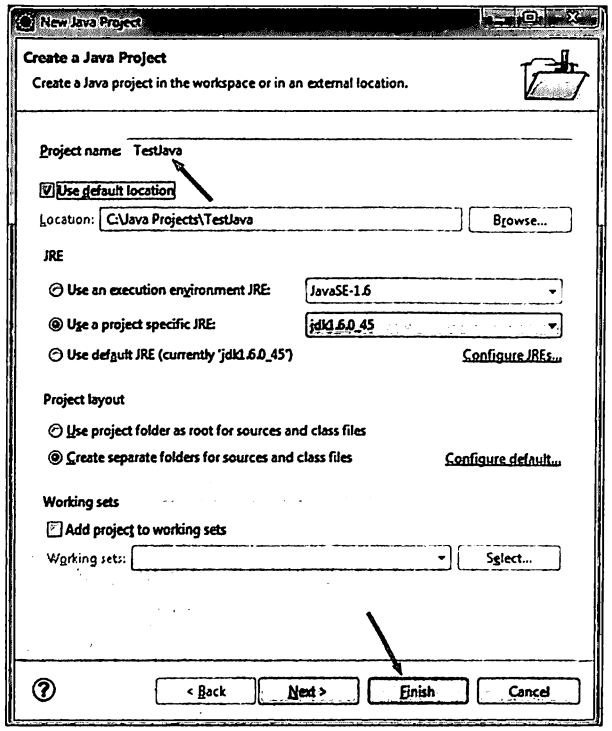


Рис. 6.5. Окно создания нового проекта

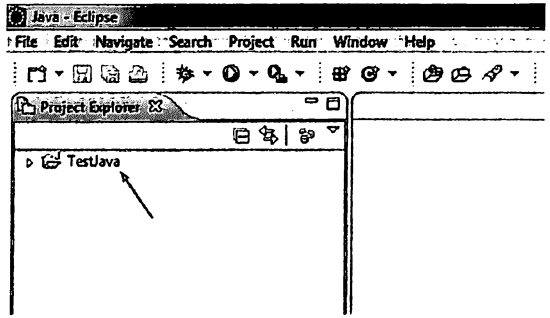


Рис. 6.6. Созданный проект

6. Нажмите кнопку *Finish*.

В результате на текущей вкладке редактирования кода основного окна появится класс Runner с методом main в пакете ru.rsreu.testjava (рис. 6.8).

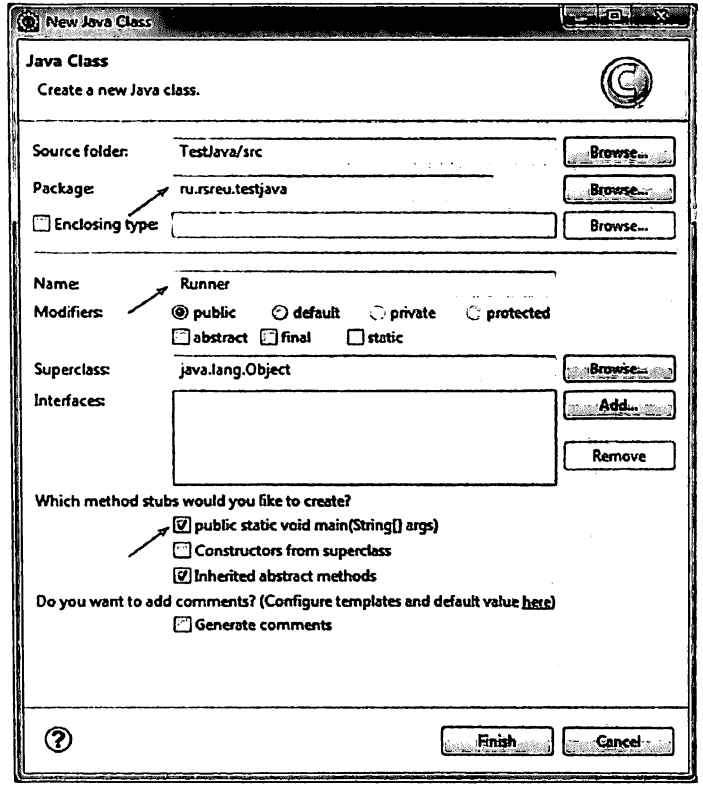


Рис. 6.7. Окно создания нового класса

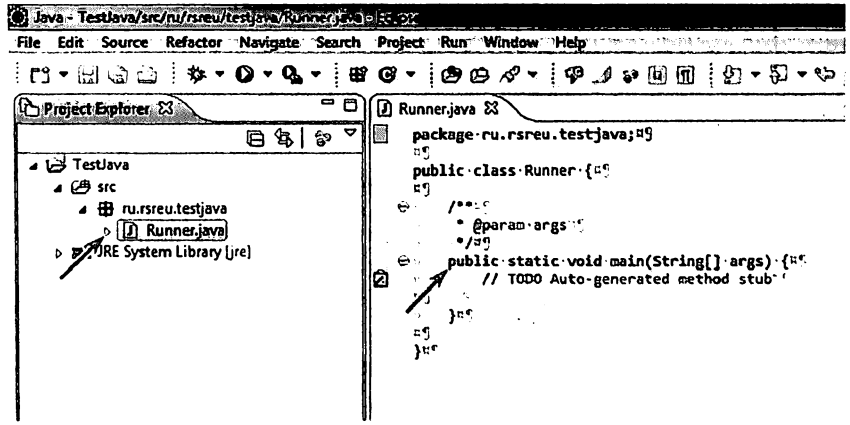


Рис. 6.8. Созданный класс Runner

ЗАЧЕМ ЭТО НУЖНО?

Приложение на объектно-ориентированном языке программирования Java состоит из одного или более класса. Один из классов должен включать метод main. Метод main называют точкой входа в приложение. С него начинается выполнение приложения.

Добавьте в класс Runner строчки так, чтобы этот класс выглядел следующим образом:

```
1   import java.util.Arrays;
2
3   public class Runner
4
5       public static final int ITEM_1 = 2;
6       public static final int ITEM_2 = 3;
7
8       /**
9       * @param args
10      */
11      public static void main(String[] args) {
12
13          Summarizer summarizer = new Summarizer(ITEM_1, ITEM_2);
14
15          int c = summarizer.sum();
16
17          System.out.println("c = " + c);
18
19          System.out.println(Arrays.toString(args));
20      }
21
22  }
```

В методе main класса Runner задаются исходные данные, вызывается метод решения задачи и выводятся результаты решения на консоль окна просмотра. Класс Runner не содержит код для сложения двух чисел.

Два числа будут складываться в методе класса Summarizer.

Создайте еще один класс с именем Summarizer в том же пакете ru.rsreu.testjava, выполнив все пункты создания класса Runner, за исключением п. 5, так как только один класс в проекте должен иметь метод main.

ЗАЧЕМ ЭТО НУЖНО?

Работа с кодом приложения не заканчивается после его окончательной разработки. К приложению добавляются новые функции, реализуемые в новом коде. Одним из требований при написании кода является простота его расширения. Это достигается удовлетворением множеству принципов. Один из них — «Один класс — одна задача». Этот принцип означает, что класс должен решать только одну задачу, а не несколько. Только в этом случае код легко расширять. Именно поэтому метод сложения двух чисел будет вынесен в отдельный класс.

Добавьте в созданный класс строчки так, чтобы этот класс выглядел следующим образом (рис. 6.9):

```
1   package ru.rsreu.testjava;
2
3   public class Summarizer {
4       private int arg1;
5       private int arg2;
6
7       public Summarizer(int arg1, int arg2) {
8           super();
9           this.arg1 = arg1;
10          this.arg2 = arg2;
11      }
12
13      public int sum() {
14          return this.arg1 + this.arg2;
15      }
16
17  }
```

Обратите внимание на то, что все имена классов, полей, переменных и других элементов в коде и проекте соответствуют их назначению в коде (кроме переменной *c* в методе main).

ЗАЧЕМ ЭТО НУЖНО?	Правильное именование элементов в проекте, т.е. именование, соответствующее назначению в коде, улучшает читаемость кода.

Обратите внимание на то, что элементы класса следуют в следующем порядке: поля (arg1 и arg2), конструкторы (конструкторы с параметрами arg1 и arg2), методы (метод sum).

ЗАЧЕМ ЭТО НУЖНО?	Такой порядок в классе (поля, конструкторы, методы) определен в соглашении по оформлению кода Java Code Conventions, разработанном создателями языка программирования Java.

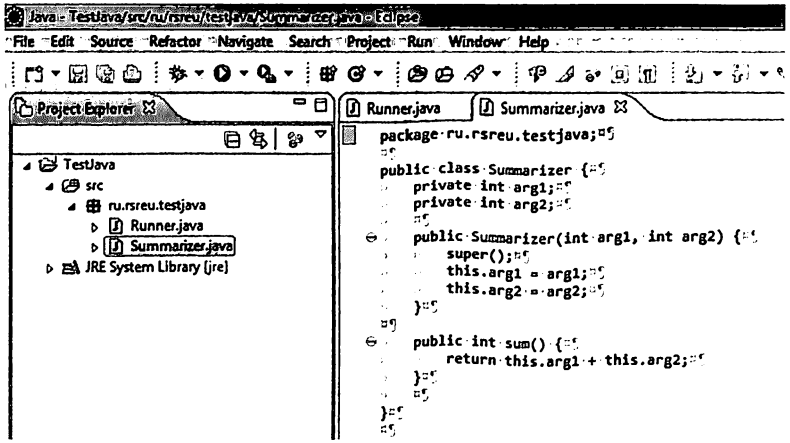


Рис. 6.9. Класс Summarizer

6.4.3. Форматирование кода

Чтобы отформатировать код класса Runner, выполните следующие действия.

1. Перейдите на вкладку с кодом класса Runner в редакторе кода.
2. Щелкните правой кнопкой мыши на коде класса Runner. В результате появится контекстное меню.
3. Выберите пункт контекстного меню *Source | Format*.

В результате код класса Runner будет отформатирован согласно правилам оформления кода, установленным в настройках среды разработки Eclipse.

Если вы выделите фрагмент кода и выполните перечисленные выше действия, то будет отформатирован только выделенный фрагмент.

ЗАЧЕМ ЭТО НУЖНО?	Форматирование кода необходимо для того, чтобы автоматически приводить код, написанный различными разработчиками, к единому оформлению, а не вручную расставлять пробелы, знаки табуляции, знаки конца стоки и т.д.

☑ Для форматирования кода используйте сочетание клавиш CTRL+SHIFT+F.

Запуск проекта на выполнение

Чтобы запустить текущий проект на выполнение, выберите пункт меню *Run | Run*.

В результате проект будет запущен на выполнение. В консоли окна просмотра будут выведены результаты работы приложения (рис. 6.10):

Вывод на экран
c = 5 []

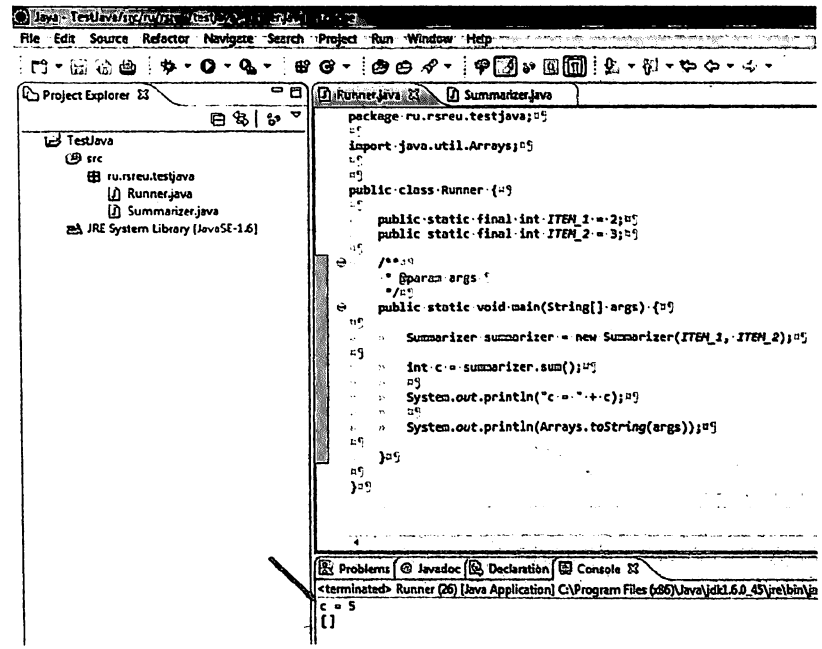


Рис. 6.10. Результат работы приложения

☑ Для запуска проекта на выполнение используйте сочетание клавиш CTRL+F11.

Запустим приложение с параметрами в командной строке.

Чтобы получить доступ к расширенным настройкам запуска, таким как выбор класса с методом main, задание параметров командной строки, выполните следующие действия.

1. На панели *Package Explorer* щелкните правой кнопкой мыши по проекту TestJava. В результате появится контекстное меню.
2. Выберите пункт контекстного меню *Run As* | *Run Configurations...*. В результате появится окно *Run Configurations*.
3. На вкладке *Main* в поле *Project:* введите имя проекта TestJava или нажмите кнопку *Browse...* и выберите проект TestJava.
4. В поле *Main class* введите имя класса Runner, содержащего метод main, или нажмите кнопку *Search...*, чтобы найти этот класс.
5. На вкладке *Arguments* в поле *Program arguments:* введите параметры командной строки через пробел: 1 4.
6. Нажмите кнопку *Run*.

В результате на вкладке Console в окне просмотра (под редактором кода) будет выведен результат выполнения проекта:

Вывод на экран
c = 5 [1, 4]

6.4.5. Просмотр исходного кода стандартных классов Java Standard Edition

В методе main класса Runner для вывода элементов массива аргументов командной строки args используется вызов метода toString класса Arrays.

Чтобы посмотреть исходный код класса Arrays, входящего в Java Standard Edition, выполните следующие действия.

1. В редакторе кода перейдите на вкладку с кодом класса Runner.
2. Зажмите клавишу *CTRL* и щелкните левой кнопкой мыши по названию классу Arrays в строке System.out.println(Arrays.toString(args));.

В результате в редакторе кода откроется новая вкладка с исходным кодом класса Arrays. Класс также содержит комментарии к методам.

6.4.6. Переименование элементов проекта

Переименование переменных, полей, методов, классов и других элементов во всем проекте выполняется похожим образом.

Чтобы переименовать класс Runner, выполните следующие действия.

1. В окне *Package Explorer* щелкните правой кнопкой мыши по классу Runner. В результате появится контекстное меню.
2. Выберите пункт контекстного меню *Refactor* | *Rename...*. В результате появится окно *Rename Compilation Unit* (рис. 6.11).
3. Введите в поле *New Name* новое имя класса ApplicationRunner и нажмите кнопку *Finish*.
4. В случае если появится окно *Rename Compilation Unit*, предупреждающее, что после переименования класса некоторые приложения могут не запускаться, то нажмите кнопку *Finish*.

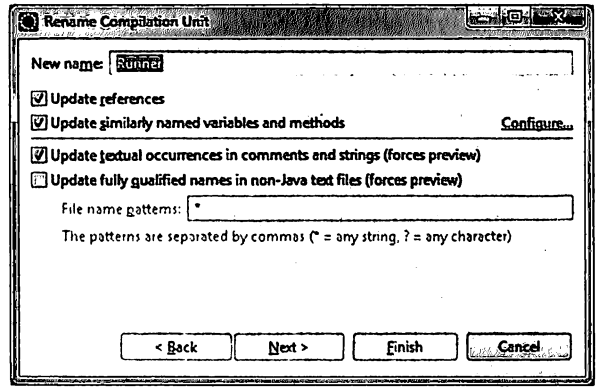


Рис. 6.11. Окно переименования класса

В результате класс Runner будет переименован в ApplicationRunner. Обратите внимание, что также был переименован файл, содержащий класс ApplicationRunner, а также вкладка в редакторе кода.

Чтобы переименовать переменную *c* в методе main класса Runner, выполните следующие действия.

1. Выделите переменную *c* в методе main и нажмите правой кнопкой мыши. В результате появится контекстное меню.
2. Выберите пункт контекстного меню *Refactor* | *Rename*. В результате имя переменной станет доступным для редактирования.
3. Введите новое имя переменной summ и нажмите клавишу *Enter*.

В результате переменная *c* будет переименована в summ во всех классах проекта.

ЗАЧЕМ ЭТО НУЖНО?

Переименование элементов приложения именно этим способом необходимо для того, чтобы изменить имена элементов одновременно во всем проекте.

☑ Для переименования элементов в коде установите курсор на его имени или выделите этот элемент и нажмите сочетание клавиш ALT+SHIFT+R.

6.4.7. Создание нового пакета. Назначение пакетов

ЗАЧЕМ ЭТО НУЖНО?

Обычно приложения состоят из сотен или тысяч классов. Классы, решающие одну задачу или однотипные задачи, образующие иерархию, обычно хранятся в одном месте. Это место называется пакетом. Пакеты могут иметь подпакеты без ограничения уровня вложенности. Пакеты позволяют структурировать исходный код приложения.

Принято следующее правило именования пакетов. Имя пакета начинается с названия веб-сайта разработчика (в нашем случае rsreu.ru) в обратном порядке (ru.rsreu). Далее следует название приложения или кодового имени проекта по разработке приложения (testjava). Далее следуют имена подпакетов.

Чтобы создать новый пакет с именем ru.rsreu.testjava, выполните следующие действия.

1. Выберите пакет ru.rsreu.testjava на панели *Package Explorer*.
2. Выберите пункт меню *File* | *New* | *Package*. В результате появится окно *New Java Package*.
3. В поле *Name* введите имя нового пакета ru.rsreu.testjava.classes.
4. Нажмите кнопку *Finish*.

В результате будет создан новый пакет с именем ru.rsreu.testjava.classes.

6.4.8. Перемещение класса из одного пакета в другой

Чтобы переместить класс Summarizer в пакет ru.rsreu.testjava.classes, выполните следующие действия.

1. На панели *Package Explorer* щелкните правой кнопкой мыши по классу Summarizer. В результате появится контекстное меню.
2. Выберите пункт контекстного меню *Refactor* | *Move*.... В результате появится окно *Move*.

3. В поле *Choose destination for 'Summarizer.java'* выберите пакет ru.rsreu.testjava.classes, в который будет перемещен класс Summarizer.
4. Нажмите кнопку *OK*.

В результате класс Summarizer будет перемещен в пакет ru.rsreu.testjava.classes.

☑ Для перемещения классов из одного пакета в другой используйте технику Drag'n'Drop.

6.4.9. Экспорт проекта в архив zip

Чтобы экспортировать проект TestJava в архив в формате zip, выполните следующие действия.

1. На панели *Package Explorer* щелкните правой кнопкой мыши по проекту TestJava. В результате появится контекстное меню.
2. Выберите пункт контекстного меню *Export*.... В результате появится окно мастера *Export*.
3. В списке *Select an export destination* выберите пункт *General* | *Archive File* и нажмите кнопку *Next*. В результате мастер перейдет на следующий шаг (рис. 6.12).

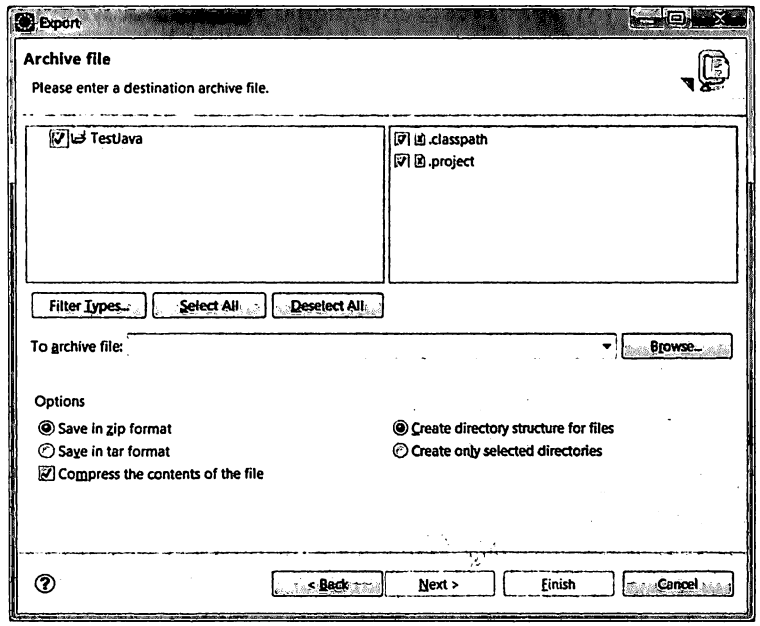


Рис. 6.12. Второе окно мастера экспорта проекта в архив в формате zip

4. В поле *To archive file:* введите путь для сохранения и имя файла testjava.zip.

5. В списке *Options* установите переключатели *Save in zip format* и *Create directory structure for files*.

6. Установите флажок *Compress the contents of the file*.

7. Нажмите кнопку *Finish*.

В результате проект TestJava будет экспортирован в архив testjava.zip.

ЗАЧЕМ ЭТО НУЖНО?

Архив с проектом в формате zip используется для хранения проекта вне среды разработки, для передачи их по сети или для переноса их на носителях памяти (флэш-карты, флэш-накопители) с последующим открытием их в среде разработки Eclipse.

6.4.10. Удаление проекта

Чтобы удалить проект TestJava, выполните следующие действия.

1. На панели *Package Explorer* щелкните правой кнопкой мыши по проекту TestJava. В результате появится контекстное меню.

2. Выберите пункт контекстного меню *Delete*. В результате появится окно *Delete Resources*.

3. Установите флажок *Delete project contents on disk (cannot be undone)*. В этом случае проект будет удален не только из панели *Package Explorer*, но и из рабочей области среды разработки, но не из экспортированных архивов в формате zip.

4. Нажмите кнопку *OK*.

В результате проект TestJava будет удален.

6.4.11. Импорт проекта из архива zip

Ранее экспортированный проект можно вновь добавить в среду разработки Eclipse.

Чтобы импортировать проект TestJava из архива zip в среду разработки Eclipse, выполните следующие действия.

1. Выберите пункт меню *File* | *Import...*. В результате появится окно мастера *Import*.

2. В поле *Select an import source* выберите пункт *General* | *Existing Projects into Workspace* и нажмите кнопку *Next*. В результате мастер перейдет на следующий шаг.

3. Установите переключатель *Select archive file:* и нажмите кнопку *Browse*. В результате появится окно выбора файла *Select archive containing the projects to import*.

4. Выберите путь и ранее экспортированный архив с проектом testjava.zip и нажмите кнопку *Открыть*. В результате окно выбора файла *Select archive containing the projects to import* закроется.

5. Нажмите кнопку *Finish* окна мастера *Import*.

В результате проект TestJava будет добавлен на панель *Package Explorer*.

7. ПРИМЕРЫ ПРОГРАММ ДЛЯ ПРОВЕДЕНИЯ ЛЕКЦИОННЫХ ЗАНЯТИЙ

7.1. Пояснения к примерам программ

7.1.1. Для студентов

Примеры программы из этого раздела используются для проведения лекционных занятий. По каждому фрагменту программы вам будут заданы вопросы, призванные выявить понимание синтаксиса программы, назначения конструкций языка программирования Java, выполнение объектно-ориентированных программ. Вопросы задаются всем студентам или индивидуально. Ответы на вопросы будут учтены при выставлении оценки на экзамене или зачете. Для подготовки к лекционному занятию необходимо прочитать разделы, указанные преподавателем.

7.1.2. Для самостоятельно изучающих язык программирования Java

Если вы изучаете язык программирования Java самостоятельно, то используйте примеры программ из этого раздела для проверки закрепления теоретического материала.

Цель этих примеров — научить читать программу и понимать назначение как программы целиком, так и отдельных ее строк.

Прежде чем разбирать примеры программ, прочитайте соответствующие разделы с теоретическим материалом.

По каждому примеру программы ответьте на три вопроса:

1. Что делает каждая строка программы?
2. Что будет выведено на консоль, если такой вывод есть в примере?
3. Что делает метод класса, если он есть в примере?

Примеры разбирайте до выполнения практических работ.

7.2. Примеры программ по различным темам

7.2.1. Основы языка программирования Java. Класс

```
1  public class PDFDoc {
2
3      private PDFPages pages;
4
5      protected PDFDoc() {
6          pages = new PDFPages();
7      }
8
9      public PDFDoc(PDFPages pages) {
10         this.pages = pages;
11     }
12
13     public int PDFDocConstructor() {
14         return this.pages.length();
15     }
16
17     void deletePages(int from, int to) {
18         this.pages.delete(from, to);
19     }
20
21     public void deletePages(int from) {
22         this.pages.delete(from, this.pages.length() - 1);
23     }
24
25     public void insertPages(int from, PDFPages newPages) {
26         this.pages.add(from, newPages);
27     }
28
29     public PDFPages extractPages(int from, int to) {
30         PDFPages extractedPages = pages.subrange(from, to);
31         this.deletePages(from, to);
32         return extractedPages;
33     }
34
35     public PDFPages getPages() {
36         return this.pages;
37     }
38 }
```

7.2.2. Основы языка программирования Java. Классы. Проектирование классов и методов

```
1  public class SumCalc {
2      public void execute() {
3          int a, b, c;
4          Scanner in = new Scanner(System.in);
5
6          System.out.print("Enter item a: ");
7          a = in.nextInt();
8
9          System.out.print("Enter item b: ");
10         b = in.nextInt();
11
12         c = a + b;
13
14         System.out.print("Sum of " + a + " and " + b+ " is " + c);
15
16         in.close();
17     }
18 }
```

7.2.3. Основы языка программирования Java. Программные блоки

```
1  int i = 1;
2  {
3      int i = 2;
4      System.out.println("1. i=" + i);
5
6      i = 3;
7      System.out.println("2. i=" + i);
8  }
9  System.out.println("3. i=" + i);
```

7.2.4. Элементы языка программирования Java. Переменные и типы. Объявление переменных. Базовые типы. Классы-оболочки базовых типов

```
1  double doubleNumber;
2  int integerNumber1, integerNumber2;
3  char aChar;
4  real realNumber;
5  Double anotherDouble;
6  float f1, f2;
7  final int intValue;
```


7.2.5. Элементы языка программирования Java. Переменные и типы. Инициализация переменных

```
1  double doubleValue = 1.5f;
2  float floatValue = 1.5d;
3
4  int i = doubleValue;
5
6  byte b = i;
7  int anotherInteger = b;
8
9  byte anotherByte = 2 * b;
10
11 Integer integerObject = i;
12 Integer doubleValueReceiver = doubleValue;
13
14 char symbol = "a";
15 char doubleSymbol = "aa";
16
17 final int integerValue;
18 integerValue = 5;
```

7.2.6. Элементы языка программирования Java. Переменные и типы. Преобразования числовых значений в различные типы. Методы parseXxx, toString и valueOf

```
1  String s = "101";
2  int i = 101;
3  Integer integerValue = i;
4
5  System.out.println(Integer.parseInt(s));
6  System.out.println(Integer.parseInt(s, 2));
7
8  System.out.println(Integer.valueOf(i));
9  System.out.println(Integer.valueOf(s));
10
11 System.out.println(Integer.toString(i));
12 System.out.println(Integer.toString(integerValue, 2));
13
14 System.out.println(String.toString(i));
15 System.out.println(String.valueOf(integerValue));
16
17 System.out.println(integerValue.doubleValue());
18 System.out.println(Float.parseFloat(s));
```

7.2.7. Элементы языка программирования Java. Строки. Создание. Операции

```
1  String splittingString = new String ("abc abcd abcd ab");
2  String[] splittedStrings = splittingString.split("b");
3  System.out.println(splittedStrings.length);
4  System.out.println(Arrays.toString(splittedStrings));
5
6  String trimmingString = "123456 ";
7  String trimmedString = trimmingString.trim();
8  trimmedString += " ";
9  System.out.println(trimmingString == trimmedString);
10 System.out.println(trimmingString.equals(trimmedString));
11
12 String substring1 = trimmingString.substring(2, 6);
13 System.out.println(substring1);
14 String substring2 = substring1.substring(3);
15 System.out.println(substring2);
16
17 String whatIsThis = "[a, c a, cd a, cd a]";
18 String searchingString = "c";
19 System.out.println(splittingString.indexOf(searchingString));
20 System.out.println(splittingString.indexOf(searchingString, 4));
```

7.2.8. Элементы языка программирования Java. Строки. String. StringBuilder. StringBuffer

```
1  String numbers = "Numbers:\n";
2
3  for (int i = 0; i < 10; i++) {
4      numbers += Integer.toString(i) + "\n";
5  }
6
7  String result = numbers;
```

```
1  StringBuilder numbers = new StringBuilder("Numbers:\n");
2
3  for (int i = 0; i < 10; i++) {
4      numbers.append(Integer.toString(i)).append("\n");
5  }
6
7  String result = numbers.toString();
```


Производительность и расход памяти для аналогичного фрагмента программы:

	String	StringBuilder	StringBuffer
Время выполнения, мс	23113	5	6
Используемая память, Кбайт	2674	1199	1199

Источник: http://java.globinch.com/java-quick-tips/performance-string-stringbuffer-stringbuilder-memory-runtime-analysis/

7.2.9. Элементы языка программирования Java. Строки. Неизменяемый тип String (Immutable)

```
1  String s = "Some string";
2  Object sPointer = s;
3  s = s + "Postfix";
4  System.out.println(sPointer == s);
5
6  StringBuilder sb = new StringBuilder("String too");
7  Object sbPointer = sb;
8  sb.append("Another prefix");
9  System.out.println(sbPointer == sb);
```

7.2.10. Элементы языка программирования Java. Массивы. Инициализация

```
1  int arrayLength = 2;
2
3  int[] array = new int[arrayLength];
4
5  array[0] = 0;
6  array[1] = 2;
```

7.2.11. Элементы языка программирования Java. Массивы. Индексы элементов

```
1  int[] numberArray = {2, 6, 3, 7, 5};
2
3  numberArray[3] = numberArray[0] + numberArray[2];
4  numberArray[4] = numberArray[4] + numberArray[3];
5
6  System.out.println(Arrays.toString(numberArray));
```

7.2.12. Элементы языка программирования Java. Массивы. Многомерные массивы

```
1  int[][] anArray = new int[3][];
2  anArray[0] = new int[2];
3  anArray[1] = anArray[0];
4  anArray[2] = new int[anArray.length];
5
6  int totalLength = 0;
7  for (int[] array : anArray) {
8      totalLength += array.length;
9  }
10
11 System.out.println(totalLength);
12 System.out.println(anArray[1][0]);
```

7.2.13. Основы языка программирования Java. Пакеты

Компанией «BusinessData» (http://businessdata.com) разработано приложение AnalyticAssistant, которое получает данные из различных источников, обрабатывает их и формирует из них отчеты. Распределите классы, составляющие приложение, по пакетам.

Классы:
ReportToPDFConverter
ExponentialForecaster
MarketReportMaker
DBDataGetter
NetworkDataParser
DataFromFileReader
ConfigSaver
TotalReportCreator
ReportPrinter
BusinessDataEstimater
ExcelDataExporter
ConfigRestorer
CompetitorDataAggregator
DataApplier
BusinessDataFormatter

7.2.14. Объектно-ориентированное программирование. Классы и их структура. Параметры передаются в методы только по значению

```
1  public class ParameterChanger {
2      public void run(int baseTypeParameter,
3          String constantObjectParameter, String[] objectParameter) {
4          baseTypeParameter = 100;
5          constantObjectParameter = "New String";
6          System.out.println("pointerKeeper == constantObjectParameter: "
7              + (pointerKeeper == constantObjectParameter));
8          objectParameter[0] = "New Array Element Value";
9      }
10 }
11
12 // In some method
13 String outputFormat = "%d; %s; %s\n";
14
15 int baseType = 1;
16 String constantObject = "Some String";
17 String[] object = { "Array element" };
18
19 ParameterChanger parameterChanger = new ParameterChanger();
20
21 System.out.printf(outputFormat, baseType, constantObject,
22     Arrays.toString(object));
23
24 parameterChanger.run(baseType, constantObject, object);
25
26 System.out.printf(outputFormat, baseType, constantObject,
27     Arrays.toString(object));
```

7.2.15. Объектно-ориентированное программирование. Классы и их структура. Спецификатор static

```
1  public class StaticModifierTester {
2      public static int aStaticField=3;
3
4      public static void modifyTheStaticField(int newValue) {
5          aStaticField = newValue;
6      }
7
8      public int getTheStaticField() {
9          return StaticModifierTester.aStaticField;
10     }
11 }
12
13 // in some method
14 StaticModifierTester theFirst = new StaticModifierTester();
15 StaticModifierTester theSecond = new StaticModifierTester();
16
17 String outputResult = StaticModifierTester.aStaticField + "\n";
18
19 StaticModifierTester.modifyTheStaticField(4);
20
21 theFirst.modifyTheStaticField(5);
22
23 outputResult += theFirst.getTheStaticField() + "\n";
24 outputResult += theSecond.getTheStaticField() + "\n";
25 outputResult += StaticModifierTester.aStaticField + "\n";
26
27 System.out.println(outputResult);
```

7.2.16. Объектно-ориентированное программирование. Классы и их структура. Спецификатор final

```
1  public final class FinalModifierApplier {
2      private final int anIntField;
3
4      public final int runAMethod(final int aParameter) {
5          this.anIntField = aParameter;
6          final int finalResult = aParameter;
7          return finalResult;
8      }
9  }
```


7.2.17. Объектно-ориентированное программирование. Классы и их структура. Спецификаторы доступа

```
1  package com.prutzkow.modifiers.classa;
2  public class ClassA {
3      private int privateField;
4      protected int protectedField;
5      int friendlyField;
6      public int publicField;
7  }
8
9  package com.prutzkow.modifiers.classa;
10 public class ClassB {
11     public void getAccessToField() {
12         ClassA objectA = new ClassA();
13         objectA.friendlyField = 1;
14         objectA.protectedField = 2;
15 }}
16
17 package com.prutzkow.modifiers.classa;
18 public class ClassC extends ClassA {
19     public void getAccessToField() {
20         super.protectedField = 1;
21         super.friendlyField = 2;
22 }}
23
24 package com.prutzkow.modifiers.classd;
25     public class ClassD extends ClassA {
26         public void getAccessToField() {
27           super.protectedField = 1;
28           super.friendlyField = 2;
29 }}
30
31 package com.prutzkow.modifiers.classe;
32 public class ClassE {
33     public void getAccessToField() {
34         ClassA objectA = new ClassA();
35         objectA.friendlyField = 1;
36         objectA.protectedField = 2;
37 }}
```

7.2.18. Объектно-ориентированное программирование. Принципы

```
1  public class ClassA {
2      private int aField = 1;
3
4      public ClassA() { super(); }
5
6      public int getTheField() { return this.aField; }
7      public void setTheField(int aField) { this.aField = aField; }
8      public int modifyTheField() { return this.getTheField() * 2; }
9
10     public String toString() {
11         return "Class \"" + this.getClass().getName()
12           + "\" [TheField:" + this.modifyTheField() + "]";
13     }
14 }
15
16 public class ClassB extends ClassA {
17     public ClassB() { super(); this.setTheField(10); }
18
19     public int modifyTheField() { return super.modifyTheField()* 2; }
20 }
21
22 public class ClassC extends ClassB {
23     public ClassC() { this.setTheField(100); }
24
25     public int modifyTheField() { return this.getTheField() * 2; }
26 }
27
28 public static void main(String[] args) {
29     int arrayLength = 3;
30     ClassA[] classAArray = new ClassA[arrayLength];
31
32     classAArray[0] = new ClassA();
33     classAArray[1] = new ClassB();
34     classAArray[2] = new ClassC();
35
36     System.out.println(Arrays.toString(classAArray));
37 }
```


7.2.19. Элементы языка программирования Java. Компараторы

```
1  public class ComparableObject implements
2      Comparable<ComparableObject> {
3
4      private int aField;
5
6      public ComparableObject(int fieldValue) {
7          this.aField = fieldValue;
8      }
9
10     @Override
11     public int compareTo(ComparableObject o) {
12         return - (getTheField() - o.getTheField());
13     }
14
15     public int getTheField() {
16         return this.aField;
17     }
18
19     public void setTheField(int fieldValue) {
20         this.aField = fieldValue;
21     }
22 }
```

```
1  public class EveBeforeOddComparator implements
2      Comparator<ComparableObject> {
3      @Override
4      public int compare(ComparableObject o1, ComparableObject o2) {
5          int compareResult = o1.getTheField() % 2 - o2.getTheField() % 2;
6          if (compareResult == 0) {
7              compareResult = o1.getTheField() - o2.getTheField();
8          }
9          return compareResult;
10     }
11 }
```

7.2.20. Элементы языка программирования Java. Компараторы. Использование

```
1  ComparableObject[] array = { new ComparableObject(5),
2      new ComparableObject(1), new ComparableObject(4),
3      new ComparableObject(6) };
4
5  String resultText = "Origin: " + Arrays.toString(array) + "\n";
6
7  Arrays.sort(array);
8  resultText += "Natural Sorted: " + Arrays.toString(array) + "\n";
9
10 Arrays.sort(array, new EveBeforeOddComparator());
11 resultText += "Via Comparator Sorted: " + Arrays.toString(array) + "\n";
12
13 System.out.println(resultText);
```

Вывод на экран

```
Origin: [5 , 1 , 4 , 6 ]
Natural Sorted: [6 , 5 , 4 , 1 ]
Via Comparator Sorted: [4 , 6 , 1 , 5 ]
```

7.2.21. Объектно-ориентированное программирование. Перечисления Enum. Типовое использование перечислений

```
1  public enum Season {
2      WINTER, SPRING, SUMMER, AUTUMN;
3  }
```

```
1  Season season = Season.SUMMER;
2
3  switch (season) {
4  case WINTER:
5      System.out.println("Winter");
6      break;
7  case SUMMER:
8      System.out.println("Summer");
9      break;
10 }
```


7.2.22. Объектно-ориентированное программирование. Перечисления Enum. Задача

Стоимость железнодорожного билета рассчитывается по следующей формуле:

Стоимость_билета = Стоимость_1_км · Расстояние.

Стоимость 1 км пути рассчитывается в зависимости от класса вагона.

Для эконом-класса:

Стоимость_1_км = Базовая_стоимость.

Для бизнес-класса:

Стоимость_1_км = Базовая_стоимость · Коэффициент_бизнес_класса.

Для люкс-класса 10% пути рассчитываются по базовой стоимости, а остальные 90% домножаются на коэффициент:

Стоимость_1_км = Базовая_стоимость · 10% +
+ Базовая_стоимость · 90% · Коэффициент_люкс_класса.

Как описать расчет стоимости билета с помощью ООП?

7.2.23. Объектно-ориентированное программирование. Перечисления Enum. Перечисления как суперкласс

```
public enum WagonClass {
    ECONOM {
        public double getKmCost() {
          return WagonClass.getBaseCost();
        }
    },
    BUSINESS {
        public double getKmCost() {
          return WagonClass.getBaseCost() *
            this.BUSINESS_CLASS_MULTIPLICATOR;
        }
    },
    LUX {
        public double getKmCost() {
          return WagonClass.getBaseCost() *
            this.LUX_CLASS_DISCOUNT
            + WagonClass.getBaseCost() *
            (1 — this.LUX_CLASS_DISCOUNT)
            * this.LUX_CLASS_MULTIPLICATOR;
        }
    };

    public final double BUSINESS_CLASS_MULTIPLICATOR = 1.2;
    public final double LUX_CLASS_MULTIPLICATOR = 2;
    public final double LUX_CLASS_DISCOUNT = 0.1;

    private static double baseCost = 0;

    public static double getBaseCost() {
        return WagonClass.baseCost;
    }

    public static void setBaseCost(double baseCost) {
        WagonClass.baseCost = baseCost;
    }

    public abstract double getKmCost();
}
```

```
public class Ticket {
    private Wagon wagon;
    private int placeNumber;

    public double getCost() {
        double distance = ...
        double ticketCost = distance *
          this.wagon.getWagonClass().getKmCost();
        return ticketCost;
    }

...
}
```

```
public class Wagon {
    private WagonClass wagonClass;

    public WagonClass getWagonClass() {
        return wagonClass;
    }

...
}
```

8. ДОПОЛНИТЕЛЬНЫЕ МАТЕРИАЛЫ

8.1. Порядок проведения лекционных и практических занятий

1. Порядок проведения лекционных занятий.

1.1. Преподаватель предварительно задает разделы для чтения из этого учебника, необходимые для выполнения практических работ.

1.2. Студенты читают заданные разделы в указанных учебных пособиях и готовятся теоретически по темам этих разделов.

1.3. На лекционном занятии преподаватель демонстрирует студентам примеры программ, связанные с заданными для чтения разделами, и задает по ним вопросы, поясняя с помощью них теоретический материал разделов.

2. Порядок проведения практических занятий.

2.1. Преподаватель выдает варианты практических работ и назначает дату их сдачи.

2.2. Студент выполняет свой вариант практической работы до срока, указанного в п. 2.1.

2.3. Практические работы выполняются последовательно без пропусков.

2.4. Студент высылает на адрес электронной почты, указанный преподавателем, два файла:

- файл с исходным кодом проекта;
- файл с отчетом, оформленный по правилам, установленным преподавателем.

2.5. Элементы практических работ именовать следующим образом.

Условные обозначения:

xx — порядковый номер практической работы (обязательно две цифры; если необходимо, добавить ведущий ноль);

yy — номер варианта практической работы, выполняемой студентом (две цифры);

zzzz — полная фамилия студента, записанная транслитерацией (см. раздел 8.2).

Использовать следующие шаблоны для имен:

- проекта: xx-yy-zzzz;
- пакетов: ru.rsreu.zzzzxxyy;

- файла проекта варианта практической работы: xx-yy-zzzz.zip;
- файла с отчетом: xx-yy-zzzz.doc;
- темы письма по электронной почте: РГРТУ xx-yy-zzzz.

2.6. Преподаватель не подтверждает получение писем. В случае отсутствия упоминания о полученной практической работе на ближайшем практическом занятии необходимо сообщить об этом преподавателю.

2.7. Преподаватель проверяет варианты работ, выполненных студентами, и фиксирует имеющиеся по ним замечания.

2.8. На ближайшем практическом занятии преподаватель зачитывает, поясняет и разбирает замечания по практическим работам, выполненным студентами, и выставляет оценку «зачтено»/«не зачтено».

2.9. В случае получения оценки «не зачтено» студент обязан устранить замечания, сделанные преподавателем, и выслать ему на адрес электронной почты, указанный в п. 2.4, исправленный вариант практической работы до срока сдачи следующей практической работы.

2.10. После выдачи 3–4 практических работ по их темам проводится контрольная работа.

2.11. Преподаватель консультирует студентов по всем вопросам только в часы своих занятий в вузе, но не по электронной почте.

2.12. Каждый студент выполняет практические работы самостоятельно.

2.13. В случае если преподаватель обнаружит, что практическая работа сдавалась кем-то еще, то все оценки «зачтено», полученные за практические работы, аннулируются и выдается новый вариант работы.

3. Порядок выставления оценок на экзамене или зачете.

3.1. Итоговая оценка по дисциплине выставляется после получения оценки «зачтено» за все практические работы и получения не менее чем удовлетворительных оценок за все контрольные работы.

3.2. Итоговая оценка по дисциплине выставляется на основе оценок за контрольные работы и результатов и сроков сдачи практических работы, работы на лекциях.

8.2. Правила транслитерации букв русского алфавита

8.2.1. Порядок транслитерации

1. Замените буквы своей фамилии буквами латинского алфавита, используя соответствие из следующего раздела.

В случае наличия однофамильца в вашей группе и отсутствия номера варианта в названии файла добавьте в конец фамилии номер варианта по студенческому журналу, например Ivanov7.

8.2.2. Соответствие букв русского алфавита буквам латинского алфавита

Буква русского алфавита	Буква латинского алфавита
А	A
Б	B
В	V
Г	G
Д	D
Е	E
Ё	E
Ж	ZH
З	Z
И	I
Й	J
К	K
Л	L
М	M
Н	N
О	O
П	P
Р	R
С	S
Т	T
У	U
Ф	F
Х	KH
Ц	TS
Ч	CH
Ш	SH
Щ	SCH
Ь	Нет обозначения
Ы	Y

Буква русского алфавита	Буква латинского алфавита
Ъ	Нет обозначения
Э	E
Ю	YU
Я	YA

8.3. Методические рекомендации по проведению курса

Лекционные и практические занятия проводились следующим образом. Студентам задавалось чтение определенных разделов этого учебника, необходимых для выполнения практической работы. На следующем лекционном занятии представлялись фрагменты программ, связанные с темами этих разделов, и задавались вопросы по этим фрагментам. В конце лекционного занятия выдавалось задание практической работы. Студенты присылали по электронной почте выполненные варианты практической работы. Работы проверялись, и ошибки в них фиксировались. На следующем практическом занятии ошибки разбирались как в виде объяснения, почему такое написание программы неправильно и какие проблемы это может вызвать при обслуживании программы, так и в виде вопросов: что неправильного в этом фрагменте программы. Описанный порядок проведения занятий приведен в разделе 8.1.

После 2–3 практических работ проводилась контрольная работа. Задание на контрольную работу разрабатывалось на основе заданий прошедших практических работ. Студентам подчеркивалось, что практические работы — это лишь способ подготовки к контрольной работе и именно по результатам контрольных работ будет выставлена окончательная оценка за курс, а не по факту сдачи практических работ.

На зачете или экзамене студентам выдавалось практическое экзаменационное задание, а после его выполнения проводилось собеседование. Формулировка экзаменационного задания основана на практических заданиях курса.

В следующем семестре после чтения данного курса автор давал задания по разработке и программированию следующих приложений:

1) игровых программ с использованием шаблона MVC (Model-View-Controller) и многопоточности;

2) веб-приложений с базой данных с использованием сервлетов, технологий JDBC (Java Data Base Connectivity), JSP (Java Server Pages) и шаблонов Frontcontroller и DAO (Data Access Object).

8.4. Временны́е требования к проведению курса

Параметр	Значение
Продолжительность	Не более 1 семестра
Количество необходимых часов для проведения лекционных занятий	Не менее 16 часов
Количество необходимых часов для проведения практических работ	Не менее 16 часов
Количество необходимых часов для проведения одной практической работы	Не менее 2 часов

8.5. Таблица покрытия понятий практическими работами

Изучаемые понятия	Практические работы						
	1	2	3	4	5	6	7
Принципы работы в среде разработки Eclipse	++			+			
Управляющие конструкции: условный оператор, операторы циклов		+	+				
Строки. Класс String		++					
Ввод данных с консоли; класс Scanner		+				+	
Одномерные массивы			++				
Методы с неизвестным количеством параметров			+				
Файлы ресурсов			+	+	+	+	+
Класс StringBuilder			+	+	+	+	+

Изучаемые понятия	Практические работы						
	1	2	3	4	5	6	7
Двумерные массивы с переменной длиной строк				++			
Наследование, полиморфизм и инкапсуляция				+	+		
Переопределение и перегрузка методов				+			
Проектирование классов и методов		+	+	+	+	+	
Ключевые слова this и super				+			
Подготовка программной документации				+			
Проектирование иерархии классов					++		
Отношения между классами: наследование и включение					+		
Методы класса Arrays: sort и binarySearch					+		
Определение порядка сортировки					+		+
Метод toString класса Object		+	+			+	
Методы equals и hashCode класса Object						+	
Исключения и способы их обработки						++	
Символьные и байтовые потоки ввода и вывода						++	
Работа с файловой системой						+	
Коллекции							++
Форматированный вывод						+	+
Перечисления						+	+

Обозначения:
++ — практическая работа посвящена изучаемому понятию;
+ — практическая работа предполагает использование изучаемого понятия.

8.6. Используемая терминология и ее перевод на английский язык

По-английски	По-русски
Abstract	Абстрактный
Charset	Кодировка символов
Constructor	Конструктор
Encapsulation	Инкапсуляция
Exception	Исключительная ситуация, исключение
Field	Поле
Getter	Метод, возвращающий значение поля
Inheritance	Наследование
Integrated Development Environment (IDE)	Среда разработки
Interface	Интерфейс
Java Application	Приложение
Java Runtime Environment (JRE)	Среда выполнения Java
Marker interface	Интерфейс-маркеры
Method	Метод
Multithreading	Многопоточность
Pattern	Шаблон
Package	Пакет
Project	Проект
Polymorphism	Полиморфизм
Refactoring	Рефакторинг
Setter	Метод, присваивающий полю новое значение
Static	Статический
Stream	Поток ввода или вывода
Subclass	Подкласс
Superclass	Суперкласс
Thread	Поток выполнения (в многопоточности)
Workspace	Рабочая область

Библиографический список

1. *Бишоп Д.* Эффективная работа: Java 2 [Текст] / Д. Бишоп. — СПб.: Питер; К.: Издательская группа BHV, 2002. — 592 с.
2. *Блинов И.Н.* Java. Методы программирования [Текст]: учеб.-метод. пособие / И.Н. Блинов, В.С. Романчик. — Минск: Четыре четверти, 2013. — 896 с.
3. *Васильев А.Н.* Java. Объектно-ориентированное программирование [Текст]: учеб. пособие / А.Н. Васильев. — СПб.: Питер, 2013. — 400 с.
4. *Власенко О.Ф.* Основы программирования на Java. Основные управляющие конструкции [Текст]: метод. указ. / О.Ф. Власенко. — Ульяновск: УлГТУ, 2015. — 80 с.
5. *Гаврилов А.В.* Программирование на Java. Конспект лекций [Текст] / А.В. Гаврилов, С.В. Клименков, Е.А. Цопа. — СПб: СПбГУ ИТМО, 2010. — 130 с.
6. *Гамма Э.* Приемы объектно-ориентированного проектирования. Паттерны проектирования [Текст] / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. — СПб.: Питер, 2015. — 368 с.
7. *Герасимова Т.В.* Лабораторный практикум по программированию на языке Java [Текст] / Т.В. Герасимова. — СПБ.: Изд-во СПбГЭТУ «ЛЭТИ», 2012. — 52 с.
8. *Дубаков А.А.* Введение в объектно-ориентированное программирование на Java [Текст]: учеб. пособие / А.А. Дубаков. — СПб: Ун-т ИТМО, 2016. — 250 с.
9. *Ерхов Р.В.* Разработка приложений в среде Eclipse [Текст]: метод. указ. / Р.В. Ерхов, А.В. Пруцков. — Рязань: Рязан. гос. радиотехн. ун-т, 2018. — 28 с.
10. Информатика и программирование. Алгоритмизация и программирование: учебник для студ. учреждений высш. проф. образования [Текст] / Н.И. Парфилова, А.В. Пруцков, А.Н. Пылькин, Б.Г. Трусов; под ред. Б.Г. Трусова. — М.: Академия, 2012. — 336 с.
11. Информатика и программирование. Основы информатики: учебник для студ. учреждений высш. проф. образования [Текст] / Н.И. Парфилова, А.В. Пруцков, А.Н. Пылькин, Б.Г. Трусов; под ред. Б.Г. Трусова. — М.: Академия, 2012. — 256 с.
12. *Макконнелл С.* Совершенный код. Мастер-класс [Текст] / С. Макконнелл. — М.: Русская редакция; СПб.: Питер, 2005. — 896 с.
13. *Мартин Р.* Чистый код: создание, анализ и рефакторинг. Библиотека программиста [Текст] / Р. Мартин. — СПб.: Питер, 2012. — 464 с.

14. *Машнин Т.С.* Современные Java-технологии на практике [Текст] / Т.С. Машнин. — СПб.: БХВ-Петербург, 2010. — 560 с.
15. *Монахов В.В.* Язык программирования Java и среда NetBeans [Текст] / В.В. Монахов. — 3-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2011. — 704 с.
16. *Мухортов В.В.* Объектно-ориентированное программирование, анализ и дизайн [Текст]: метод. пособие / В.В. Мухортов, В.Ю. Рылов. — Новосибирск: ИМ СО РАН, 2002. — 108 с.
17. *Нимейер П.* Программирование на Java [Текст] / П. Нимейер, Д. Леук. — М.: Эксмо, 2014. — 1216 с.
18. *Павловская Т.А.* C/C++. Программирование на языке высокого уровня [Текст] / Т.А. Павловская. — СПб.: Питер, 2005. — 461 с.
19. *Парамонов И.В.* Язык программирования Java и Java-технологии [Текст]: учеб. пособие / И.В. Парамонов. — Ярославль: Яросл. гос. ун-т., 2006. — 92 с.
20. *Прохоренок Н.А.* Основы Java [Текст] / Н.А. Прохоренок. — СПб.: БХВ-Петербург, 2017. — 704 с.
21. *Пруцков А.В.* Особенности преподавания промышленной разработки программных продуктов в технических вузах [Текст] / А.В. Пруцков // Современные технологии в науке и образовании — СТНО-2017: материалы 2-й Междунар. науч.-техн. и науч.-метод. конф. / Рязан. гос. радиотехн. ун-т. — Рязань, 2017. — С. 40–41.
22. *Пруцков А.В.* Математическая логика и теория алгоритмов [Текст]: учебник / А.В. Пруцков, Л.Л. Волкова. — М.: КУРС: ИНФРА-М, 2016. — 156 с.
23. *Пруцков А.В.* Язык программирования Java. Введение в курс: объектно-ориентированное программирование [Текст]: учеб. пособие / А.В. Пруцков. — Рязань: Рязан. гос. радиотехн. ун-т, 2016. — 56 с.
24. *Пруцков А.В.* Язык программирования Java. Введение в курс: операторы и типы данных [Текст]: учеб. пособие / А.В. Пруцков. — Рязань: Рязан. гос. радиотехн. ун-т, 2016. — 72 с.
25. *Пруцков А.В.* Проблемно-ориентированное объектное программирование [Текст] / А.В. Пруцков, Д.М. Цыбулько // Вестник Рязанского государственного радиотехнического университета. — 2013. — № 3 (45). — С. 57–62.
26. *Стелтинг С.* Java. Применение шаблонов Java [Текст] / С. Стелтинг, О. Маасен. — М.: Вильямс, 2002. — 576 с.
27. *Степанцов В.А.* Практическая работа в Eclipse 3.2 [Текст]: учеб.-метод. пособие для вузов / В.А. Степанцов. — Воронеж: Воронеж. гос. ун-т, 2007. — 57 с.

28. *Сухов С.А.* Основы программирования на Java [Текст]: учеб. пособие / С.А. Сухов. — Ульяновск: УлГТУ, 2006. — 88 с.
29. *Сьерра К.* Изучаем Java [Текст]: пер. с англ. / К. Сьерра, Б. Бэйтс. — М.: Эксмо, 2013. — 720 с.
30. *Титенко Е.А.* Программирование Web-сервисов на языке Java [Текст]: метод. указ. по выполнению лабораторной работы / Е.А. Титенко, М.В. Бородин. — Курск: Юго-Зап. гос. ун-т, 2013. — 21 с.
31. *Фримен Эр.* Паттерны проектирования [Текст] / Эр. Фримен, Эл. Фримен, К. Сьерра, Б. Бейтс. — СПб.: Питер, 2011. — 656 с.
32. *Хабибуллин И.Ш.* Самоучитель Java [Текст] / И.Ш. Хабибуллин. — 3-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2008. — 758 с.
33. *Хорстманн К.С.* Java 2. Библиотека профессионала. В 2 т. Т. 1. Основы [Текст]: пер. с англ. / К.С. Хорстманн, Г. Корнелл. — 10-е изд. — М.: Вильямс, 2016. — 864 с.
34. *Хорстманн К.С.* Java 2. Библиотека профессионала. В 2 т. Т. 2. Расширенные средства программирования [Текст]: пер. с англ. / К.С. Хорстманн, Г. Корнелл. — 10-е изд. — М.: Вильямс, 2016. — 1008 с.
35. *Шилдт Г.* Java. Полное руководство [Текст]: пер. с англ. / Г. Шилдт. — 8-е изд. — М.: Вильямс, 2013. — 1104 с.
36. *Эванс Б.* Java. Новое поколение разработки [Текст] / Б. Эванс, М. Вербург. — СПб.: Питер, 2014. — 560 с.
37. *Эккель Б.* Философия Java. Библиотека программиста [Текст] / Б. Эккель. — 4-е изд. — СПб.: Питер, 2009. — 640 с.
38. *Araujo R.F.* Getting Started with Eclipse Juno. Packt Publishing, 2013.
39. *Déléchamp F., Laugié H.* Java y Eclipse: Desarrolle una aplicación con Java y Eclipse. Ediciones ENI, 2016.
40. *Liguori R., Liguori P.* Java 7 Pocket Guide, 2nd ed. O'Reilly, 2013.
41. Sun Microsystems, Java Code Conventions. Palo Alto, California: Sun Microsystems, June 2, 1997. Accessed online at http://www.oracle.com/technetwork/java/codeconventions-150003.pdf.
42. Интернет-ресурс Пруцкова Александра Викторовича http://prutzkow.com [Электронный ресурс] // URL: http://prutzkow.com. Дата просмотра 14.05.2015.
43. Интернет-ресурс http://berkut.homelinux.com [Электронный ресурс] // URL: http://berkut.homelinux.com. Дата просмотра 14.05.2015.
44. Интернет-ресурс http://www.intuit.ru [Электронный ресурс] // URL: http://www.intuit.ru. Дата просмотра 14.05.2015.
45. Интернет-ресурс http://kostin.ws [Электронный ресурс] // URL: http://kostin.ws. Дата просмотра 14.05.2015.

46. Интернет-ресурс http://docs.oracle.com [Электронный ресурс] // URL: http://docs.oracle.com. Дата просмотра 14.05.2015.
47. Интернет-ресурс http://www.realcoding.net [Электронный ресурс] // URL: http://www.realcoding.net. Дата просмотра 14.05.2015.

ОГЛАВЛЕНИЕ

Учебное издание

Пруцков Александр Викторович

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ JAVA.
ВВЕДЕНИЕ В КУРС С ПРИМЕРАМИ И ПРАКТИЧЕСКИМИ ЗАДАНИЯМИ

Учебник

Оригинал-макет подготовлен в Издательстве «КУРС»

Подписано в печать 10.01.2018.
Формат 60×90/16. Бумага офсетная. Гарнитура Newton.
Печать цифровая. Усл. печ. л. 13,0.
Тираж 500 экз. Заказ № 1095

ТК 683040-961725-100118

ООО Издательство «КУРС»
127273, Москва, ул. Олонецкая, д. 17А, офис 104.
Тел.: (495) 203-57-83.
E-mail: kursizdat@gmail.com http://www.kursizdat.ru

Для заметок