O`ZBEKISTON RESPUBLIKASI OLIY VA O`RTA MAXSUS TA'LIM VAZIRLIGI

BUXORO DAVLAT UNIVERITETI

Qo'lyozma huquqida
UDK – 681.3.06

Xayriyev Umedjon Narmon o`g`li

"Aspektga mo`ljallangan dasturlashni qo`llab-quvvatlovchi java dasturlash tili kengaytmasi"

5A130202 – Amaliy matematika va axborot texnologiyalari

Magistr

akademik darajasini olish uchun yozilgan

DISSERTATSIYA

Ilmiy rahbar: **f-m.f.n.dots. O.I. Jalolov**

Buxoro – 2019

MUNDARIJA

KIRISH

Mavzuning dolzarbligi. Globallashuv asrida biron bir faoliyat sohasini axborot-kommunikatsiya texnologiyalarisiz tasavvur qilib bo‘lmaydi. Mamlakatimizda birinchi Prezidentimiz Islom Karimov tashabbusi bilan 2002-2010 yillarda kompyuterlashtirish va axborot-kommunikatsiya texnologiyalarini rivojlantirish dasturi ishlab chiqilib, bosqichma-bosqich amalga tatbiq etilmoqda. O‘zbekistonda xalqaro huquq me’yorlarini hisobga olgan holda, AKT sohasidagi milliy qonunchilik muntazam takomillashtirilmoqda. Ushbu qonunchilik bugun mualliflik va boshqa turdosh huquqlar, elektron imzo, tijorat, to‘lovlar, hujjat aylanishi sohasidagi munosabatlarni tartibga solmoqda. Axborot xavfsizligini ta’minlash iqtisodiy, ijtimoiy va madaniy rivojlanishning milliy ustuvorliklarini hurmat qilish tamoyillari asosida ochiq axborot jamiyatini tashkil etishda muhim masala hisoblanadi. Bu borada biribchi prezidentimiz I.A. Karimov ta’kidlaganlaridek: “Bugungi kunda milliy axborot tizimini shakllantirish jarayonida internet va boshqa global axborot tizimlaridan foydalanish, ayniqsa, muhim ahamiyatga ega” [3].

O‘zbekiston Respublikasi Prezidentining “O‘zbekiston Respublikasini yanada rivojlantirish bo‘yicha Harakatlar strategiyasi to‘g‘risida”gi Farmoni, hukumat qarorlarida belgilangan va davlatimiz rahbari ta’kidlagan “…ilmiy va ijodiy izlanishlarni har tomonlama qo‘llab-quvvatlash, ular uchun zarur shart-sharoitlar yaratish vazifa”si [1][2][4] har bir fan sohasi kabi, AKT sohasida ham yangi izlanishlar olib borishga undaydi. Shunday ekan, mustaqillik davrida shakllangan AKTda dasturlash sohasi tadqiqi yangi bosqichda rivojlantirilishi lozim.

Ma’lumki, katta loyihalarni bajarishda qo`yilgan masala qismlarga ajratilib, har bir qism uchun sinflar ajratiladi. Java dasturlash tilida bu sinflar orqali amalga oshiriladi. Har bir sinf o`ziga tegishli bo`lgan masalalarni bajarishga qaratiladi. Agar biz qaysidir sinfga o`zgartirish kiritmoqchi bo`lsak, bunda java dasturlash

tilida javada yozilgan kodga o`zgartish qilishimizga to`g`ri keladi va bu dastur qaytadan kompilatsiya qilinib sinfga o`zgartirishlar kiritiladi. Bu yerda sinflar bir-biriga bog`liq bo`lishi mumkin va bu o`zgarish boshqa sinflarga ham ta’sir qilishi mumkin. Shunday muammolar tug`ilib qoladiki, unda biz dastur kodiga tegmasdan turib faqat sinfni o`zgartirishimiz kerak bo`ladi. Bunday jarayonni Java dasturlash tilida Aspectga mo`ljallangan dasturlash orqali amalga oshirishimiz mumkin. Bunda biz Javada yozilgan Aspect orqali Javada tuzilgan ixtiyoriy dastur kodiga o`zgartirish qilmay turib, sinfni o`zgartirish orqali muammoni hal qilamiz. Bu juda katta loyihalarni amalga oshirishda haqiqatdan ham juda qulay hisoblanadi. Aspect deganda biz, nuqtiy-nazar orqali qarashni tushunamiz. Ushbu tatqiqotning dolzarbligi Java dasturlash tilida Aspectga mo`ljallangan dasturlash orqali yuqorida keltirilgan muammolarni hal qilish bilan belgilanadi.

Mavzuning tadqiq darajasi. Obyektga yo`naltirilgan dasturlash (Object oriented programming – OOP) tillari o`tgan asrning 90-yillaridan boshlab rivojlana boshlandi. OOP keng tarqalgan tillar qatoriga C++, C#, Java, Eyphil, Phyton kabi dasturlash tillari kiradi. AOP bo`yicha xorij mamlakatlarida ko`plab tadqiqotlar olib borilgan va ularda, asosan, ishning nazariy qismi ishlab chiqilgan. Obyektga yo'naltirilgan dasturlash bo'yicha Yevropa konferensiyasilari (ECOOP) 1997-yil iyun oyida Finlyandiyaning Springer-Verlag shahrida va Germaniyaning Heidelberg shaharlarida bo`lib o`tgan. Ularda AOP ning nazariy masalalari, tarkibiy qismlari, amaliy ahamiyati xususida bahs yuritilgan. Shu yili Kikzales G., Lamping J., Mendesh A. ning **“Aspect-oriented programming”** nomli maqolalari e’lon qilingan. Rus olimlaridan Grachev M.K., Grigoriev D.A., Maslennikov A.I. Safonov V.O. lar AOP bo`yicha tadqiqotlarni davom ettirganlar va ular tomonidan bir necha nomzodlik va doktorlik ishlari, ilmiy qo`llanmalar, darsliklar ishlab chiqilgan. Biroq o`zbek olimlarining hech biri bugunga qadar AOP bilan maxsus shug`ullanmagan.

Tadqiqotning obyekti. Tadqiqotning obyekti obyektga yo`naltirilgan dasturlash (OOP), Aspectga yo`naltirilgan dasturlash (AOP).

Tadqiqotning predmeti. Java Eclipse, sinf (class), polimorfizm, merosxo`rlik.

Tadqiqotning maqsadi va vazifalari. Tadqiqotning asosiy maqsadi mavjud AOP vositalaridagi foydalanuvchilar bilan o'zaro aloqani tashkil etish, AOP foydalanuvchi interfeysini amalga oshirish uchun bir qator yangi usullarni taklif etish va muallif tomonidan amalga oshiriladigan taklif qilingan usullarning bajarilishini tavsiflashdan iboratdir. Ushbu maqsadlardan kelib chiqqan holda quyidagi **vazifalarni** belgiladik:

–AOP vositalari uchun foydalanuvchi interfeysini joriy qilish usullarini ishlab chiqish va dasturiy ta'minotni ishlab chiqish uchun zamonaviy muhitga integratsiyalash;

–Ishlab chiqilgan usullarga asoslangan Aspect.NET Framework foydalanuvchi interfeysini joriy etish;

–Aspect.aj kengaytmasi misolida ba'zi bir muammolarni hal qilishda AOP ni yopiq tarzda ishlatish uchun Aspect.aj kengaytmasi joriy etilishini izohlash;

–Aspect.NET tizimidagi Aspect.aj quyi tizimini bir qator vazifalarni hal qilish uchun dasturiy mahsulotni ishlab chiqishda foydalanish.

Tadqiqotning asosiy masalalari va farazlari. Mazkur tadqiqot OOP (object-oriented programming – obyektga yo`naltirilgan dasturlash) bilan bevosita aloqador quyidagi asosiy masalalarni qamrab oladi:

- Sinf sutrukturasiga o`zgartirish kiritmasdan turib, unga qo`shimcha funksionallik qo`shish;
- OOP dagi o`zaro bog`liq muammolarni yechish;
- OOP ni moslashuvchanligini ta’minlash;
- Dasturning hayot siklini uzaytirsh.

Tadqiqotning metodologik asosi va tadqiq usullari. O`zbekiston Respublikasi birinchi Prezidenti Islom Karimovning milliy mafkura, ma’naviyat

va madaniy qadriyatlarga doir asarlari, AOP bo`yicha nashr etilgan darslik, o`quv qo`llanmalar ishning metodologik asosini tashkil etadi. Ishda asosan, tavsifiy metoddan foydalanildi.

Tadqiqot natijalarining nazariy va amaliy ahamiyati. Mazkur tadqiqot nazariy jihatdan asoslangan bo`lib, unda quyidagi nazariy va amaliy masalalar amalga oshirildi:

–AOP ning konsepsisiyalarini o`rganish;

–AOP ning funksionalligini oshirish;

–AOP yordamida OOP ning o`zaro bog`liq bo`lgan muammolarini yechish;

–aj. kengaytmali kodni amalga oshirish.

Tadqiqotning ilmiy yangiligi. Java aspektga mo`ljallangan dasturlash imkoniyatlarini o`rganish muammosi tadqiqiga bag`ishlangan mazkur dissertatsiyaning ilmiy yangiligi:

1) AOP konsepsiyasi O`zbekistonda ilk bora amalga oshirilayotganligi;

2) Tadqiqot ishida ishlab chiqilgan va joriy qilingan aspektlarni foydalanuvchi qo'llash usuli (hozirgi vaqtda Aspect.NET Frameworkdan tashqari hech bir AOP vositasi bunday imkoniyatga ega emas) ishlab chiqish;

3) AOP yordamida yasalgan aj. kengaymali dasturlarning amaliyotda keng qo`llanila boshlagani;

Tadqiqotning tarkibi. Dissertatsiya kirish, uch bob, umumiy xulosalar, foydalanilgan adabiyotlar ro`yxati kabi qismlardan tuzilgan. Ishning umumiy hajmi 72 sahifadan iborat.

I bob. Java dasturlash tili

1.1 Java dasturlash tili strukturasi

Jeyms Gosling, Mayk Sheridan va Patrik Nauton 1991 yil iyun oyida Java tilini ishlab chiqdi [5]. Java aslida interaktiv televizor uchun mo`ljallangan edi, ammo u vaqtdagi raqamli kabel televideniyesi sanoati uchun anchagina rivojlangan til edi [6]. Java tili dastlab Goslingning ofisi tashqarisida turgan eman daraxti nomiga Oak deb atalgan. Keyinchalik esa loyiha Green nomi bilan atalib va nihoyat hozirgi nomi Java qahvasiga atab Java deb o`zgartirildi [8]. Gosling C/C++ uslubidagi sintaksisi bilan Java dasturini ishlab chiqardi.

Sun Microsystems birinchi ommaviy dasturni 1996 yilda Java 1.0 sifatida e`lon qildi [17]. Bu dasturlash tili "Bir marta yoz, qayerda bo`lsa ham ishlat" (WORA) jumlalari bilan ommaga taqdim etildi. Java 1.0 juda xavfsiz va konfiguratsiya qilinadigan, xavfsizlikni ta`minlaydigan bu tarmoq va faylga kirish cheklovlariga ruxsat berdi. Katta veb-brauzerlar tez orada veb-sahifalardagi Java ilovalarini ishga tushirish qobiliyatiga qo`shildi va Java tezlik bilan mashhur bo`ldi. Java 1.0 kompilyatori Java 1.0 tilining spetsifikatsiyasi bilan qattiq rioya qilish uchun Arthur van Hoff tomonidan Java-da qayta yozildi [*10*]. Java 2 ning (1998 yil dekabr - 1999 yillarda J2SE 1.2 sifatida chiqarilgan) kelishi bilan yangi versiyalarda turli xil platformalar uchun bir nechta konfiguratsiyalar yaratildi.

2018 yil 20 martidan boshlab Java 8 va 11-lar rasmiy ravishda qo`llab-quvvatlanadi. Java-ning asosiy versiyalari va ularning chiqarilish kunlari:

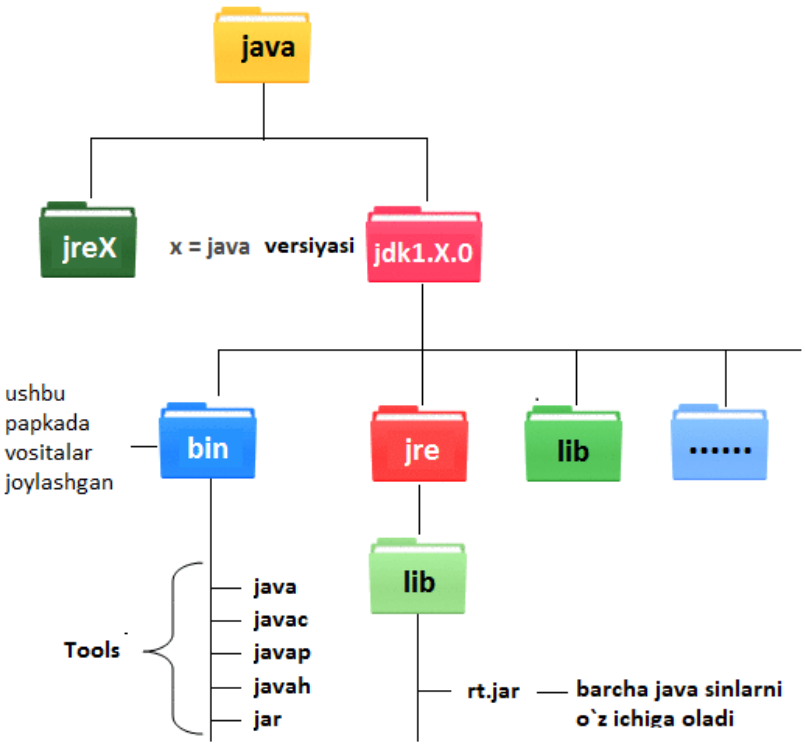
JDK 1.0 (23 yanvar 1996 yil) [*16*], JDK 1.1 (19 fevral 1997 yil), J2SE 1.2 (8 dekabr 1998 yil), J2SE 1.3 (8-May 2000 yil), J2SE 1.4 (6 fevral 2002 yil), J2SE 5.0 (30 sentyabr 2004 yil), Java SE 6 (11-dekabr, 2006 yil), Java SE 7 (2011 yil 28 iyul), Java SE 8 (2014 yil 18 mart), Java SE 9 (21 sentyabr, 2017 yil), Java SE 10 (20 mart, 2018 yil), Java SE 11 (25 sentyabr, 2018 yil) [10], Java SE 12 (19 mart 2019 yil).

Java dasturlash tilida ishlash uchun avvalo java dasturlash muhitlaridan birini kompyuterga o`rnatish kerak. Bunday muhitlarga Eclipse, Intelleje Ide va h.k lar kiradi. Muhitlarni o`rnatishda o`rnatilgan programmaning yo`li (path)ni ko`rsatishimiz kerak.

Javada **path** va **classpath** orasidagi farq qanday degan savolga javob beraylik.

Path – o`zgaruvchan Java, Javactır, javap, javah, jar, appletviewer kabi barcha Java vositalari yo`lini ta`minlash uchun o`rnatiladi. Java dasturini ishlatish uchun biz java vositasidan foydalanamiz va Java kodini javak vositasini ishlatish uchun ishlatamiz. Ushbu vositalar bin papkasida mavjud, shuning uchun biz upto bin papkasini o`rnatamiz.

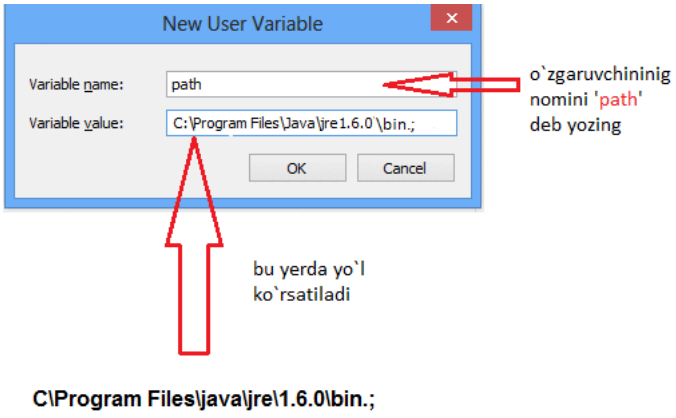
Classpath – o`zgaruvchilari bizning dasturimizda ishlatiladigan barcha Java sinflarini ko`rsatish uchun o`rnatiladi. Biz lib / rt.jar qadar classpath belgilangan, shuning uchun barcha sinflar lib / rt.jar mavjud (1-rasm).



1-rasm. Java strukturasi

Path o`zgaruvchisi java, javac, javap, javah, jar, appletviewer va hokazolar kabi barcha vositalarni ishlatish uchun o`rnatiladi.

Masalan, "C:\Program Files\Java\jdk1.6.0\bin" (2-rasm).

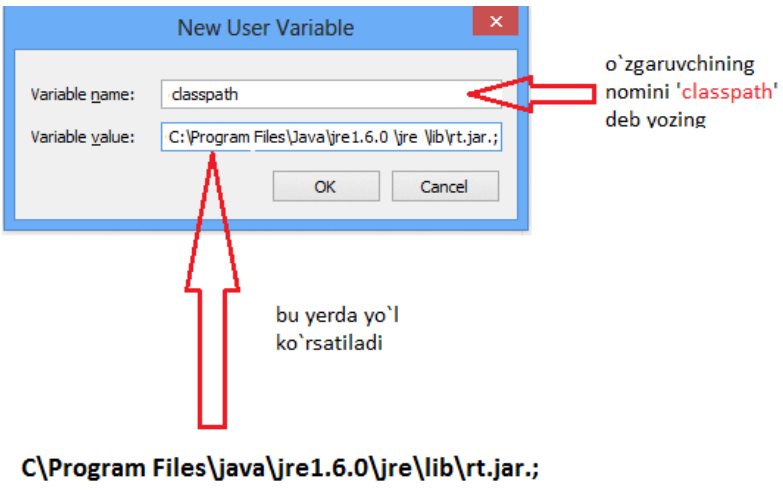


2-rasm. Path (yo`l) ni ko`rsatish

Barcha vositalar ming papkasida mavjud, shuning uchun biz upto bin papkasini o`rnatamiz.

Classpath o`zgaruvchilari dasturimizda ishlatiladigan barcha sinflar uchun yo`lni belgilash uchun ishlatiladi, shuning uchun klass usulini rj.jar-ga o`rnatamiz. rt.jar faylida barcha .class fayllari mavjud. Biz rt.jar faylini dekompressiyadan chiqarganimizda barcha .class fayllarini olamiz.

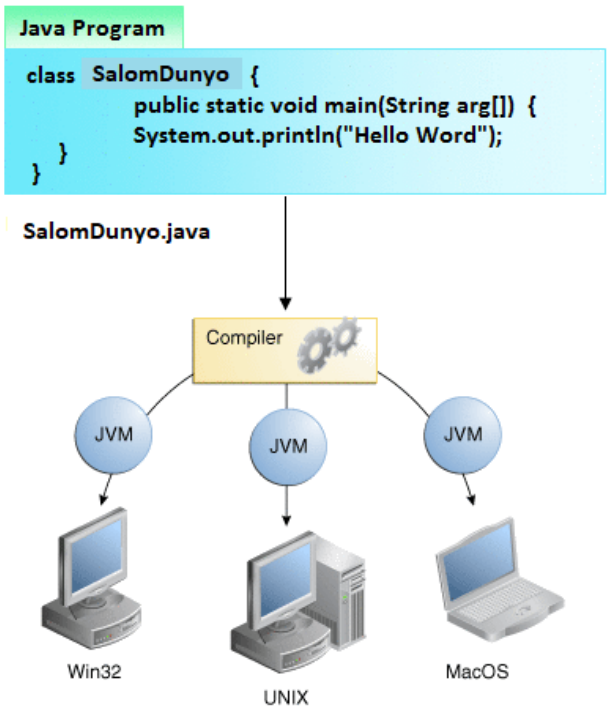
Masalan, "C:\Program Files\Java\jre1.6.0\jre\lib\rt.jar" (3-rasm).



3-rasm. Sinf yo`lini ko`rsatish

Java compilyatori ko`plab platformalarga mos tushadi. Agar dasturlash tili yoki texnologiyasi tizimning platformasiga bog`liq bo`lsa, u holda har bir platform uchun alohida mustaqil ishlovchi kod talab qilinadi. (Platforma operatsion tizimni ifodalaydi) [12].

Java platformalar uchun mustaqil dasturiy tildir, chunki tizimingizda jdk dasturini o`rnatganingizda avtomatik ravishda JVM (Java Virtual Machine) tizimi o`rnatiladi. Har bir operatsion tizim uchun alohida JVM mavjud, bu esa .class fayli yoki byte kodini o`qishga qodir. Java kodingizni kompilyatsiya qilganda, .class fayli javac kompilyatori tomonidan ishlab chiqariladi, JVM kodlar JVM tomonidan o`qilishi mumkin va har bir operatsion tizim JVM-ga ega, shuning uchun JVM platformaga qaram, ammo JVM java tili platformasi mustaqil bo`lib qoladi (4-rasm).

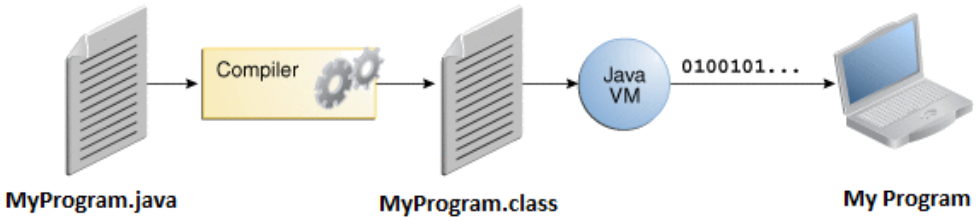


4-rasm. JVM va OS

Izoh: Java platforma mustaqil, lekin jvm platformaga bog`liq.

Java dasturini tuzish va ishga tushirish. Java dasturlash tilida, barcha manba kodlari birinchi bo`lib tekis matnli fayllarda yoziladi va .java kengaytmasi

bilan saqlanadi. kompilyatsiyadan so`ng, .class fayllari javac kompilyatori tomonidan generatsiya qilinadi. A .class faylida protsessor uchun mahalliy kod mavjud emas. Bu kodlarni JVM baytkodlarga aylantirib beradi (5-rasm) [8].



5-rasm. JVM ning ishlash prinsipi

Java dasturini kompilyatsiya qilish uchun qadamlar:

- Birinchi Java dasturini .java kengaytmasi bilan sinfi nomi bilan bir xil saqlang. Masalan, Sum.java
- Kompilyatsiya: javac Filename.java. Misol uchun, javac Sum.java

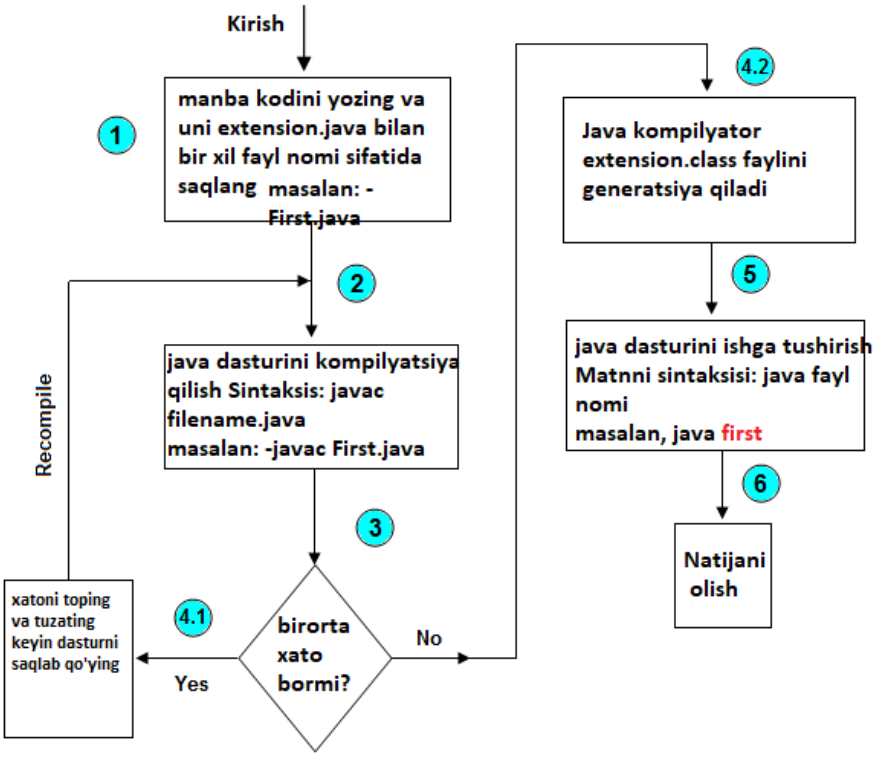
Eslatma: Bu yerda javac - Java dasturini kompilyatsiya qilish uchun ishlatiladigan asboblar yoki dastur dasturlari yoki exe fayllari.

Java dasturini ishga tushirish uchun qadamlar:

- Java dasturini ishlatish uchun **java** uskunalaridan foydalaning.
- **Run by**: java fayl nomiMisol: java sum

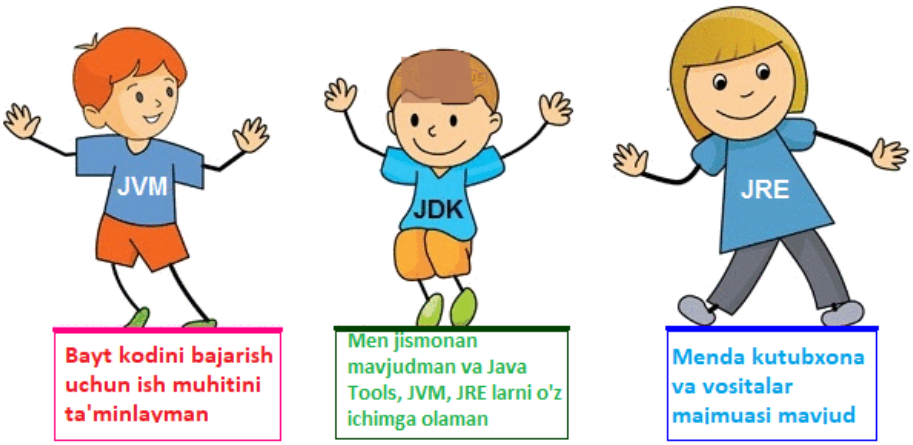
Eslatma: Bu yerda java - Java dasturini ishlatish uchun ishlatiladigan asboblar yoki dastur dasturlari yoki exe fayllari.

Java dasturini tuzish va bajarish uchun qadamlar jadvalda ifodalangan quyidagi bosqichlar ketma-ketligi java dasturini kompilyatsiya qilish va java dasturlarini bajarishdan iborat (1-jadval).



1-jadval.

**JDK, JVM va JRE o`rtasidagi farq**. Jvm, Jre, Jdk, bu java tilining barcha asosini tashkil etadi. Har bir komponentning alohida ishi mavjud. Jdk va Jre jismonan mavjuddir, lekin Jvm - bu abstrak mashinadir, ya`ni jismonan mavjud emas (6-rasm).

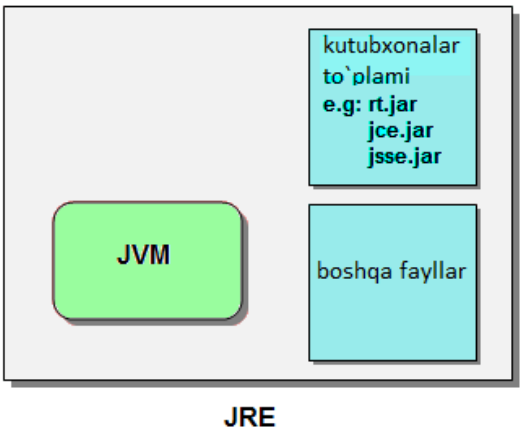


6-rasm. JVM, JDK va JRE

JVM (Java Virtual Machine) dasturiy ta`minot. Bu java bytecode bajarilishi mumkin bo`lgan ish muhitini ta`minlovchi xususiyatdir. Bu jismonan mavjud emas [11].

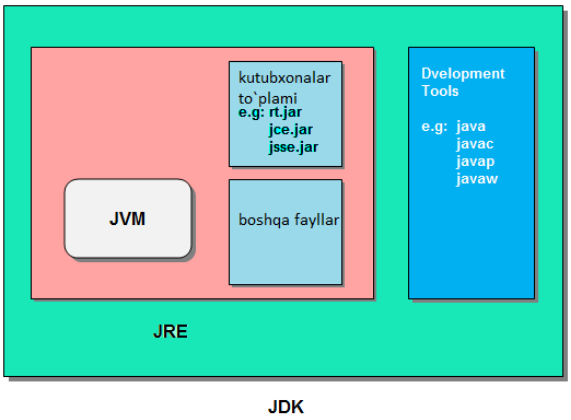
JVM barcha apparat va dasturiy ta`minot uchun bir xil emas, masalan, JVM oynasi uchun farq qiladi va Linux VJM uchun farq qiladi. JVM, JRE va JDK platformaga bog`liq, chunki har bir OS konfiguratsiyasi farqlanadi. Biroq, Java platformasidan mustaqildir.

JRE (Java Runtime Environment) Java Development Kit (JDK) ning bir qismidir. Java dasturini ishlab chiqish uchun kutubxonalar va vositalar to`plami mavjud. Java Runtime Environment Java ilovasini bajarish uchun minimal talablarni bajaradi. Bu jismonan mavjud (7-rasm). JVM ishlatish vaqtida foydalanadigan kutubxonalar + va boshqa fayllar to`plami mavjud.



7-rasm. JRE ning tuzilishi

JDK (Java Development Kit) asosiy komponentlar hisoblanadi. Bu jismonan mavjud. Dasturiy vositalar to`plami va JRE, JVM ning to`plami (8-rasm).



8-rasm. JDK ning tuzilishi

Java dasturining tuzilishi – ishlab chiquvchi tomonidan sanoat dasturchisiga chiqarilgan standart formatdir.

Sun Micro System tomonidan java dasturlarini ishlab chiqish uchun quyidagi tuzilma yaratilgan.

```
//package tafsilotlari (kutubxonalarni import qilish)
import java.util.*;

//sinf nomi
public class SalomDunyo {

    public SalomDunyo() {
        // TODO Auto-generated constructor stub
        //maydonlar
         int a,b,c;
        //metodlar
         void gapir();
         }
    //asosiy metod
public static void main(String[] args){
    //kod
    System.out.println("Salom dunyo");
}
}
```

Paket (Package) – bu sinflar, interfeyslar va kichik paketlar to`plamidir. Sub to`plami sinflarni, interfeyslarni va sinf ostilarini to`plashni o`z ichiga oladi. Java.lang. *; paketi jimlikda import qilinadi va bu paket odatiy paket sifatida tanilgan.

Sinf (Class) – foydalanuvchi tomonidan aniqlangan ma`lumotlar turini ishlab chiqishda foydalaniladigan kalit so`z bo`lib, har bir java dasturi sinf tushunchasi bilan boshlanishi shart.

"**ClassName**" – sinfning nomi sifatida ko`rib chiqiladigan java joriy o`zgaruvchining nomini ifodalaydi va javada har bir sinf nomi foydalanuvchi tomonidan belgilangan ma`lumot turi sifatida ko`rib chiqiladi.

Data memebers – Ma`lumot turi (tip) sinfi nomiga asosan tanlab olinadigan namuna yoki statikni ifodalaydi.

Foydalanuvchi tomonidan aniqlangan usullar operatsiyalarni bir martadan yoki har bir vaqtda bajarish uchun mo`ljallangan nusxani yoki statikni ifodalaydi.

main() – Har bir java dasturi main() metodidan bajarishni boshlaydi. Shuning uchun main metodi dastur ishlatuvchisi sifatida tanilgan.

Java ning main metodi hech qanday qiymat qaytarmaganligi sababli main() metodi oldidan **void** ishlatiladi.

Main metodini har bir java dasturchisi foydalana olishi kerak va shuning uchun kirish spetsifikatori **public** bo`lishi shart.

Java da main metodi parameter sifatida satr ob`ektlarining massivini olishi kerak. Shuning uchun **main(String[] args)** bo`ladi.

Main metodining bloklari foydalanuvchi tomonidan belgilangan metodlarni chaqiradi va ish bajariladigan so`zlarning majmuini ifodalaydi.

Java dasturlash tilining har bir faylida qanday sinflar ishlatilishidan qat`iy nazar, ularni main() metodini o`z ichiga oldi. Fayl nomi **.java** kengaytmali fayl nomi sifatida berilishi kerak [18].

**Java dasturlash tilida ma`lumotlarning tur (tip)lari.** Ma`lumot tiplari ma`lumotlar uchun yetarli xotira maydonini ajratish uchun ishlatiladigan maxsus

kalit so`z bo`lib, boshqacha aytganda ma`lumot turi, kompyuterning asosiy xotirasi (RAM)dagi ma`lumotlarni ko`rsatish uchun ishlatiladi [25].

Umuman olganda, har bir dasturlash tili uch toifadagi ma`lumotlar tiplarini o`z ichiga oladi. Ular

- Asosiy yoki fundamental ma`lumot tiplari
- Konteyner ma`lumotlar tiplari
- Foydalanuvchi tomonidan aniqlangan ma`lumot tiplari.

Primitiv (Sodda) ma`lumotlar turlari o`zgaruvchilar bizga birgina qiymatni saqlashga imkon beradi, lekin bizga hech qachon bir xil turdagi bir nechta qiymatlarni saqlashga imkon bermaydi. Bu ma`lumotlar bir xil bo`lib, uning o`zgaruvchisi bir vaqtning o`zida maksimal qiymatga ega bo`lishi mumkin [15].

Masalan,

```
int a; // to`g`ri
a=10; // to`g`ri
a=10, 20, 30;  // noto`g`ri
```

Konteyner (yig`ilgan) ma`lumotlar tiplari o`zgaruvchilarni bizga bir xil turdagi bir nechta qiymatini saqlashga imkon beradi. Lekin ular hech qachon turli tipdagi bir nechta qiymatlarni saqlashga imkon bermaydi. Ular o`zgaruvchiga o`xshash turdagi bir nechta qiymatga ega bo`lgan ma`lumot tiplari. Umuman olganda ma`lumotlar turini massivdan foydalanib ham olish mumkin.

Masalan.

```
int  a[] = {10,20,30};  // to`g`ri
int b[] = {100, `A`, "ABC"};   //  xato
```

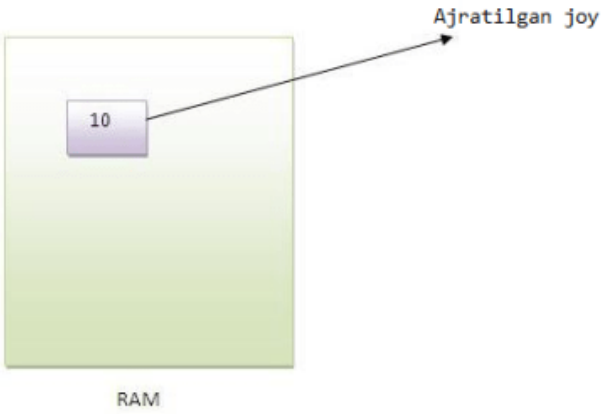
Foydalanuvchilarning aniqlangan ma`lumot tiplari - tillarning tegishli xususiyatlaridan foydalangan holda dasturchilar tomonidan ishlab chiqilgan ma`lumotlar.

Foydalanuvchi tomonidan identifikatsiya qilingan ma`lumotlar tiplari bilan bog`liq parametrlarga bir xil tipdagi yoki har xil tipdagi yoki har ikkalasida bir

nechta qiymatlarni saqlay olamiz. Bu o`zgaruvchining o`zgaruvchan tipdagi bir nechta qiymati bo`lishi mumkin bo`lgan ma`lumot tipi java da sinf tushunchasi yordamida amalga oshiriladi [11].

Masalan, talaba s = new talaba();

O`zgaruvchi – xotiradan ajratilgan, himoyalangan maydon nomidir (9-rasm)**.**



9-rasm. Xotirada o`zgaruvchi uchun ajratilgan joy

Oddiy qilib tushuntiradigan bo`lsak, o`zgaruvchi – ma`lum bir turdagi ma`lumotni o`zida saqlovchi va o`lchami chegaralangan idish. Tushunarliroq bo`lishi uchun bir ikkita hayotiy misollar keltiramiz: meva solish uchun tayyorlangan savatga suv sola olmaymiz o`zgaruvchilar ham shunday bir turdagi o`zgaruvchi uchun ajratilgan joyga boshqa turdagi o`zgaruvchini saqlay olmaymiz [34].

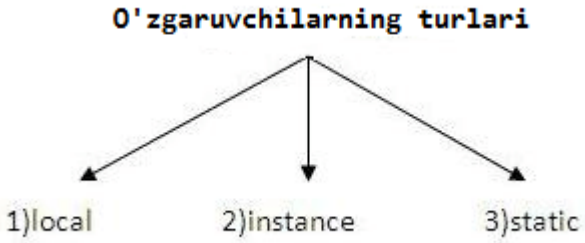
4 litrlik idishga 5 litr suv quya olmaymiz, chunki idishga 4 litr suv sig`adi. O`z o`zidan kelib chiqadiki 5 litrlik suvni saqlash uchun kattaroq idish tanlashimiz kerak. O`zgaruvchilar ham shunday ma`lumotning o`lchami xotiradan ajratilgan joydan oshib ketsa dastur xatolik yuz beradi [33].

5 *bayt*lik butun sonni ma`lumot turi *int* bo`lgan o`zgaruvchiga saqlay olmaymiz, chunki int = 4 *bayt*. Bu turdagi ma`lumotni saqlash uchun *long* dan foydalanamiz.

Types of Variable – o`zgaruvchi turlari

Javada 3 ta o`zgaruvchilar turi mavjud (10-rasm):

1. local variable
2. instance variable
3. static variable



10-rasm. O`zgaruvchilar

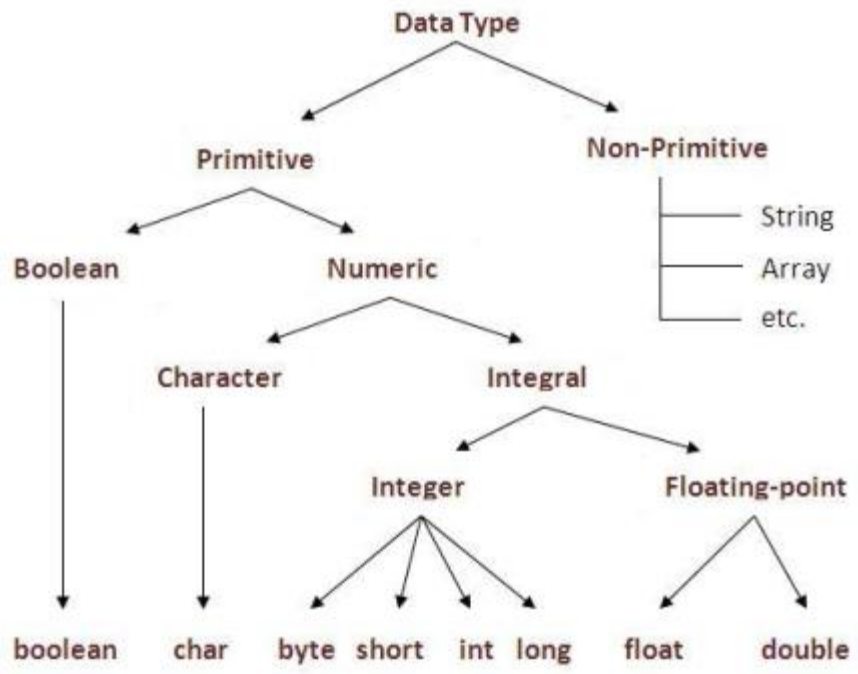
- **local variable** – funksiya ichida e`lon qilinadi va bu o`zgaruvchilar lokal (mahalliy) o`zgaruvchilar deyiladi.
- **instance variable** – class ichida e`lon qilinadi
- **static variable** – statik deb e`lon qilingan o`zgaruvchi static o`zgaruvchi deyiladi. Bu local(mahalliy) bo`lishi mumkin emas [19].

Misol uchun:

```
class A {
 int data=50; //instance variable
 static int m=100; //static variable
 void method() {
 int n=90; //local variable
 }} //endof
```

**Javada ma`lumot turlari (data types)**. Javada ma`lumot turlari 2 ta (11-rasm):

1. Sodda (*primitive*)
2. Sodda bo`lmagan (*non primitive*)



11-rasm. O`zgaruvchi tiplar

Ma'lumotlar turlari	Oraliq qiymati	Xotira xajmi
boolean	false	1 bit
char	'\u0000'	2 byte
byte	-128 , 127	1 byte
short	$(-2^{15}$, 2^{15} - 1)	2 byte
int	$(-2^{31}$, 2^{31} - 1)	4 byte
long	$(-2^{63}$, 2^{63} - 1)	8 byte
float	3.4e-38, 3.4e38	4 byte
double	1.7e-308, 1.73e308	8 byte

2-jadval. Tiplarning oraliq qiymatlari.

Quyidagi misolda ham ko`rishimiz mumkin:

```
public class Varibles {
public static void main(String[] args) {
System.out.println(Character.SIZE / 8 + " byte");
System.out.println(Byte.SIZE / 8 + " byte");
System.out.println(Short.SIZE / 8 + " byte"); } }
```

Javada dasturlash tilida kiritish va chiqarish operatorlari.

**Ma`lumotlarni chiqarish operatori.** Java kirish va chiqishni amalga oshirish uchun dastur doirasida ko`p sinflar va metodlarni taqdim etadi.. Grafik foydalanuvchi interfeysi dizayni uchun Java-da sinflar mavjud, shuningdek, matn va raqamlarni ko`rsatish va kiritish uchun turli metodlar bor. Grafik obyektlar, tasvirlar, tovushlar, veb-sahifalar va sichqoncha hodisalari (masalan, sekin urish, sichqonchani sinchkovlik bilan olib tashlash kabi) mavjud. Bundan tashqari, ushbu kirish va chiqishning ko`pchilik metodlari yakka o`zi yoki dasturlarda ishlatilishi mumkin.

Java "standart chiqish" qurilmasi chiqishni amalga oshiruvchi System.out nomli o`rnatilgan statik ob`ektni ta`minlaydi. Ko`pgina operatsion tizimning qobiqlari foydalanuvchilarni qayta yo`naltirishga imkon beradi [15].

Java konsoli oynasiga tegishli. System.out ob`ektiga misol sifatida java.io.PrintStream sinfini keltirish mumkin. Bu sinf buferlangan chiqish oqimi uchun metodlarni belgilaydi. Bu belgilar vaqtinchalik joyga, ya`ni buferga qo`yiladi, buferdan esa konsol oynasi belgilarni chop etishga tayyor bo`lganda bo`shatiladi [16].

Xususan, java.io.PrintStream sinfi quyidagi metodlarni taqdim etadi.

print(String *s*): *s* satrni chop etish.
print(Object *o*): *o* Ob`ektini toString usuli yordamida chop eting
print(*baseType b*): *b* bazaviy tipni chop etish.
println(String *s*): *s* satrni va keying yangi satri chop etish.
println(Object *o*): print(*o*) ga o`xshash, keyin yangi satr belgisini chop etish.
println(*baseType b*): print(*b*) ga o`xshash, keyin yangi satr belgisini chop etish.

Masalan, System.out.print("Java values: ");

```
System.out.print(3.1416);
System.out.print(`,`);
System.out.print(15);
System.out.println(" (double,char,int).");
```

**Ma`lumotlarni kiritish operatori.** Java konsolining oynasiga chiqish uchun maxsus ob`ekt bo`lgani kabi, Java da kirishni amalga oshirish uchun System.in deb nomlangan maxsus ob`ekt ham mavjud. Texnik jihatdan kirib, aslida "standart kirish" jimlik bo`yicha o`z belgilarini aks ettiradigan kompyuter klavishi hisoblanadi.

Java konsoli – System.in obyekti standart bilan assotsiatsiyalangan ob`ekt

kirish qurilmasidir. Ushbu obyekt bilan kirishni oddiy usulda ifodalashdan foydalanib, skaner ob`ektini yaratish uchun ham ishlatish mumkin [12,39].

new Scanner(System.in)

Scanner sinfi berilgan kirish oqimidan o`qish uchun bir qator qulay metodlarga ega, ulardan birini quyidagi dasturda ko`rishimiz mumkin:

```
import java.util.Scanner;
public class InputExample {
public static void main(String[ ] args) {
Scanner input = new Scanner(System.in);
System.out.print("Tug`ilgan yilingizni kiriting: ");
double age = input.nextDouble( );
System.out.print("maksimal yurak urish tezligini kiriting: ");
double rate = input.nextDouble( );
double fb = (rate − age) * 0.65;
System.out.println("Sizning ideal yurak urish tezligingiz " + fb);
} }
```

1.2 Java dasturlash tilida boshqaruv operatorlari

Java boshqaruvidagi operatorlar boshqa yuqori darajali tillarga o`xshash. Ushbu bobda Java-ning boshqaruv operatorlarining asosiy tuzilishi va sintaksisini, shu jumladan, metodlar natijalari, **if** operatori, **switch** operatori, **while** va **for** operatorlari haqida batafsil to`xtalamiz.

Java dasturlash tilida ikkita tanlash operatori bo`lib ular **if** va **switch** lardir. Masalani qo`yilishiga qarab ularning birini ishlatish mumkin. If operatori kodlashni ikkita yo`ldan biriga burib yuboradi. Hayotda shart tekshirish operatorlaridan ko`p foydalanamiz. Tasavvur qiling siz bekatda turipsiz sizga, biror yo`nalishdagi avtobus kerak. Uzoqdan kelayotgan avtobusga ko`zingiz tushadi va ko`zingiz orqali ko`rgan ma`lumotingiz miyyangizga uzatiladi. Shundan so`ng miyyangiz kelgan ma`lumot asosida shart tekshiradi. Agar yo`nalish raqami siz kutgan raqam bo`lsa, avtobusga chiq aks holda keyingisini kut degan buyruqni tanaga miyya uzatadi. Bu jarayon shunday tez bo`ladiki hatto sezmaymiz ham. Bu jarayonlar hayotimizning har daqiqasida, soniyasida yuz beradi. Endi bu shart tekshirishlarimiz javada qanday yozilishini ko`rib chiqamiz.

If tanlash operatori turlari

- if ifodasi
- if-else ifodasi
- nested if ifodasi
- if-else-if ladder (narvon)

If operatori shart tekshiradi agar shart **true**(rost) bo`lsa if ning tanasidagi operator(lar) bajariladi, aks holda shart bajarilmaydi. If operatori, butun kod ishlashini ikkita yo`ldan biriga yo`naltirib yuboradi, ya`ni qo`yilgan shart asosida kompilyator biror yo`lni tanlab, o`z ishini davom ettiradi.

```
if(shart){
 operator1;  operator2;  … }
```

Tasavvur qiling, mashinani bilan sayohatga chiqmoqchisiz. Manzilingizgacha bo`lgan masofani bosib o`tish uchun sizga ma`lum miqdorda yoqilg`I kerak bo`lsin. Bakdagi yoqilg`i ko`rsatgichiga qaraganimizda agar yetarlicha yo`ilg`I bo`lmasa yoqilg`I quyib olinsin, aks holda yo`lda davom ettirilsin.

Bu jarayonni javada dastur ko`rinishi:

```
public class IFClass {
    public static void main(String[] args) {
        int kerakli_benzin=25;
  int bakdagi_benzin=13;
        if(bakdagi_benzin< kerakli_benzin){
            Sytem.out.print("Yoqilg`I quyib olinsin!");
        }
  else Sytem.out.print("Yoqilg`I quyish shart emas.");
       }
}
```

Java IF-else operatori ta`rifi.

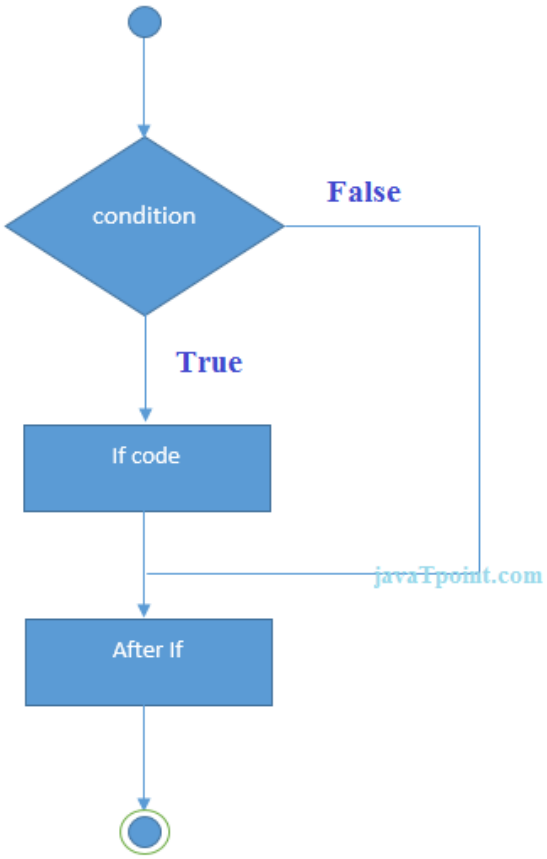
Shart tekshirish jarayonida shart *true* bo`lsa, if blokdagi amal bajariladi, aks holda *else* blokdagi amal bajariladi (12-rasm).

```
if(condition){
//code if condition is true
}
else{
//code if condition is false
}
```

Misol: Berilgan sonni toq yoki juftligini bilish dasturi:

```
public class IfElseClass {
    publicstatic void main(String[] args) {
        int number=15;
        if(number%2==0){
        out.println("juft son");
        }
        else{
            out.println("toq son");
        }
    }
}
```

Ekranda: “*Toq son*” degan yozuv chiqadi.



12-rasm. Shart operatori

Java IF-else-if tanlash operatori – o`zbekcha qilib aytganda unisi bo`lmasa bunisi, bunisi bo`lmasa keyingisi degan ma`noda. if-else-if ketma-ketligi shartlar ko`p bo`lgan hollarda ishlatiladi (13-rasm).

if(condition1){ //code to be executed **if** condition1 is **true**

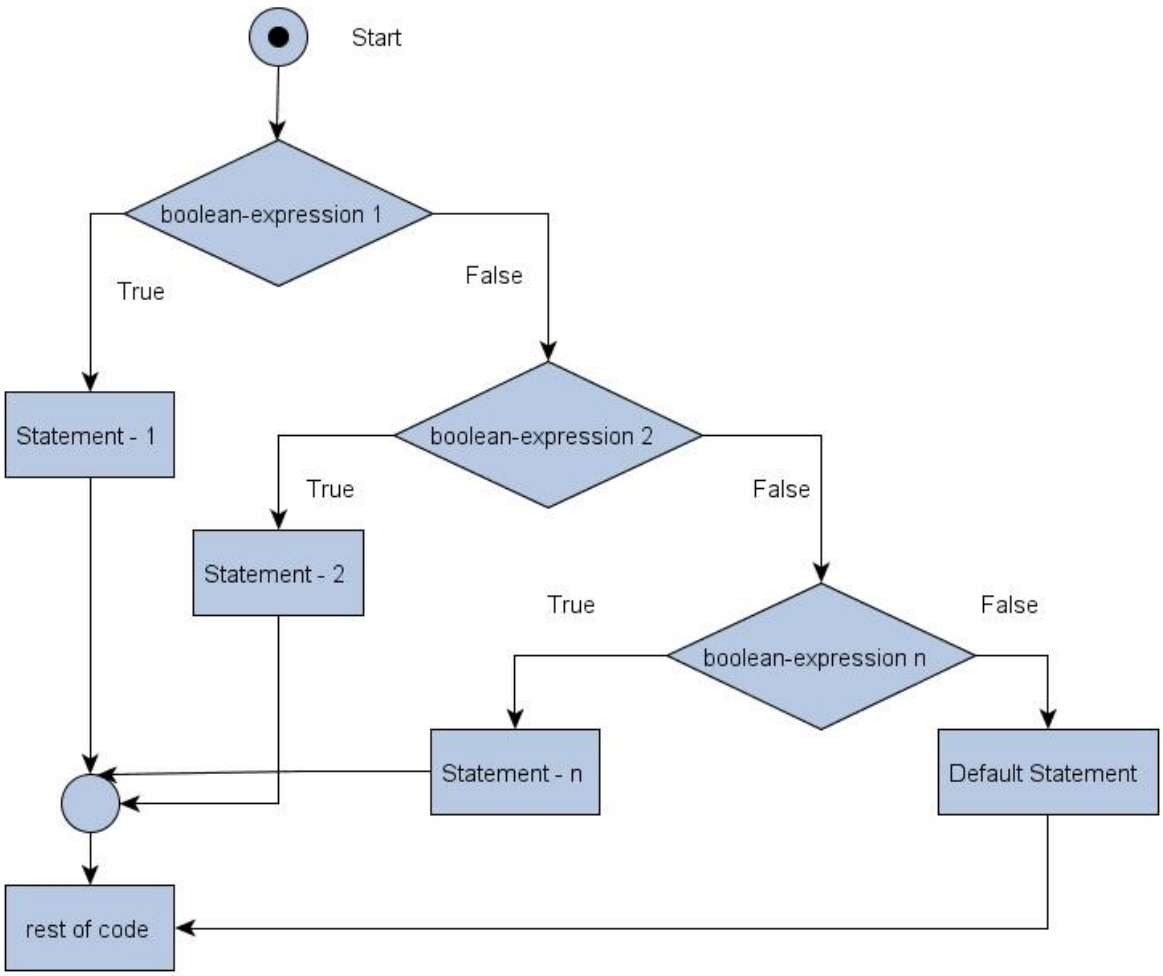
} **else if**(condition2){ //code to be executed **if** condition2 is **true**

} **else if**(condition3){//code to be executed **if** condition3 is **true**

} ...

else { //code to be executed **if** all the conditions are **false**

}



13-rasm. Ichma-ich shart operatori

```
public class IfElseIfExample {
    public static void main(String[] args) {
        int ball=65;
        if(ball<50){
            System.out.println("yiqildi");
        } else if(ball>=50 && ball<60) {
            System.out.println("D daraja");
            } else if(ball>=60 && ball<70) {
            System.out.println("C daraja");
            } else if(ball>=70 && ball<80) {
            System.out.println("B daraja");
            } else if(ball>=80 && ball<90) {
            System.out.println("A daraja");
                        }
             else if(ball>=90 && ball<100) {
            System.out.println("A+ daraja");
            } else {
            System.out.println("Invalid!");
                                    }
        }
}
```

Ekranda : C daraja

Java Switch tanlash. Switch operatori – dastur ishlashini berilgan qiymatga mos holda biror yo`nalishga o`zgartirib berishni ta`minlaydi. Bu operator bir necha variantlardan kerakligini tanlab olishda, « **if**» operatori o`rnida ishlatiladi. Albatta, barcha tanlovlarni «**If**» orqali amalga oshirish mumkin,. Farqi u shart *true* bo`lguniga qadar tekshiradi. Shart to`g`ri bo`lgandan

keyin *switch* operatoridan boshqa qiymatlarni tekshirmaydi. Bu esa bizga bir nechta qadam yutish imkonini yaratadi.

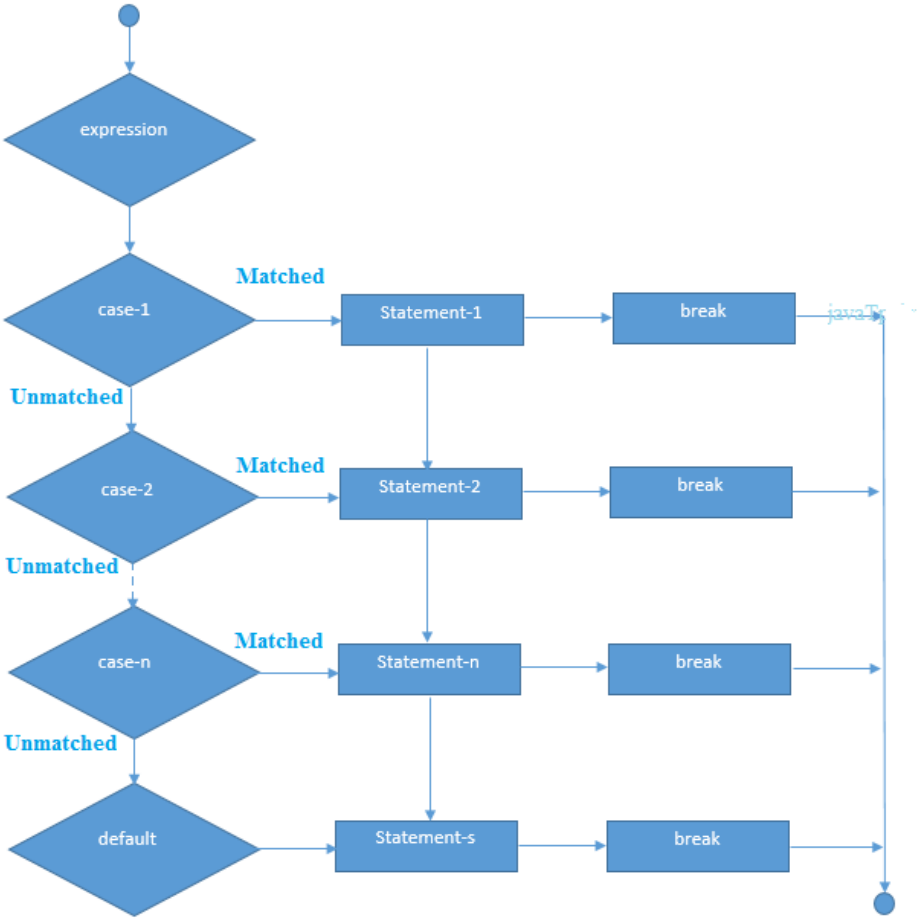
```
switch(shart){
   case value1:
   //code to be executed;
   break;  //optional
   case value2:
   //code to be executed;
   break;  //optional
   ......
   default:
   //hech qaysi shartni qanoantlantirmasa;
 }
```

Bu operator qanday ishlaydi? « **Switch**» operatoriga berilgan qiymat, «**case**» operatoridagi qiymatlar bilan birma-bir solishtiriladi. Agar qiymat biror variant bilan mos tushsa, shu variantga tegishli blok ichidagi operator yoki operatorlar bloki ishlaydi va «**break**» orqali bu tanlov(switch) yakuniga yetadi. Agar qiymat hech qaysi variant bilan mos tushmasa, «**default**» operatori ishlaydi va uning ichidagi operatorlar dastur ishlashini davom ettiradi.

«**default**» bloki bo`lmasligi ham mumkin, bunda agar variantlar orasida bizning qiymat topilmasa, «**switch**» hech qanday xabar bermaydi.

«**break**» operatori «**switch**» operatoridan chiqishni bildiradi, shuning uchun har bir «**case**» operatoridan keyin qo`yilgan. Agar bu operatorni olib tashlasak, bizning qiymat variantlar orasidan topilsa ham, «**switch**» operatori ishlayveradi va hamma «**case**» operatoriga murojat qilaveradi. Bu dasturning xato ishlashiga olib keladi (14-rasm).

Masalan, biror son kiritiladi, bu son haftaning qaysi kuniga to`g`ri kelishini topish dasturini ko`rib chiqaylik:



14-rasm. Switch case

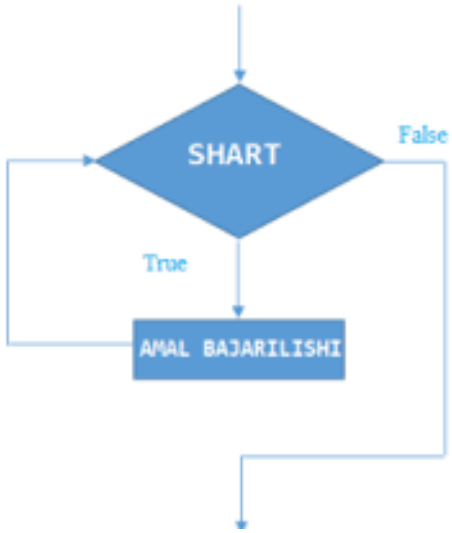
```
    switch (new Date().getDay()){
case 0:day = "Sunday";
  break;
case 1:
  day = "Monday";
  break;
case 2:day = "Tuesday";
  break;
case 3:
  day = "Wednesday";
  break;
case 4:
```

```
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case 6:
    day = "Saturday";
}
```

Takrorlanuvchi sikl operatorlari. Java dasturlash tilida sikl operatorlari **for, foreach, while** operatorlari hisoblanadi. O`z nomi bilan ma`lum bo`lib turibdi sikl operatorlar yordamida takrorlanuvchi jarayonlarni algoritmlaymiz. Hafta kunlarining takrorlanishi, yerning quyosh atrofida aylanishi, yoki bir operatsiya bajarishining ayni bir qadami takrorlanishi va h.k.larni misol qilib keltirish mumkin. Takrorlanuvchi ish harakatlar qandaydir shartlar asosida bajariladi. Ularning boshlang`ich, oxirgi nuqtalari va bajarilish davriyligi mavjud bo`ladi (15-rasm). For operatorining sintaksisi quyidagicha:

for (boshlang`ich qiymat**;** shart**;** bajarilishi qadami**)**

Tushunarliroq bo`lishi uchun blok sxema orqali ko`rib chiqamiz



15-rasm. While sikl operatori

Sikl boshlangandan sikl sharti false(yolg`on) qiymat bo`lmaguncha davom etadi. Sikl sharti false bo`lgan sikl tugatiladi.

Misol: 1 dan 10 gacha bo`lgan sonlarni ekranda chiqarish

```
Public class ForExample {
 publicstatic void main(String[] args) {
 for(int i=1; i<=10; i++){
out.print(i+ “,”);}  }}
```

Ekranda :1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Siklning boshlang`ich qiymati 1 ga teng takrorlanishi ham 1 ga teng i ning qiymati o`sib borib qiymat 10 ga teng bo`lmaguncha davom etadi. Yanada tushunarliroq bo`lishi uchun boshqa bir misolni ham ko`rib chiqsak. Berilgan sondan shu songacha bo`lgan raqamlar orasida 4 yoki 8 ga bo`linadigan raqamlar sonini topish.

```
public class BulinuvchilarSoni{
public static void main(String[] args) {
int k = 0;
for (int i = 1; i < 50; i++) {
if (i % 4 == 0 || i % 8 == 0) {
k++; }}
System.out.println(k);}}
```

Natija :22

Javada For-each Loop. For-each loop dan massivlar yokiy to`plamlarni yurguzishda foydalaniladi. For-each loop dan foydalanish qulay va oson. Chunki bunda siklning boshlang`ich va oxirgi qiymatlari ko`rsatilmaydi. Bunda elementlar birma bir qaytarib uning qiymatlari ko`rsatiladi .

```
for(Type var : array) { //code to be executed }
public class ForeachExample {
```

```
public static void main(String[] args) {
String arr[]={“Java”, “C++”, “C#”, “Pascal”};
 for(int i:arr){
 System.out.print(i+”, ”); }}}
```

Ekranda : Java, C++, C#, Pascal

«**While**» operatori dastur tuzishda ko`p ishlatiladigan sikl operatori hisoblanadi. Bu operator bir yoki bir necha operatorlar guruhini qo`yilgan shart yolg`on (*false*) bo`lguncha bajaradi. Qachonki shart rost bo`lsa, sikl o`z ishini boshlaydi va shartdagi qiymatlar sikl ichida o`zgartirib boriladi. While operatorining sintaksis quyidagicha**:**

```
while (shart) {
   // operatorlar bloki
}
```

Shart har doim mantiqiy qiymat qabul qiladi: rost(***true***) yoki yolg`on(***false***). Blok ichiga istalgancha operatorlar yozish mumkin, yoki umuman yozmaslik ham mumkin.

- «**While**» operatorida avval shart tekshirilib keyin amal bajariladi.
- «**While**» operatoridan raqamlar ketma-ketligi doimiy bo`lmagan hollarda ishlatiladi.
- «**While**» operatori takrorlanish davri har xil bo`lgan sikllar ustida amallar bajarishda qo`l keladi.

Yuqorida 1 dan 10 gacha bo`lgan sonlarni chiqarish dasturini while da bajaraylik:

```
public class WhileExample {
    public static void main(String[] args) {
        int i=1;
```

```
while(i<=10){
    out.print(i+",);
    i++;     }}}
```

Natija: 1,2,3,4,5,6,7,8,9,10

Sikl operatorlarida **break** yoki **continue** dan ham foydalanish mumkin. **«Break» operatori.** Bu operator bilan, «**switch**» tanlash operatorida tanishgan edik. U yerda bu operator, operatorlar ketma-ketligini tugatish uchun ishlatilgan edi. «**Break**» operatori dasturda, sikldan chiqish va «**shartsiz o`tish**» operatori kabi vazifalarni ham bajaradi.

Sikl shartga bog`liq holda operator yoki operatorlar guruhini ishlatib turadi. Qachonki shart yolg`on bo`lsa, sikl o`z ishini tugatadi va sikldan keyingi operatorlar ishlashni boshlaydi. «**Break**» operatori sikl shartiga qaramasdan, siklni tugatadi, hattoki, sikl sharti rost qiymat qabul qilgan bo`lsa ham.

```
class Test {
 public  static  void main(String args[]){
 for(int i=0; i<=10; i++){
 System.out.println(i);
 if (i==5) break;
 }
 System.out.println("Sikl tugatildidi");}}
```

Yuqoridagi misoldan ko`rinib turibdiki, sikl yordamida **10** ta qiymat ekranga chiqarilishi lozim, lekin bu jarayon oxirigacha bajarilmaydi, chunki «**break**» operatori mavjud. Bu operator «**if**» operatori bilan birga kelgan. Demak, shart qo`yilgan, agar «**i**» o`zgaruvchi qiymati «**5**» ga teng bo`lsa, «**break**» operatori ishga tushadi, ya`ni siklni tugatib dasturdan chiqib ketiladi.

«Continue» operatori. Bu operator, ishlayotgan sikl qadamini tashlab ketib, navbatdagi qiymat bilan siklni boshidan boshlab davom ettiradi. Sikl tanasida

ishlatilgan «**continue**» operatori, o`zidan keyin kelgan operatorlarni ishlatmaydi. Tub sonlarni chiqarib beruvchi dastur ko`rib chiqaylik.

```
public class MainClass {
 public static void main(String[] args) {
 int nValues = 40;
 OuterLoop: for (int i = 2; i <= nValues; i++) {
 for (int j = 2; j < i; j++) {
 if (i % j == 0) {
 continue OuterLoop;}
 }
System.out.print(i+",");
}} }
```

Natija:

2,3,5,7,11,13,17,19,23,29,31,37

Shartga ko`ra i ni j ga bo`lganimizda qoldiq 0 ga teng bo`lsa, **continue** amalni to`xtatib siklning boshiga qaytarib yuboradi.

Bob bo`yicha xulosalar

- **Paket** (Package) – bu sinflar, interfeyslar va kichik paketlar to`plamidir. Sub to`plami sinflarni, interfeyslarni va sinf ostilarini to`plashni o`z ichiga oladi. Java.lang. *; paketi jimlikda import qilinadi va bu paket odatiy paket sifatida tanilgan.
- **Sinf** (Class) – foydalanuvchi tomonidan aniqlangan ma`lumotlar turini ishlab chiqishda foydalaniladigan kalit so`z bo`lib, har bir java dasturi sinf tushunchasi bilan boshlanishi shart.

- "**ClassName**" – sinfning nomi sifatida ko`rib chiqiladigan java joriy o`zgaruvchining nomini ifodalaydi va javada har bir sinf nomi foydalanuvchi tomonidan belgilangan ma`lumot turi sifatida ko`rib chiqiladi.
- **Data memebers** – Ma`lumot turi (tip) sinfi nomiga asosan tanlab olinadigan namuna yoki statikni ifodalaydi.
- Foydalanuvchi tomonidan aniqlangan usullar operatsiyalarni bir martadan yoki har bir vaqtda bajarish uchun mo`ljallangan nusxani yoki statikni ifodalaydi.
- **main() –** Har bir java dasturi main() metodidan bajarishni boshlaydi. Shuning uchun main metodi dastur ishlatuvchisi sifatida tanilgan.
- Main metodini har bir java dasturchisi foydalana olishi kerak va shuning uchun kirish spetsifikatori **public** bo`lishi shart.

Java dasturlash tilida boshqaruv operatorlari **shart** operatori, **tanlash** operatori va **sikl** (**takrorlash**) operatorlaridan iborat.

Shart operatori – if (shart) {shart rost bo`lsa } else{ shart yolg`on bo`lsa}

Tanlash operatori – switch(a) {case 1:op1; break; case 2: op2; break; default: opn; break;}

Sikl operatorlari – for(int i=0;i<10;i++){} / while(shart){tanasi} /

foreach(int i: array){tanasi}

II bob. Sinf va shablonlar

2.1 Java dasturlash tilida sinf iyerarxiyasi

**Obyektga mo`ljallangan dasturlash tarixi**. Obyektga mo`ljallangan yondoshuv (OMYo) bir kunda o`ylab topilgan emas. Uning paydo bo`lishi dasturiy ta`minotning tabiiy rivojidagi navbatdagi pog`ona xolos. Vaqt o`tishi bilan qaysi uslublar ishlash uchun qulay-u, qaysinisi noqulay ekanini aniqlash oson bo`lib bordi. OMYo eng muvaffaqiyatli, vaqt sinovidan o`tgan uslublarni o`zida samarali mujassam etadi [7].

Dastlab dasturlash anchayin boshqotirma ixtiro bo`lib, u dasturchilarga dasturlarni kommutatsiya bloki orqali kompyuterning asosiy xotirasiga to`g`ridan-to`g`ri kiritish imkonini berdi. Dasturlar mashina tillarida ikkilik ko`rinishda yozilar edi. Dasturlarni mashina tilida yozishda tez-tez xatolarga yo`l qo`yilar edi, eng ustiga ularni tuzilmalashtirish imkoni bo`lmagani tufayli, kodni kuzatib borish amalda deyarli mumkin bo`lmagan hol edi. Bundan tashqari, mashina kodlaridagi dastur tushunish uchun g`oyat murakkab edi.

Vaqt o`tishi bilan kompyuterlar tobora kengroq qo`llanila boshlandi, hamda yuqoriroq darajadagi prosedura tillari paydo bo`ldi. Bularning dastlabkisi FORTRAN tili edi. Biroq obyektga mo`ljallangan yondoshuv rivojiga asosiy ta`sirni keyinroq paydo bo`lgan, masalan, ALGOL kabi prosedura tillari ko`rsatdi. Prosedura tillari dasturchiga axborotga ishlov berish dasturini pastroq darajadagi bir nechta proseduraga bo`lib tashlash imkonini beradi. Pastroq darajadagi bunday proseduralar dasturning umumiy tuzilmasini belgilab beradi. Ushbu proseduralarga izchil murojaatlar proseduralardan tashkil topgan dasturlarning bajarilishini boshqaradi [7].

Dasturlashning bu yangi paradigmasi mashina tilida dasturlash paradigmasiga nisbatan ancha ilg`or bo`lib, unga tuzilmalashtirishning asosiy vositasi bo`lgan proseduralar qo`shilgan edi. Maydaroq funksiyalarni nafaqat tushunish, balki sozlash ham osonroq kechadi. Biroq, boshqa tomondan, prosedurali dasturlash koddan takroran foydalanish imkonini cheklab qo`yadi. Buning ustiga dasturchilar tez-tez «makaron» dasturlar ham yozib turishganki, bu dasturlarni bajarish lidishdagi spagetti uyumini ajratishga o`xshab ketar edi. Va, nihoyat, shu narsa aniq bo`ldiki, prosedurali dasturlash usullari bilan dasturlarni ishlab chiqishda diqqatni ma`lumotlarga qaratishning o`zi muammolarni keltirib chiqarar ekan. Chunki ma`lumotlar va prosedura ajralgan, ma`lumotlar inkapsulatsiyalanmagan. Bu nimaga olib keladi? Shunga olib keladiki, har bir prosedura ma`lumotlarni nima qilish kerakligini va ular qayerda joylashganini bilmog`i lozim bo`ladi. Agar prosedura o`zini yomon tutsa-yu, ma`lumotlar ustidan noto`g`ri amallarni bajarsa, u ma`lumotlarni buzib qo`yishi mumkin. Har bir prosedura ma`lumotlarga kirish usullarini dasturlashi lozim bo`lganligi tufayli, ma`lumotlar taqdimotining o`zgarishi dasturning ushbu kirish amalga oshirilayotgan barcha o`rinlarining o`zgarishiga olib kelar edi. Shunday qilib, xatto eng kichik to`g`rilash ham butun dasturda qator o`zgarishlar sodir bo`lishiga olib kelar edi.

Modulli dasturlashda, masalan, Modula2 kabi tilda prosedurali dasturlashda topilgan ayrim kamchiliklarni bartaraf etishga urinib ko`rildi. Modulli dasturlash dasturni bir necha tarkibiy bo`laklarga, yoki, boshqacha qilib aytganda, modullarga bo`lib tashlaydi. Agar prosedurali dasturlash ma`lumotlar va prosedsralarni bo`lib tashlasa, modulli dasturlash, undan farqli o`laroq, ularni birlashtiradi. Modul ma`lumotlarning o`zidan hamda ma`lumotlarga ishlov beradigan proseduralardan iborat. Dasturning boshqa qismlariga moduldan foydalanish kerak bo`lib qolsa, ular modul interfeysiga murojaat etib qo`yaqoladi. Modullar barcha ichki axborotni dasturning boshqa qismlaridan yashiradi.

Biroq modulli dasturlash ham kamchiliklardan holi emas. Modullar kengaymas bo`ladi, bu degani kodga bevosita kirishsiz hamda uni to`g`ridan-to`g`ri o`zgartirmay turib modulni qadamba-qadam o`zgartirish mumkin emas. Bundan tashqari, bitta modulni ishlab chiqishda, uning funksiyalarini boshqasiga o`tkazmay (delegat qilmay) turib boshqasidan foydalanib bo`lmaydi. Yana garchi modulda turni belgilab bo`lsa-da, bir modul boshqasida belgilangan turdan foydalana olmaydi.

Modulli va proseduralı dasturlash tillarida tuzilmalashtirilgan va tuzilmalashtirilmagan ma`lumotlar o`z «tur»iga ega. Biroq turni kengaytirish usuli, agar «agregatlash» deb ataluvchi usul yordamida boshqa turlarni yaratishni hisobga olmaganda, mavjud emas.

Va, nihoyat, modulli dasturlash — bu yana proseduraga mo`ljallangan gibridli sxema bo`lib, unga amal qilishda dastur bir necha proseduralarga bo`linadi. Biroq endilikda proseduralar ishlov berilmagan ma`lumotlar ustida amallarni bajarmaydi, balki modullarni boshqaradi.

Obyektga mo`ljallangan dasturlash (OMD) modulli dasturlashdan keyingi mantiqiy pog`onani egallaydi, u modulga nasldan-naslga o`tishni va polimorfizmni qo`shadi. OMD dan foydalanr ekan, dasturchi dasturni bir qator oliy darajali obyektlarga bo`lish yo`li bilan tizimlashtiradi. Har bir obyekt hal qilinayotgan muammoning ma`lum bir tomonini modellashtiradi. OMD endilikda dasturni bajarish jarayonini boshqarish uchun dasturchi diqqatini proseduralarni ketma-ketlikda chaqirib olish ro`yxatini tuzib o`tirishga qaratmaydi. Buning o`rniga obyektlar o`zaro aloqada bo`ladi. OMYo yordamida ishlab chiqilgan dastur hal qilinayotgan muammoning amaldagi modeli bo`lib xizmat qiladi.

Dasturga obyektlar atamalari bilan ta`rif berish dasturiy ta`minotni ishlab chiqishning eng tushunarli usulidir. Obyektlar hamma narsani obyekt nima

qilayotgani nuqtai nazaridan idrok etishga, ya`ni uning hatti-xarakatlarini hayolan modellashtirishga majbur qiladi. Shu tufayli obyektga yondoshishda u dasturning bajarilishi jarayonida qanday ishlatiladi degan nuqtai nazardan biroz e`tiborni chalg`itish mumkin. Shunday qilib, dasturni yozish jarayonida haqiqiy dunyoning tabiiy atamalaridan foydalanish mumkin. Dasturni alohida proseduralar va ma`lumotlar shaklida (kompyuter dunyosi atamalarida) qurish o`rniga, obyektlardan iborat dastur qurish mumkin. Obyektlar otlar, fe`llar va sifatlar yorlamida haqiqiy dunyoni dasturda modellashtirishga imkon beradi. Joriy qilish (realizasiya) hatti-xarakatlar qanday bajarilayotganini belgilaydi. Dasturlash atamalarida joriy qilish — bu dasturiy kod.

Yechilayotgan masala atamalari bilan fikrlab, joriy qilishning mayda-chuyda detallarida o`ralashib qolish havfidan qochish mumkin. Albatta, ayrim oliy darajadagi obyektlar kompyuter bilan aloqa qilishda past darajadagi, mashinaga mo`ljallangan usullardan foydalanishi lozim. Biroq obyekt bu aloqani tizimning boshqa qismlaridan izolyasiya qiladi.

Obyekt dastur konsturksiyasi bo`lib, unda holat va hatti-xarakat inkapsulalangan bo`ladi. Obyekt holati bu ichki obyekt o`zgaruvchanlari qiymatlarining yig`indisidir.

Ichki o`zgaruvchan deb obyekt ichida saqlanadigan qiymatga aytiladi.

Mohiyat e`tibori bilan, obyekt bu sinfning ekzemplyari (nushalaridan biri)dir.

OMD, haqiqiy dunyo kabi, obyektlardan tashkil topadi. Obyektga mo`ljallangan sof dasturlash tilida, eng dastlabki, bazaviy, butun, mantiqiy turlardan tortib, to sinflarning murakkabroq nushalarigacha, barchasi obyekt hisoblanadi. Biroq obyektga mo`ljallangan tillarning hammasi ham bu darajada

chuqurlashib ketmagan. Ayrim tillarda (masalan, Java kabi) int va float ga o`xshash oddiy primitivlar obyekt sifatida olib qaralmaydi.

OMD obyektlari, haqiqiy olam obyektlari kabi, o`z xususiyatlari va xatti-harakatlari bo`yicha tasniflanadi.

Biologiyada itlar, mushuklar, fillar va odamlar sut emizuvchilarga kiradi. Bu turli xildagi jonivorlarni umumiy xususiyatlar birlashtirib turadi. Xuddi shunday, dasturiy ta`minot olamida ham obyektlar bitta yoki bir nechta sinflarga mansub bo`ladi.

Bitta sinfga mansub obyektlarga umumiy xususiyatlar xos bo`ladi. Boshqacha qilib aytganda, sinf obyektni tavsiflaydigan xususiyatlar va xulq-atvorlarni, shuningdek, va bu eng muhimidir, obyekt javob beradigan xabarlarni belgilab beradi. Biron bir obyekt boshqa obyektning xulq-atvoriga ta`sir ko`rsatgan vaqtda, u bu ta`sirni bevosita ko`rsatmaydi, balki undan qandaydir bir qo`shimcha axborotdan foydalangan holda o`zini-o`zi o`zgartirishni iltimos qiladi. Odatda bu «xabarni jo`natish» deb ataladi.

Sinf umumiy xususiyatlar va xulq-atvorga ega bo`lgan obyektlarni birlashtiradi. Bitta sinfga mansub obyektlar bir xil xususiyatlarga ega bo`lib, bir xil xatti-harakat namoyon etadi.

Sinflar shablon (qolip)ga o`xshaydi: ular obyektlarning ekzemplyarlari (nushalari)ni tayyorlash uchun qo`llanadi.

Belgilar — sinfning tashqaridan ko`rinib turgan xususiyatlari.

Obyekt ichki o`zgaruvchiga bevosita kirishni taqdim etganda yoki usul yordamida qiymatni qaytargandagina, o`z belgilarini namoyon qilishi mumkin.

Xulq-atvor — xabarga yoki holatning o`zgarishiga javoban obyekt tomnidan bajariladigan xatti-harakatlar. U obyekt nima qilayotganini bildiradi.

Bir obyekt ikkinchi obyekt ustida xatti-harakatlar bajarib, uning xulq-atvoriga ta`sir ko`rsatishi mumkin. «Xatti-harakat» atamasi o`rniga «usulni chaqiirish», «funksiyani chaqirish» yoki «xabarni uzatish» atamalari qo`llanadi. Muhimi bu atamalarning qaysi biri qo`llanayotganida emas, albatta, muhimi bu xatti-harakatlar obyekt xulq-atvorini namoyon qilishga da`vat etishidadir.

Obyektlar o`rtasida aloqa obyektga mo`ljallangan dasturlashning muhim tarkibiy qismidir. Obyektlar o`zaro aloqasining ikkita asosiy usuli mavjuddir.

Birinchi usul: obyektlar biri ikkinchisidan mustaqil ravishda mavjud bo`ladi. Agar alohida obyektlarga o`zaro aloqa kerak bo`lib qolsa, ular bir-birlariga xabar jo`natadi.

Obyektlar bir-birlari bilan xabarlar yordamida aloqa qiladi. Xabar olgan obyekt ma`lum xatti-harakatlarni bajaradi.

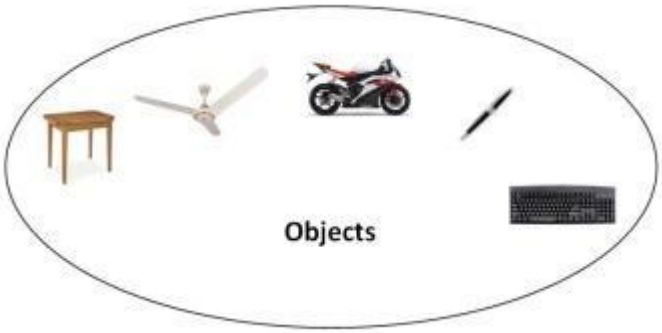
Xabar uzatish bu obyekt holatini o`zgartirish maqsadida uslubni chaqirib olish yoki xulq-atvor modellaridan birini qo`llashning o`zginasidir.

Ikkinchi usul: obyekt tarkibida boshqa obyektlar bo`lishi mumkin. Xuddi OMD da bo`lganidek, dastur obyektlardan tashkil topganidek, obyektlar ham, o`z navbatida, agregasiya yordamida boshqa obyektlardan jamlanishi mumkin. Ushbu obyektlarning har bittasida uslub va belgilarga ega bo`lgan interfeys mavjud bo`ladi.

Xabar — obyektga mo`ljallangan yondoshuvning muhim tushinchasi. Xabarlar mexanizmi tufayli obyektlar o`z mustaqilligini saqlab qolishi mumkin. Boshqa biron obyektga xabar jo`natayotgan obyekt uchun xabar olgan obyekt

talabdagi xatti-harakatni qanday bajarishi unchalik muhim emas. Unga xatti-harakat bajarilganligining o`zi muhimdir [18].

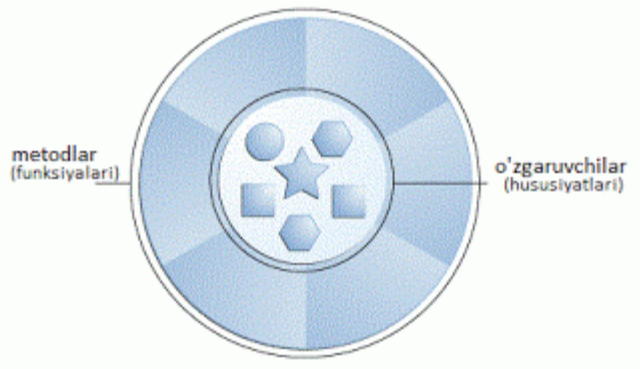
Obyekt – Obyektga yo`naltirilgan dasturlash(OYD) dasturlash texnologiyasining eng asosiy kalit tushunchasidir. Atrofga qarang, haqiqiy hayotdagi bir necha obyektlarni ko`rishingiz mumkin: stol, uy, qalam , motosikil , televizor va h.k. (16-rasm).



16-rasm. Obyektlar.

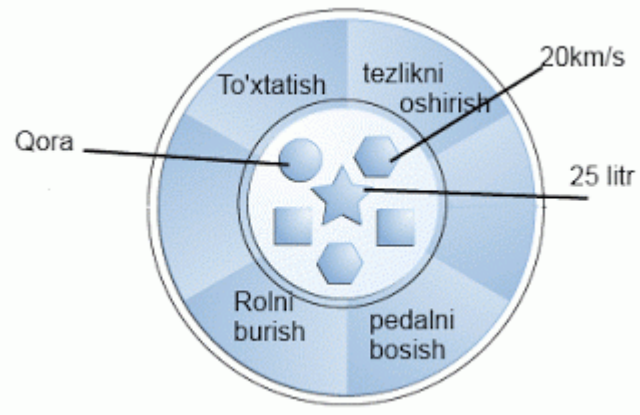
Ularning barchasining albatta xususiyatlari va bajaradigan vazifalari (funksiyalari) bor. Masalan, Mushuk hususiyatlari: rangi, qorni to`qligi, yoshi, jinsi; funksiyalari: ovqat yeyishi, miyovlashi, yurishi, sichqon tutishi. Mashina xususiyatlari: tezligi, rangi, nomi, narxi; funksiyalari: yurishi, to`xtashi, oyna artgichlarining ishlashi, eshiklarning ochilib yopilishi va h.k. Bu kabi hayotiy misollarning hususiyatlari va funksiyalarini aniqlash OYD nuqtai nazaridan fikrlashning eng zo`r ko`rinishidir.

Bir daqiqaga to`xtang va hozirda atrofingizdagi biror narsalarni tahlil qiling. Har bir obyekt uchun o`zingizdan so`rang: "Bu obyektning qanday hususiyatlari bor?", "Qanday vazifalarni bajaradi?" kabi. Va kuzatish natijalaringizni yozib oling, sezgan bo`lsangiz tuziladigan ro`yxat obyektning murakkabligiga qarab ko`payib boradi. Kompyuter indikatorining 2 ta *xususiyati* bor o`chiq va yoniq; *funksiyalari* esa yonish va o`chish (17-rasm). Bu barcha kuzatishlar OYD dunyosiga o`tkazish mumkin.



17-rasm. Obyekt ustida bajariladigan amallar

Obyekt ham haqiqiy hayotdagi obyektlarga o`xshash: ular ham qandaydir xususiyatlar va bajaradigan funksiyalardan iborat bo`ladi. Obyektning xususiyatlari har xil dasturiy *o`zgaruvchilardan* iborat bo`ladi va ularning o`zgartirish uchun qandaydir *funksiyalar* bajariladi. Bunday *funksiyalar* bilan o`zgaruvchilarning holatini berkitish mumkin ya`ni aynan o`sha o`zgaruvchini tashqaridan o`zgartirish uchun albatta maxsus funksiyadan foydalanish kerak bo`ladi. Bu jarayon "**Enkapsulatsiya**" deb atalib, OYDning eng muxim tushunchalaridan biridir. Mashinani tasavvur qiling (18-rasm).



18-rasm. Obyektning parametrlari

Uni dasturlash obyekti sifatida modellashtiramiz:

Uning o`zgaruvchilari( hozirgi tezligi, qolgan benzini, va h.k) va uning funksiyalari(to`xtatish, tezlikni oshirish, rolni burish va h.k.). Bu yerda uning bakidagi benzini yurishi tufayli kamayib boradi demak uning qiymatining o`zgarishi 0 dan bakning sig`imigacha bo`ladi, yoki uning tezligi ham shu kabi

aynan qaysidir funksiyalarning amalga oshirilishi orqali u ham 0 dan maksimal tezligigacha o`zgarishi mumkin. Bulardan tashqari mashinaning ba`zi hususiyatlari borki ular o`zgarmasligi mumkin, masalan rangi.

Demak, ko`rinib turipdiki mashina ham o`z navbatida bir necha mayda obyeklardan iborat bo`ladi. Va albatta ularni kodda yozganda ham alohida obyekt sifatida ifodalash kerak bu orqali nimalarga erishish mumkin:

Qismlilik. Har bir obyektga tegishli bo`lgan kodlar alohida-alohida, boshqa obyeklarga bog`liq bo`lmagan holda boshqarish imkoniyatiga ega bo`lamiz. Bu hammasi emas, tasavvur qiling mashina obyektini ifodalovchi kodni bo`lmasdan faqat bitta faylda ifodaladik; bu esa murakkabligiga qarab yuzlab xatto minglab qatorli kod bo`lishi mumkin. Undan biror narsani topib-o`zgartrish ancha mashaqqat bo`ladi [12, p.62].

Qayta foydalanish. Yana boshqa foydali tarafi biz bo`laklagan mashinaning detallarini boshqa obyektlarda ham ishlatishimiz mumkin. Masalan, 2 xil mashina ularning aynan bir xil qismlari bor, ana o`shalar uchun ikki marta alohida kod yozmasdan, bitta yozganimizni qayta ishlatishimiz mumkin [12, p.63].

Uzilib-ulanuvchanligi. Buni tushunish uchun yuqoridagi misoldan foydalanamiz, aytaylik, mashinaning biror qismi ishlamayapti, xo`sh nima qilinadi? O`sha qismni ishlab turgan boshqa ehtiyot qismga almashtiramiz, yoki tuzatamiz. Mashinaning biror vinti buzilsa uni boshqa ishlab turgani bilan almashtirasiz yoki tuzatamiz lekin mashinani butunligicha yahlit almashtirmaymiz [12, p.63].

OYD ning asosiy tushunchalari. *Obyekga yo`naltirilgan dasturlash* yoki *OYD* – haqiqiy hayotiylikka asoslangan dasturlash usulidir. Yana protsedurali dasturlash tillari (masalan, Paskal, Basic, Fortan) ham mavjud. OYD ning undan asosiy farqi shundaki, OYD asosan obyektlarga asoslangan holda ishlasa, *protsedurali dasturlash tillari* esa asosan funksiyalarga asoslangan bo`ladi

ya`ni bu usuldagi dasturlashda har bitta buyruqlar qadamma-qadam bajarilib boriladi masalan: faylni och, raqamni o`qi, 4 ga ko`paytir va ekranga chiqar.

OYD ni tashkil etuvchilari quydagilar:

- Object – Obyekt
- Class – Sinf
- Inheritance – Meros olish
- Polymorphism – Ko`p formalik
- Abstraction – Mavhumlik
- Encapsulation – Enkapsulyatsiya (Kapsula ichiga joylamoq)

Object – Obyekt klass turidagi o`zgaruvchi. Obyekt bu klass bilan farqli tushuncha hisoblanadi. Objekt biz yozgan klassimizdagi har xil qoidalarga bo`ysunadigan ma`lumot bo`lib, u tezkor hotirada saqlanadi, klass esa qattiq diskda saqlanadi. Har bir yasalgan Obyect tezkor xotiraning ma`lum bir xonachalariga joylashadi. Hayotiy bir misol, masalan, ko`p qavatli binoni tezkor xotira deb qarasak. Unda istiqomat qiluvchi insonlar esa unda saqlanuvchi obyektlar bo`lib. Agar biz Insonning hususiyatlari, bajaradigan ishlari va hokazo xususiyatlari haqidagi bilimlarni qog`ozga tushursak bu qoralamani klass deb qaralishi mumkin garchi u texnik usulda yozilmagan bo`lsa ham. Biz ana o`sha qoralamani klassimizda kompyuter tushunadigan tilga keltiramiz [12, p.62].

Class – OYD ning marzkazi hisoblanadi va u har xil kodlar, ma`lumotlar va shu ma`lumotlar qay tarzda o`zgarishini ifodalovchi xususiyatlar saqlanadi. Boshqacharoq qilib aytadigan bo`lsak hayotiy obyektlarning qanday faoliyat yuritishi, nimalardan iborat ekanligi, qanday xususiyatlarga ega ekanligini tavsiflovchi kichik bir hujjat sifatida qarash ham mumkin. Javada hamma narsa Klass ichida sodir bo`ladi. Klass o`z ichiga o`zgaruvchilar va metodlar(funksiyalar) va qiymati o`zgarmaydigan konstantalarni oladi. Yana shuni ham ta`kidlash kerakki, har bitta klass bitta o`zgaruvchi turi bo`lib ham xizmat

qiladi. Huddi Integer, String yoki boshqa turlar kabi har bir klass ham ma`lum bir tur sifatida qaralishi mumkin [12, p.62].

Javada klasslar quydagilardan tashkil topadi:

1. data member – o`zgaruvchilar
2. method – funksiya
3. constructor – kontruktor (obyekt dastlabki holatida ishlaydi)
4. block – blok ({} qavslar orasi blok hisoblanadi)
5. class and interface – klass va interfeys

**O`zgaruvchilar (data member).** Obyektlarimizni o`zgaruvchi sifatida ham qarashimiz mumkin bunda uning qiymatlari o`zgarishi mumkin. Masalan Inson klassimizda insonning yurish tezligi o`zgaradi bunda uning tezligi qandaydir sonlarda o`zgaradi masalan **yur()** metodidan **yugur()** metodiga o`tganda yoki **to`xta()** metodiga o`tganda uning tezligi o`zgaradi. Shu va shunga o`xshash obyektlar o`zgaruvchilar deb yuritiladi.

```
public class Inson{
    int tezlik=0;
    public void yur(){
    tezlik=5;
    }
 public void yugur(){
    tezlik=15;
    }
 public void yur(){
    tezlik=0;
   }
}
```

"instance variable" – o`zgaruvchi namuna olish kompilyatsiya vaqtida xotiradan joy olmaydi. O`zgaruvchilarimizni obyekt sifatida qaraganimizda runtime (dastur ish payti) paytida tezkor xotiradan joy oladi.

Funksiya (Method) lar. Obyektimiz nimalar qila olishini izohlaydi, masalan, inson haqida yozgan qoralamamizda insonni yura olishi va boshqa harakatlarini keltirganmiz. Aynan ana o`sha qila oladigan ishlarini Metodlar orqali ifodalaymiz. Masalan quyida **yur()** metodi keltirilgan.

Funksiyalarning imkoniyatlari

1. Kodni qayta ishlash
2. Kodni optimallashtirish (muvofiqlashtirish)

Obyekt va klass. Bu misolda **student** klassidan ikkita obyekt yaratildi va insertRecord funksiyasi orqali boshlang`ich qiymatlarni o`zlashtirildi. *displayInformation* funksiyasidan foydalanib ekranda qimatlarni chop etiladi [12, p.62].

```
class Student{
    int rollno;
    String name;
    void insertRecord(int r, String n){  //method
       rollno=r;
       name=n;
       }

    Void displayInformation(){System.out.println(rollno+” ”+name);}
//method
    public static void main(String args[]){
     Student s1=new Student();
     Student s2=new Student();
```

```
        s1.insertRecord(111,"Karan");
        s2.insertRecord(222,"Aryan");
        s1.displayInformation();
        s2.displayInformation();
       }
}
```

Ekranda :

111 Karan

222 Aryan

**Merosxo`rlik (Inheritance).** Ma`lum obyekt asosida boshqa obyektni yaratish jarayoniga aytiladi. Bunda obyektning barcha xususiyatlarini meros qilib oladi ya`ni **private(shaxsiy)** bo`lmagan o`zgaruvchilari funksiyalari konstantalarini bemalol foydalanish. *Meros olish*dan dastur ishchi vaqtida *Ko`p formalik*dan foydalaniladi.[12, p.4]

**Ko`p formalik (Polymorphism).** OYDning uchinchi ustuni. Yuqorida berilgan ustunlar bilan doim birga yuradi. Ma`lum obyekt asosida boshqa obyektni yaratish jarayoniga aytiladi. Bir klassning boshqa klassdan meros olishi yordamida amalga oshiriladi. Meros olingan obyekt ota obyektdagi xususiyatlarni tanlovga ko`ra meros oladi. Masalan, avtoulov bu ota obyekt. Bu obyekt yordamida yengil mashina, yuk mashinasi, poyga mashinasi kabi boshqa obyektlarni yaratib olishimiz mumkin. Ota klassda bo`lgan 4 g`ildirak farzand klasslarda ham mavjud bo`ladi. Ya`ni poyga mashinasi, avtoulovdan g`ildiraklarni meros oladi.[12, p.62]

19-rasm. Polimorfizm

Mavhumlik – Abtraksiya (Abstraction). Obyekt bu real jism. Abstraksiya esa ushbu jismni reallikdan uzoq deb tassavur qilib, u haqida fikr yuritishga

aytiladi. Abstraksiyaga fikrlar to`plami sifatida ham qarasa bo`ladi. Misol uchun telefon jiringlasa, telefonga qo`ng`iroq bo`layotganini bilamiz lekin uning ichida nima jarayon bo`layotganini bilmaymiz [12, p.62].

Inkapsulyatsiya (Encapsulation). Ma`lumotlar va funksiyalarni bir komponent ichiga yig`ishga aytiladi. Buning uchun klasslardan foydalaniladi. Enkapsulatsiya tanlov asosida klassning ba`zi xususiyatlarini foydalanuvchidan yashirish imkonini beradi. Bunga misol qilib, ustidan maxsus modda bilan o`ralgan dorilarni keltirsak bo`ladi [12, p.62] (19-rasm).

2.2 Java dasturlash tilida sinf shablonlarini ishlatish

Java dastrulash tilida sinf shablonlaridan keng qo`llaniladi. Ular yordamida kerakli sinflar oson va tez yaratiladi.

Sinf shablonlari. Dasturiy muhandislik, dizayn shablonlar, dasturiy ta'minot dizayni uchun keng tarqalgan bo'lib, qolgan muammolarga umumiy bir yechimdir. Sinf shablonlari to'g'ridan-to'g'ri kodga aylantirilishi mumkin bo'lgan tugallanmagan dizayn emas. Ko'p turli vaziyatlarda ishlatilishi mumkin bo'lgan muammoni qanday hal qilish bo'yicha tavsif yoki shablondir [21].

Sinf shablonlarini qo'llash. Dizayn shablonlari testlangan, tasdiqlangan rivojlanish paradigmalarini ta'minlab, rivojlanish jarayonini tezlashtirishi mumkin. Samarali dasturiy ta'minot dizayni, keyinchalik amalga oshirishga qadar ko'rinmasligi mumkin bo'lgan masalalarni hisobga olishni talab qiladi. Dizayn shablonlarini qayta ishlatish katta muammolarga olib kelishi mumkin bo'lgan nozik masalalarni oldini olishga yordam beradi va modellar bilan tanish bo'lgan kodlovchi va mimarlar uchun kodni okunabilirliğini yaxshilaydi [21].

dizayn shablonlarini yaratish. Ushbu dizayn shablonlarida sinfga tegishli hamma narsa qamrab olinadi. Ushbu shablon, shuningdek, sinf yaratilish modellari

va obyekt yaratish modellariga bo'linishi mumkin. Sinf yaratish shablonlari merosni eskirish jarayonida samarali ishlatish bilan birga, obyektni yaratish shablonlari ishni bajarish uchun delegatsiyadan samarali foydalanishadi.

Java dasturlash tilida shablonlarni ishlatishning modeli.

1. Bir algoritmning skeletining bazaviy “shablon” usulida standartlashtirilishi
2. Individual qadamlarning umumiy qo'llanilishi asosiy sinfda aniqlanadi
3. O'z-o'ziga xos ilovalarni talab qiladigan qadamlar, asosiy sinfda "to'ldiruvchilar" dir
4. Ishlatadigan sinflar ayrim metodlarni bekor qilishi mumkin
5. ishlatadigan sinflar amalga oshirilgan metodlarni bekor qilishi mumkin
6. Tarkibiy sinflar bekor qilishi va asosiy sinf maetodlarini "qayta chaqirishi" mumkin.

Quyida sinf shablonining ishlatilishiga misol ko`ramiz:

```
abstract class Generalization {
    // 1. Bir algoritmning skeletining bazaviy “shablon” usulida standartlashtirilishi
    void findSolution() {
        stepOne();
        stepTwo();
        stepThr();
        stepFor();
    }
    // 2. Individual qadamlarning umumiy qo'llanilishi asosiy sinfda aniqlanadi
```

```
    private void stepOne() {
        System.out.println("Generalization.stepOne");
    }
    // 3. O'z-o'ziga xos ilovalarni talab qiladigan qadamlar, asosiy sinfda "to'ldiruvchilar" dir
    abstract void stepTwo();
    abstract void stepThr();
    void stepFor() {
        System.out.println( "Generalization.stepFor" );
    }
}
abstract class Specialization extends Generalization {
    // 4. Ishlatadigan sinflar ayrim metodlarni bekor qilishi mumkin
    // 1. Bir algoritmning skeletining bazaviy “shablon” usulida standartlashtirilishi
    protected void stepThr() {
        step3_1();
        step3_2();
        step3_3();
    }
    // 2. Individual qadamlarning umumiy qo'llanilishi asosiy sinfda aniqlanadi
    private void step3_1() {
        System.out.println("Specialization.step3_1");
    }
    // 3. O'z-o'ziga xos ilovalarni talab qiladigan qadamlar, asosiy sinfda "to'ldiruvchilar" dir
    abstract protected void step3_2();
    private void step3_3() {
        System.out.println("Specialization.step3_3");
```

```
    }
}
class Realization extends Specialization {
    // 4. Ishlatadigan sinflar ayrim metodlarni bekor qilishi mumkin
    protected void stepTwo() {
        System.out.println("Realization.stepTwo");
    }
    protected void step3_2() {
        System.out.println( "Realization.step3_2");    }
    // 5. ishlatadigan sinflar amalga oshirilgan metodlarni bekor qilishi mumkin
    // 6. Tarkibiy sinflar bekor qilishi va asosiy sinf maetodlarini "qayta chaqirishi" mumkin
    protected void stepFor() {
        System.out.println("Realization.stepFor");
        super.stepFor();
    }
}
 public class TemplateMethodDemo {
    public static void main(String[] args) {
        Generalization algorithm = new Realization();
        algorithm.findSolution();
    }
}
```

Chiquvchi natija

Generalization.stepOne

Realization.stepTwo

Specialization.step3_1

Realization.step3_2

Specialization.step3_3

Realization.stepFor

Generalization.stepFor

Bob bo`yicha xulosalar

Obyekt – Obyektga yo`naltirilgan dasturlash(OYD) dasturlash texnologiyasining eng asosiy kalit tushunchasidir. Atrofga qarang, haqiqiy hayotdagi bir necha obyektlarni ko`rishingiz mumkin: stol, uy, qalam , motosikil , televizor va h.k.

Class – OYD ning marzkazi hisoblanadi va u har xil kodlar, ma`lumotlar va shu ma`lumotlar qay tarzda o`zgarishini ifodalovchi xususiyatlar saqlanadi. Boshqacharoq qilib aytadigan bo`lsak hayotiy obyektlarning qanday faoliyat yuritishi, nimalardan iborat ekanligi, qanday xususiyatlarga ega ekanligini tavsiflovchi kichik bir hujjat sifatida qarash ham mumkin. Javada hamma narsa Klass ichida sodir bo`ladi. Klass o`z ichiga o`zgaruvchilar va metodlar(funksiyalar) va qiymati o`zgarmaydigan konstantalarni oladi. Yana shuni ham ta`kidlash kerakki, har bitta klass bitta o`zgaruvchi turi bo`lib ham xizmat qiladi. Huddi Integer, String yoki boshqa turlar kabi har bir klass ham ma`lum bir tur sifatida qaralishi mumkin.

Bobning ikkinchi fasli *“Java dasturlash tilida sinf shablonlarini ishlatish”* deb nomlanib, unda qanday qilib sinf shablonlarini ishlatish, uning afzalliklari ko`rsatilgan.

Sinf shablonlari. Dasturiy muhandislik, dizayn shablonlar, dasturiy ta'minot dizayni uchun keng tarqalgan bo'lib, qolgan muammolarga umumiy bir yechimdir. Sinf shablonlari to'g'ridan-to'g'ri kodga aylantirilishi mumkin bo'lgan

tugallanmagan dizayn emas. Ko'p turli vaziyatlarda ishlatilishi mumkin bo'lgan muammoni qanday hal qilish bo'yicha tavsif yoki shablondir.

Sinf shablonlarini qo'llash. Dizayn shablonlari testlangan, tasdiqlangan rivojlanish paradigmalarini ta'minlab, rivojlanish jarayonini tezlashtirishi mumkin. Samarali dasturiy ta'minot dizayni, keyinchalik amalga oshirishga qadar ko'rinmasligi mumkin bo'lgan masalalarni hisobga olishni talab qiladi. Dizayn shablonlarini qayta ishlatish katta muammolarga olib kelishi mumkin bo'lgan nozik masalalarni oldini olishga yordam beradi.

III bob. Java dasturlash tilida Aspektga mo`ljallangan dasturlash

3.1 AOP ning maqsadi

Ma’lumki, katta loyihalarni bajarishda qo`yilgan masala qismlarga ajratilib, har bir qism uchun sinflar yaratiladi. Java dasturlash tilida bu sinflar orqali amalga oshiriladi. Har bir sinf o`ziga tegishli bo`lgan masalalarni bajarishga qaratiladi. Agar biz qaysidir sinfga o`zgartirish kiritmoqchi bo`lsak, bunda java dasturlash tilida javada yozilgan kodga o`zgartish qilishimizga to`g`ri keladi va bu dastur qaytadan kompilatsiya qilinib sinfga o`zgartirishlar kiritiladi. Bu yerda sinflar bir-biriga bog`liq bo`lishi mumkin va bu o`zgarish boshqa sinflarga ham ta’sir qilishi mumkin. Shunday muammolar tug`ilib qoladiki, unda biz dastur kodiga tegmasdan turib faqat sinfni o`zgartirishimiz kerak bo`ladi. Bunday jarayonni Java dasturlash tilida Aspectga mo`ljallangan dasturlash orqali amalga oshirishimiz mumkin. Bunda biz Javada yozilgan Aspect orqali Javada tuzilgan ixtiyoriy dastur kodiga o`zgartirish qilmay turib, sinfni o`zgartirish orqali muammoni hal qilamiz. Bu juda katta loyihalarni amalga oshirishda haqiqatdan ham juda qulay hisoblanadi. Aspect deganda biz, nuqtiy-nazar orqali qarashni tushunamiz. Ushbu tatqiqotning dolzarbligi Java dasturlash tilida Aspectga mo`ljallangan dasturlash orqali yuqorida keltirilgan muammolarni hal qilish bilan belgilanadi [22].

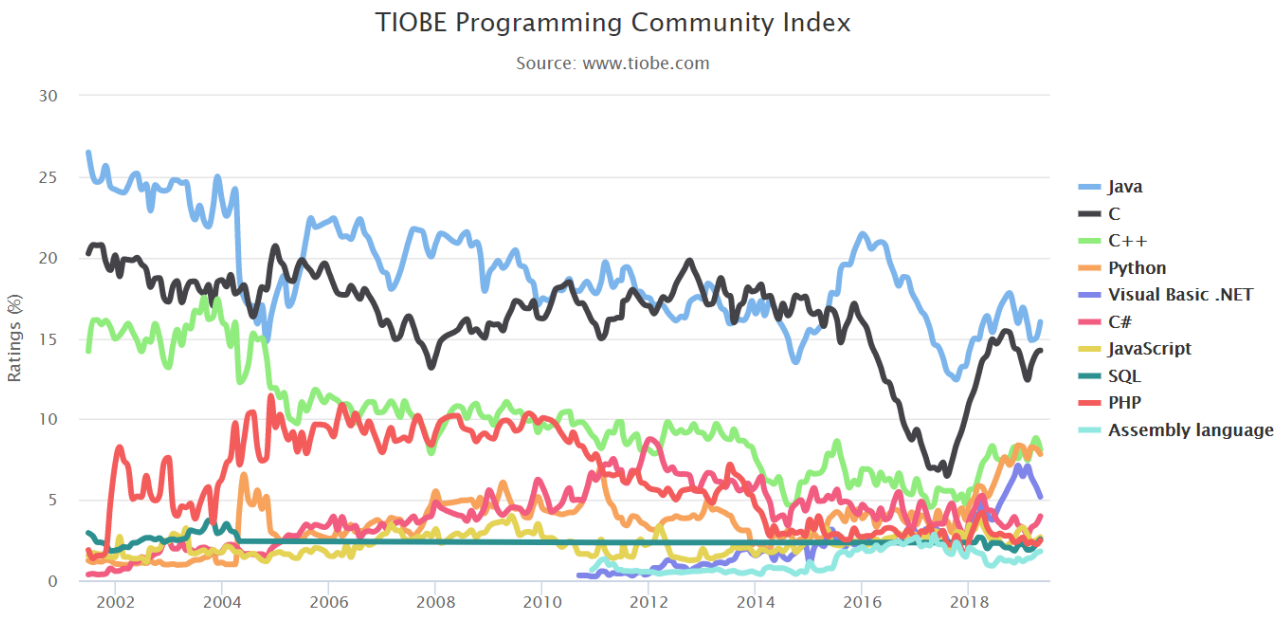
Eng keng tarqalgan dasturlash tillarining ko'pchiligi (C ++, Object Pascal, Java, Python, va hokazo) ko'p paradigma bo'lib, odatda, majburiy, protsessual dasturlash bilan birgalikda katta yoki kichik darajada obyektga asoslangan dasturlashni qo'llab-quvvatlaydi. Java, C ++, C #, Python, PHP, JavaScript, Ruby, Perl, ob'ekt Paskal, Objective-C, Dart, Swift, Scala, Common Lisp va Smalltalk kabi muhim ob'ektga yo'naltirilgan tillar mavjud.

Quyida Aspectni qo`llab-quvvatlovchi dasturlash tillari bilan tanishamiz. Bular:

- AspectJ

- HyperJ
- AspectC++
- AspectC#
- Caesar
- CompositionFilters
- AspectWerkz
- JBoss-AOP

kabi tillardir. TIOBE Programming Community Index ning ko`rsatkichlari bo`yicha dasturlash tillarining reytingi keltirilgan (2002-2018 yillar) (20-rasm).



20-rasm. TIOBE dasturlash tillari indeksi

Reytingdan ko`rinadiki, Dastuchilar asosan Java va C dasturlash tillaridan keng foydalanishadi. Bunga sabab qulay platforma va moslashuvchanlikdir. Bundan tashqari java dasturlash tili obyektga mo`ljallangan til ekanidadir.

Reyting natijalaridan yana shu ma’lum bo`ladiki, 2002-2004 yillar oraliqlarida java dasturlash tili ko`rsatkichlari pasayib brogan, aynan shu yillarda java kodlarini AOP orqali endi-endi amalga oshirish ishlari boshlangan.

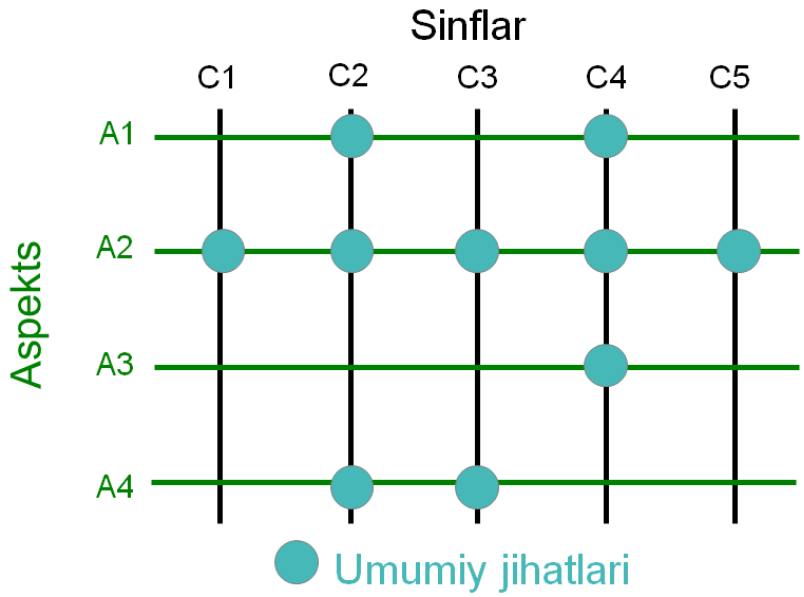
3.2 AOP ning ijobiy va salbiy tomonlari

AOP ning OOP (Object oriented programming) dan farqli tomonlari:

Paradigma	**Abstraktsiya**	**Modullashtirish**
Procedural programming	**kutubxonalar, ma’lumotlar tipi**	**Modul**
Object-oriented programming (OOP)	**ma’lumotlar tipi**	**Class / object**
Aspect-oriented programming (AOP)	**Crosscutting conserns (o'zaro bog'liq muammolar)**	**Aspect**

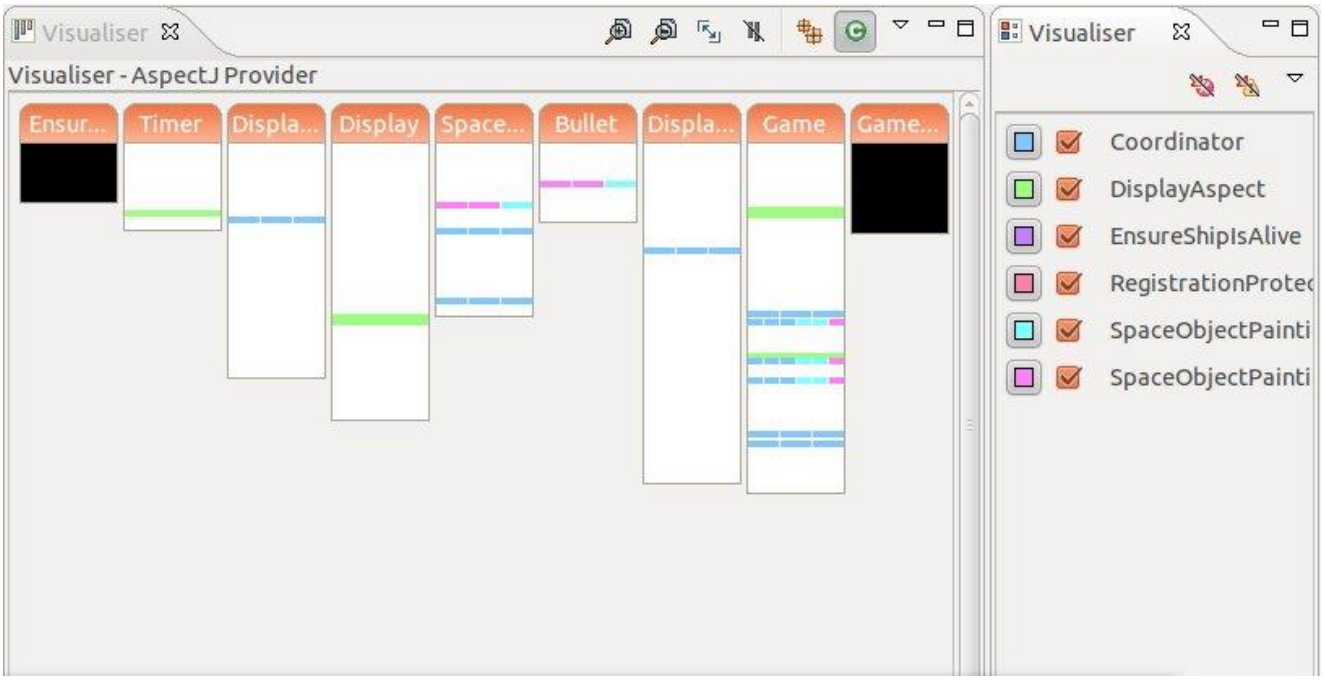
3-jadval. AOP, OOP va Prosedurali dasturlashning farqli jihatlari

AOP va OOP o`xshash tomonlari quyidagi 4-jadvalda keltirilgan:



4-jadval. AOP va OOP ning umumiy jihatlari

Eclipse da AJDT yordamida AOP quyidagicha tasvirlanadi (21-rasm). Bunda Visualiser – AspectJ Provider orqali aspectning maydonlari visual tarzda tasvirlanadi.



21-rasm. Eclipse da AJDT yordamida aspektlarni tasvirlash

AOP ning sintaksisi quyidagicha ifodalanadi:

[privileged] [*Modifiers*] aspect *Id*

[extends *Type*] [implements *TypeList*] [*PerClause*] {

Body

}

Agar privileged ishlatilgan bo`lsa, ushbu yo`nalish kodi har qanday ruxsat cheklovi e'tibordan chetda qoldiriladi: u private, protected yoki package-protected deb e'lon qilingan narsalarga ega bo`ladi.

Pointcut – bu AOP ning konstruktori hisoblanib, uningning 3 ta asosiy metodlari bor. bular:

before – birlashish nuqtasidan oldin amalga oshiriladi

after – birlashish nuqtasidan keyin amalga oshiriladi

around – birlashish nuqtasida amalga oshiriladi

Before metodi quyidagicha sintaksisga ega:

before( ): constructorCall( ) {

System.out.println("Constructor **call about to start");**

}

After metodi quyidagicha sintaksisga ega:

after( ): constructorCall( ) {

System.out.println("Constructor **call has now finished");**

}

Around metodi quyidagicha sintaksisga ega:

void around( ): setBalanceMethodExecution( ) {

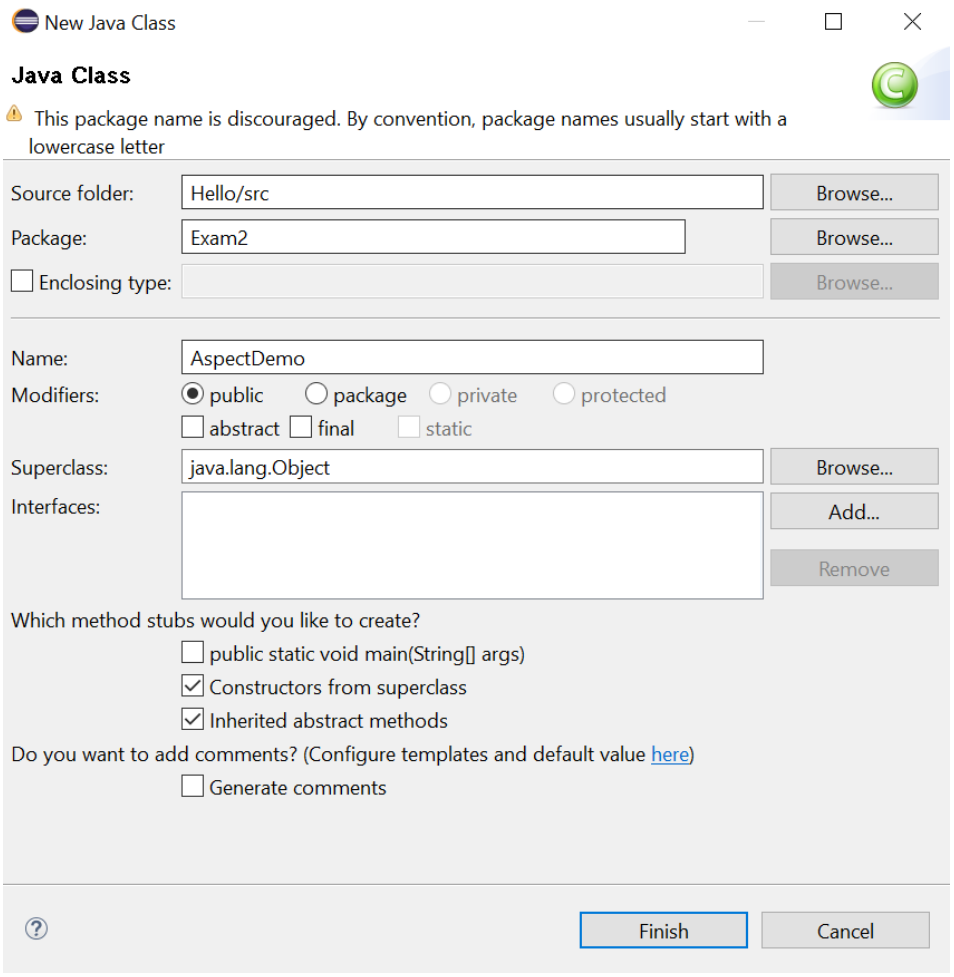
System.out.println(“I won’t let you change your balance);

}

3.3 AOP bo`yicha tavsiyalar

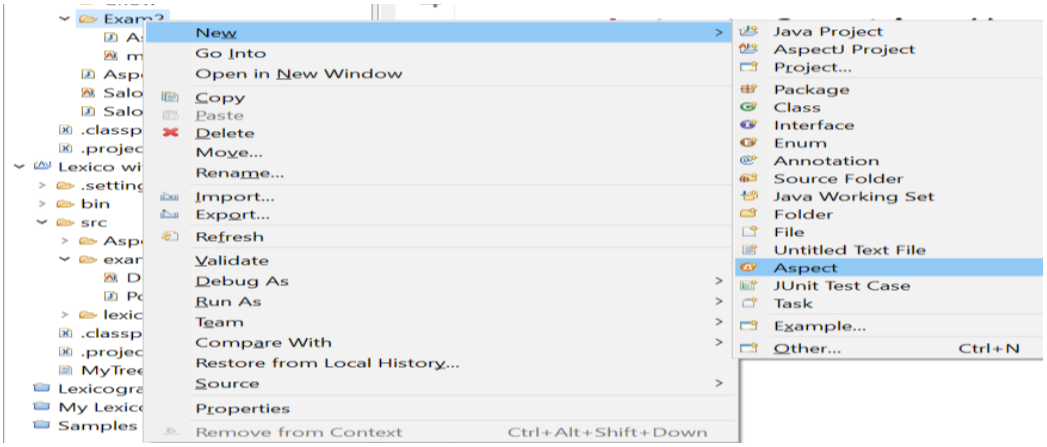
AspectJ ni amalga oshirish uchun Eclipse ning **Indigo Servise Release 1** versiyasini o`rnatamiz. Unga mos ajdt_2.2.2_for_eclipse_3.7 ni yuklab o`rnatiladi.

Dastlab class (sinf) yaratamiz. Masalan, class nomini “AspectDemo” deb nomladik. Buning uchun quyidagi amallarni 22-rasmda ko`rsatilgan tartibda bajaramiz.

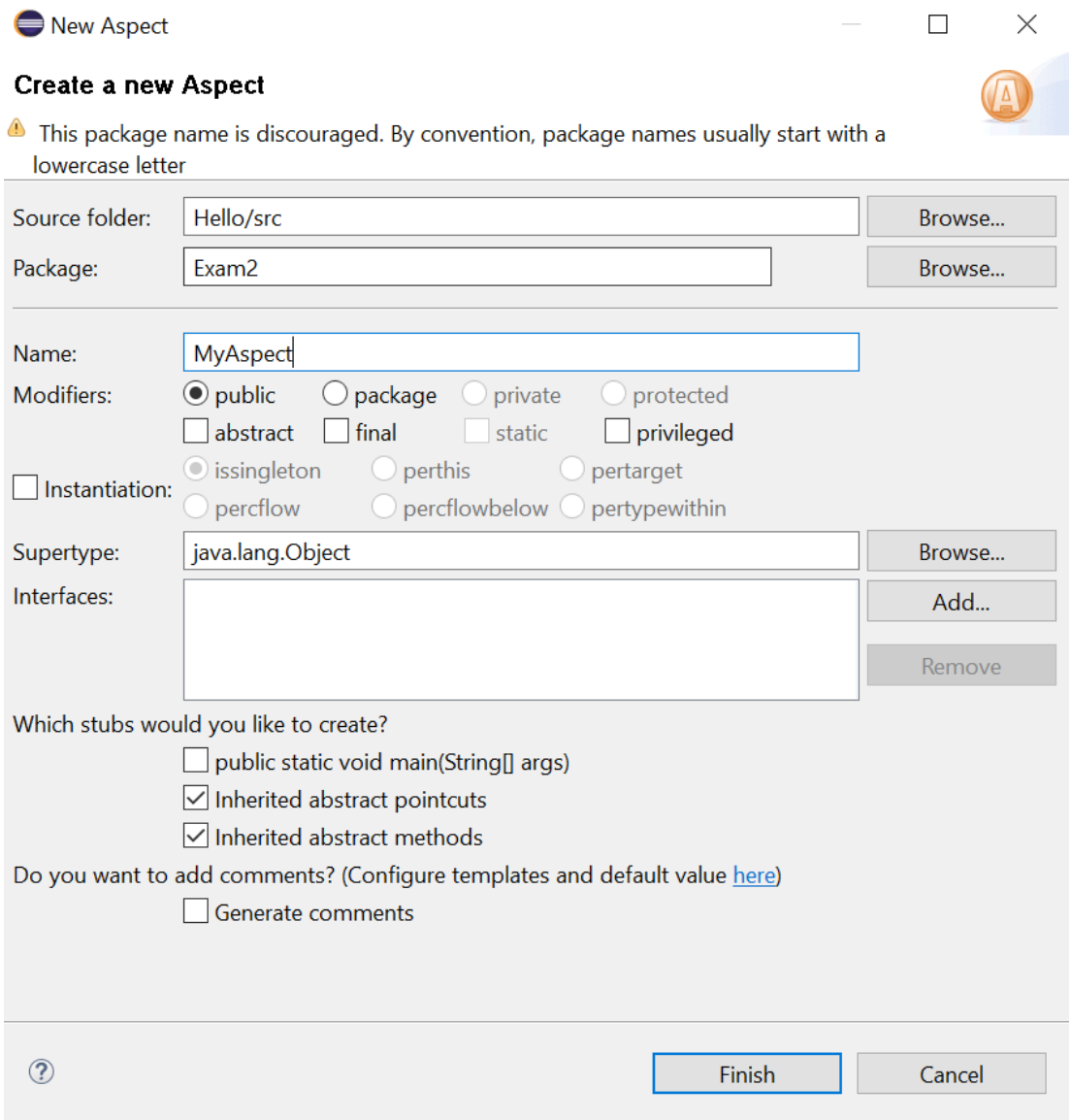


22-rasm. Yangi sinf yaratish.

Java class ga mos Aspect yaratamiz:



23-rasm. Yangi Aspect yaratish.



24-rasm. MyAspect.aj ni yaratish

AspectDemo.java fayli.

```
package Exam2;
public class AspectDemo {
 public static void main(String[] args){
      AspectDemo aop=new AspectDemo();
      aop.methodeOne(5);
      aop.methodeOne(10,"Salom");
      aop.methodeTwo("xayr");
 }
 public void methodeOne(int var){
 System.out.println("methode One (butun tipli) "+var);
 }
 public void methodeOne(int var, String str){
 System.out.println("methode One (butun va satr tipli)
"+var+" "+str);
 }
 public void methodeTwo(String str){
 System.out.println("methode One (satr tipli) "+str);
 }
}
```

MyAspect.aj fayli.

```
package Exam2;
public aspect myAspect {
 pointcut function():
      call(void AspectDemo.methodeOne(*));
 after(): function(){
      System.out.println("aspect after ... ");
 }
 before(): function(){
```

```
        System.out.println("aspect before ... ");
 }}
```

AspectDemo.java faylini kompilyatsiyab qilsak, olingan natija quyidagicha:

```
aspect before ...
methode One (butun tipli) 5
aspect after ...
methode One (butun va satr tipli) 10 Salom
methode One (satr tipli) xayr
```

Endi, AOP orqali yaratilgan bir nechta misollarni ko`rib chiqamiz.

1-masala. Mijozga biror xizmat taklif qilinadi. Buning uchun mijozning balansi tekshiriladi, qaysiki ko`rsatiladigan xizmat narxi (amount) dan kam bo`lmasligi kerak. Agar shart bajarilsa, xizmat ko`rsatiladi va balansdan yechib olinadi, hamda operatsiya muvaffaqiyatli yoki muvaffaqiyatsiz bajarilganligi haqida ma’lumot ekranga chiqariladi.

Buning uchun java da Account.java sinfini yasaymiz:

```
public class Account {
 int balance = 20;

    public boolean withdraw(int amount) {
        if (balance < amount) {
            return false;
        }
        balance = balance - amount;
        return true;
    }
    public static void main(String[] args){
      Account account=new Account();
      System.out.println(account.withdraw(25));

    }
}
Natija: False
```

Agar jarayon yakuniga yetgach mijozning qolgan balansi haqida ma’lumot chiqarish talab qilinsa, sinfga o`zgartirish qilishga to`g`ri keladi. Yuqoridagi masalada bu hech qanday muammoni keltirib chiqarmaydi. Lekin, shunga o`xshash yirik loyihalarni amalga oshirganda har bir sinf boshqa sinflar bilan o`zaro bog`liq bo`lib, birorta sinfga qilingan arzimagan o`zgartirish ham kutilmagan natijalarni yuzaga keltirib chiqarishi mumkin. bu muammoni yechishda bizga AspectJ yordam beradi. Bunda javadagi sinf kodiga o`zgartirish qilmasdan AspectJ orqali aspect yasab, o`zgartirilishi kerak bo`lgan sinf(lar) ning bir yoki bir nechta metod va maydonlari ustida amallar bajaramiz.

Endi, AspectJ yordamida AccountAspect.aj aspect yaratamiz:

```
public aspect AccountAspect {

 final int MIN_BALANCE = 10;

    pointcut callWithDraw(int amount, Account acc) :
     call(boolean Account.withdraw(*)) &&
     args(amount) && target(acc);

    after(int amount, Account balance) :
callWithDraw(amount, balance) {
    System.out.println(" "+balance.balance+" ");
    }
}
```

Natija:

20

false

Bu yerda, AspectJ orqali mijozning qolgan balansi chiqarib berilyapti, qaysiki sinfda bu narsa ko`zda tutilmagan edi.

2-masala. Telefon orqali suhbatlashish uchun qo`ng`iroq qilish kerak. Buning uchun telefon qiluvchining hisob raqami (balansi) tekshiriladi va suhbatlashgan vaqt uchun pul yechib olinadi. Qo`ng`iroq yakunlangach mijozga bu haqda xabar berilib, qoldiq summasi ko`rsatiladi (dollarda).

Buning uchun Phone.java sinfini yasaymiz:

```
public class Phone {

     public String caller;
     public double balance = 0; // dollar hisobida
     public int TIME = 13;
     public void phoned(double blnc){

     balance=blnc;
          if(time(blnc,TIME)){
               System.out.println("OK! balansingiz =
"+balance+"$");
          }

          else System.out.println("ERROR! qarzingiz =
"+balance+"$");
     }

     public boolean time(double blnc, int tm){
          tm=TIME;
          balance = blnc - 0.02 * tm;
          if(balance < 0) return false;
          return true;
     }

     public static void main(String[] args){
          Phone ph=new Phone();
          ph.phoned(1.56);
     }
}
```

Natija: `OK! balansingiz = 1.3$`

Endilikda, ushbu masalaga qo`shimcha qilib, mijozning qolgan balansini UZS so`mda chiqarish va oxringi qilingan xarajatni ko`rsatish masalasi qo`yilsin. Bu masalani AspectJ orqali quyidagicha yechamiz:

Buning uchun PhoneAspect.aj aspectini yaratamiz:

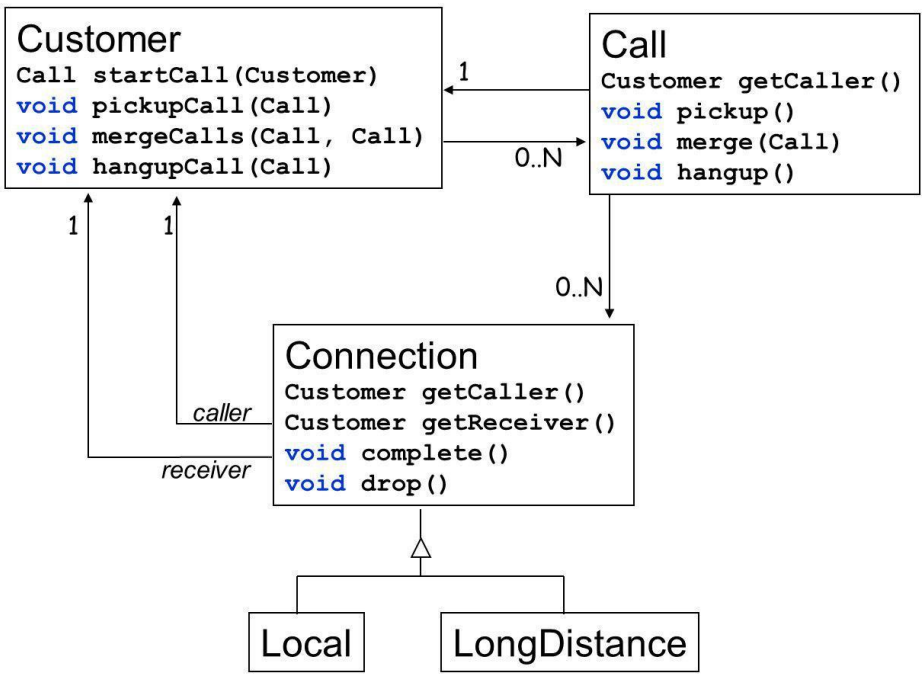
```
public aspect PhoneAspect {
```

```
     String str="aa";
     pointcut callphoneUZS(double amount, Phone balance) :
     call(void Phone.phoned(*)) && args(amount) &&
target(balance);

    after(double amount, Phone balance) :
callphoneUZS(amount, balance) {

    System.out.println(" Aspect(UZS =  " +
balance.balance*4210.33 + " so`m)");
    System.out.println("oxringi qilingan xarajat =
"+balance.TIME*0.02+"$  ");
    }
    }
```

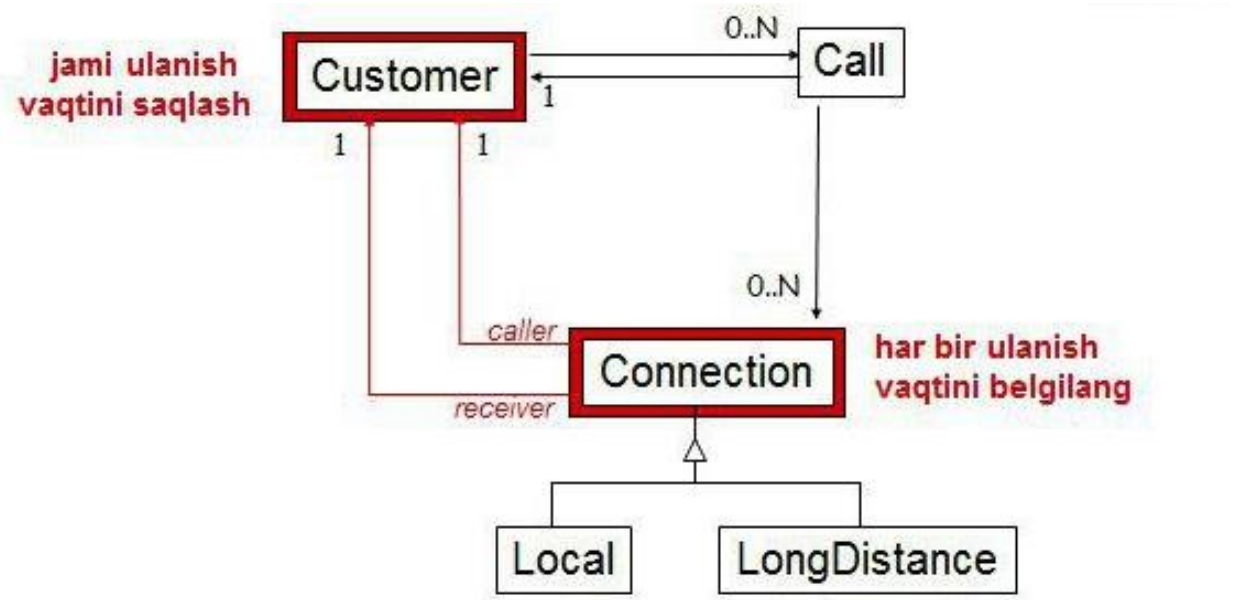
Natija:

```
OK! balansingiz = 1.3$

Aspect(UZS =  5473.429 so`m)

oxringi qilingan xarajat = 0.26$
```

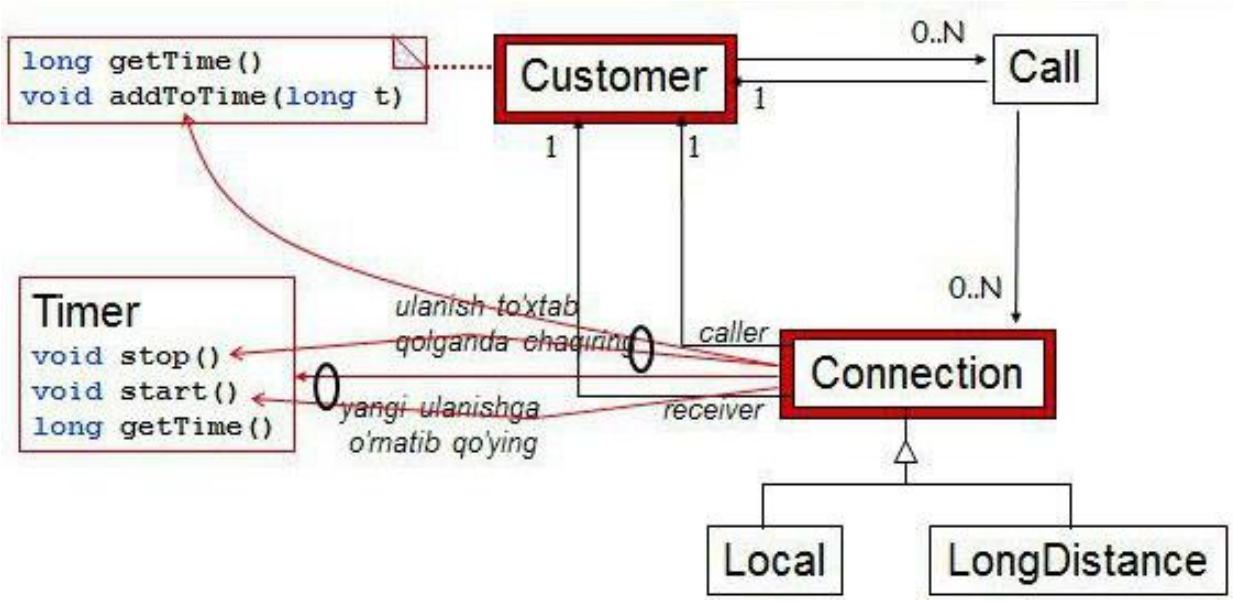
3-masala. Telekom tarmog`ida mijozlar orasidagi suhbatlashish vaqtini hisoblash masalasini ko`raylik. Buning uchun dastlab qo`ng`iroq model (design) ini yasaymiz (25, 26.1, 26.2-rasmlar).



25-rasm. Telefon tarmoq dezayni



26.1-rasm. Qo`ng`iroq vaqtini hisoblash



26.2-rasm. Qo`ng`iroq vaqtini hisoblash

Tarmoqdagi suhbatlashilgan vaqtni hisoblash masalasini aspect orqali quyidagicha amalga oshiramiz:

```
aspect Timing {
  static aspect ConnectionTiming of eachobject(instanceof(Connection)) {
    private Timer timer = new Timer();
  }

  static aspect CustomerTiming of eachobject(instanceof(Customer)) {
    private long totalConnectTime = 0;
    public long getTotalConnectTime() {
      return totalConnectTime;
    }
  }

  pointcut startTiming(Connection c): receptions(void c.complete());
  pointcut   endTiming(Connection c): receptions(void c.drop());

  static after(Connection c): startTiming(c) {
    ConnectionTiming.aspectOf(c).timer.start();
  }

 static after(Connection c): endTiming(c) {
    Timer timer = ConnectionTiming.aspectOf(c).timer;
    timer.stop();
    long currTime = timer.getTime();
    CustomerTiming.aspectOf(c.getCaller()).totalConnectTime += currTime;
    CustomerTiming.aspectOf(c.getReceiver()).totalConnectTime += currTime;
  }
}
```

Bob bo`yicha xulosalar

AOP - bu dasturiy paradigma hisoblanadi. Bu esa o'zaro bog'liq muammolarni ajratib qo'yishga imkon berib, modullik darajasini oshirishga qaratilgan. Kodni o'zi o'zgartirilmasdan mavjud kodga qo'shimcha xatti-harakatlar qo'shib, uni bajaradi.

AspectJ foydalanishga qarab turli kutubxonalar taklif qiladi. Biz Maven markaz omboridagi guruh org.aspectj ostida Maven bog'liqligini topamiz. Tadqiqotda biz tomonlarni yaratish uchun zarur bo'lgan bog'liqliklar va Weaver kompilyatsiya-vaqt, post-kompilyatsiya va load-time Weavers dan foydalanadik.

Ma’lumki, katta loyihalarni bajarishda qo`yilgan masala qismlarga ajratilib, har bir qism uchun sinflar yaratiladi. Java dasturlash tilida bu sinflar orqali amalga oshiriladi. Har bir sinf o`ziga tegishli bo`lgan masalalarni bajarishga qaratiladi. Agar biz qaysidir sinfga o`zgartirish kiritmoqchi bo`lsak, bunda java dasturlash

tilida javada yozilgan kodga o`zgartish qilishimizga to`g`ri keladi va bu dastur qaytadan kompilatsiya qilinib sinfga o`zgartirishlar kiritiladi. Bu yerda sinflar bir-biriga bog`liq bo`lishi mumkin va bu o`zgarish boshqa sinflarga ham ta’sir qilishi mumkin. Shunday muammolar tug`ilib qoladiki, unda biz dastur kodiga tegmasdan turib faqat sinfni o`zgartirishimiz kerak bo`ladi. Bunday jarayonni Java dasturlash tilida Aspectga mo`ljallangan dasturlash orqali amalga oshirishimiz mumkin. Bunda biz Javada yozilgan Aspect orqali Javada tuzilgan ixtiyoriy dastur kodiga o`zgartirish qilmay turib, sinfni o`zgartirish orqali muammoni hal qilamiz. Bu juda katta loyihalarni amalga oshirishda haqiqatdan ham juda qulay hisoblanadi. Aspect deganda biz, nuqtiy-nazar orqali qarashni tushunamiz. Ushbu tatqiqotning dolzarbligi Java dasturlash tilida Aspectga mo`ljallangan dasturlash orqali yuqorida keltirilgan muammolarni hal qilish bilan belgilanadi.

UMUMIY XULOSALAR

Dasturlash tillari yaratilgan dastlabki davrlarda kod mashina tilida yozilgan bo`lib, ular assembler (mashina) tillari deb atalgan. Mashina tilida yozilgan kodni bir kompyuterdan ikkinchi kompyuterga o`tkazish ko`plab noqulayliklar tug`dirardi. Chunki, har bir mashina o`ziga xos parametrlarga ega (masalan, registr raqami, o`qish va yozish qurilmasi, xotira fazosi va h.k.) bo`lib, bu parametrlar mashina tilida yozilgan kodda o`z aksini topardi.

Bu muammolar dasturlash tillarining yangi avlodi (yuqori sathli dasturlash tillari) – inson tiliga yaqin dasturlash tillari yaratilishi bilan hal qilindi. Xususan, Paskal, C, Ada, KARAT dasturlash tillari keng tarqalgan edi. Bu davrda asoson prosedurali dastulash tillari keng qo`llanila boshlanib, birorta tilda yozilgan kodni ixtiyoriy kompyuterda to`g`ridan-to`g`ri amalga oshirish mumkin edi.

Prosedurali dastulash tillarida algoritm ishning markazida bo`lib, qolgan obyektlar esa ikkinchi darajali hisoblanar edi. Bu yondashuvda real jarayonni abstrakt modelini yaratish, loyihasini ishlab chiqish qator qiyinchiliklarga olib kelar edi. Haqiqatdan ham, obyektning qanday tuzilishda ekanligidan qat’iy nazar algoritm ijrosini ta’minlashi zarurligi real hayotda o`z aksini topmas edi. Endi dasturchilar oldida yangi muammo yuzaga keldi. Qanday qilib haqiqiy jarayonni to`g`ridan-to`g`ri abstrakt modelini yaratish mumkin?

Bu muammoni o`tgan asrning 50-yillar oxiri, 60-yillarning boshlarida yaratilgan obyektga yo`naltirilgan dasturlash tillari orqali hal qilindi. Obyektga yo`naltirilgan yondashuvda obyekt birlamchi, algoritm esa ikkilamchi hisoblanadi. Abyektga yo`naltirilgan yondashuvning asosiy tamoyili bu – muammoni turlariga ko`ra qismlarga ajratish va shu qismlarni yagona maqsad yo`lida birlashtirishdir.

Ushbu qismlar java dasturlash tilida **sinf** (class) deb ataladi. Sinf – muayyan bir turga tegishli metod va maydonlardan tashkil topgan abstract tuzilmadir. Java dasturlash tilida o`zgaruvchi tiplariga standart sinf sifatida qarash mumkin.

Obyektga yo`naltirilgan dasturlash (OOP – object-oriented programing) tillarining asosiy termilari **obyekt** (object), **sinf osti** (extends), **merosxo`rlik (**Inheritance**)**, **polimorfizm** lar hisoblanadi.

OOP da juda ko`p sinflar o`zaro bog`liq bo`lib, bu bog`lanishda biror sinfga o`zgartirish kiritilsa, bu o`zgarish u bilan bog`langan barcha sinflarga ta’sir ko`rsatadi. Bu esa zanjir reaksiyasini yuzaga keltirib, dasturdan ko`zlangan natijaga erishishga salbiy ta’sir ko`rsatadi.

Bu muammoning yechimi ilk bora 1997-yili Kikzales G., Lamping J., Mendesh A. lar tomonidan **“Aspect-oriented programming”** nomli maqolasida o`z aksini topdi.

Aspektli yondashuv – o`zaro bog`liq muammolarni yechish hisoblanadi. AOP (Aspect-oriented programing) da sinfning biror metodi yoki maydoniga o`zgartirish qilish uchun, sinfning kodiga o`zgartish kiritilmaydi, balki, alohida aspekt aj. kengaytmali kod yozib, AOP nin standart funksiya (**after, before, around**) lari orqali sinfning metod yoki maydonlari chaqiriladi. Bu esa o`zaro bog`langan sinflar strukturasida hech qanday muammoni yuzaga keltirmaydi.

Xulosa shundan iboratki, AOP faqatgina obyektga yo`naltirilgan dasturlash tillari uchungina xizmat qiladi, ya’ni AOP OOP dan mutaqil holda ishlatilmaydi.

Mazkur tadqiqot nazariy jihatdan asoslangan bo`lib, unda quyidagi natijalarga erishildi:

1. Java dasturlash tili nazariy va amaliy jihatdan yoritib berildi
2. AOP ning konsepsisiyalar o`rganldi;
3. AOP ning funksionalligini oshirildi;
4. AOP yordamida OOP ning o`zaro bog`liq bo`lgan muammolar yechildi;
5. aj. kengaytmali kodni amalga oshirildi.

FOYDALANILGAN ADABIYOTLAR RO`YXATI:

I. Ijtimoiy-siyosiy adabiyotlar:

1. Мирзиёев Ш.М. Эркин ва фаровон, демократик Ўзбекистон давлатини биргаликда барпо этамиз.– Т.: Ўзбекистон, 2016. – 56 б.

2. Мирзиёев Ш.М. Буюк келажагимизни мард ва олижаноб халқимиз билан бирга қурамиз.– Т.: Ўзбекистон, 2017.

3. Каримов Ислом. Юксак маънавият – енгилмас куч. –Т.: Маънавият, 2008. 176 б.

4. Mirziyoyev Sh.M. Erkin va farovon, demokratik O`zbekiston davlatini birgalikda barpo etamiz. O`zbekiston Respublikasi Prezidenti lavozimiga kirishish tantanali marosimiga bag`ishlangan Oliy Majlis palatalarining qo`shma majlisidagi nutq.-Toshkent: “O`zbekiston” NMIU,2016.-56b.

II. Ilmiy risola, darslik va o`quv qo`llanmalari:

5. Kindler, E.; Krivy, I. "Object-Oriented Simulation of systems with sophisticated control". International Journal of General Systems: pp. 313–343. 2011.

6. Byous, Jon "Java technology: The early years". *Sun Developer Network*. Sun Microsystems. Archived from the originalon April 20, 2005. Retrieved 2005-04-22

7. Object-oriented programming "The History of Java Technology". *Sun Developer Network. c. 1995*. Archived *from the original on February 10, 2010*. Retrieved April 30, 2010.

8. "So why did they decide to call it Java? Archived November 15, 2013, at the Wayback Machine", Kieron Murphy, JavaWorld.com, 10/04/96

9. Object-oriented Programming with Java: Essentials and Applications. *Tata McGraw-Hill Education. p. 34.*

10. Chander, Sharat. "Introducing Java SE 11". oracle.com. Archived from the original on September 26, 2018. Retrieved September 26, 2018

11. Guttag, J. 1980. Abstract Data Types and the Development of Data Structures. In Programming Language Design. New York, NY: ComputerSociety Press

12. Michael T. Goodrich, Roberto Tamassia and Michael H. Goldwasser – Data Structures and Algorithms in Java, 6th edition Wiley, 2014

13. P. Ammann and J. Offutt, Introduction to software testing. Cambridge University Press, 2008

14. R. Lipton, "Fault diagnosis of computer programs," tech. rep., Carnegie Mellon University,1971.

15. R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," Computer, vol. 11, no. 4, pp. 34–41, 1978.

16. E. Omar, S. Ghosh, and D. Whitley, "Constructing subtle higher order mutants for Java and AspectJ programs," in International Symposium on Software Reliability Engineering, pp. 340–349, 2013.

17. E. Omar, S. Ghosh, and D. Whitley, "HOMAJ: A tool for higher order mutation testing in AspectJ and Java," in *Workshop on Mutation Analysis*, pp. 165–170, 2014.

18. K. H. T. Wah, "An analysis of the coupling effect I: single test data," *Science of Computer Programming*, vol. 48, no. 23, pp. 119 – 161, 2003.

19. R. Delamare, B. Baudry, and Y. Le Traon, " AjMutator: A Tool for the Mutation Analysis of AspectJ Pointcut Descriptors," in *International Conference on Software Testing, Verification, and Validation, Workshop*, pp. 200–204, 2009.

20. G. Kiczales1, E. Hilsdale2, J. Hugunin2, M. Kersten2, J. Palm2, and W. G. Griswold3, "An overview of AspectJ," in *European Conference on Object-Oriented Programming*, pp. 327–354, 2001.

21. Vaskaran Sarcar "Java Design Patterns", pp. 128-133, 2011.

III. Ilmiy maqolalar:

22. U. Khayriyev, O. Jalolov “Introduction to AspectJ”. Taffakkur va talqin jurnali, pp 231-234, Buxoro, 2018.

23. Jalolov O., Xayriyev U. “Aspectga yo`naltirilgan dasturlash”, “XXI asrda ilm-fan taraqqiyotining rivojlanish istiqbollari va ularda innovatsiyalarning tutgan o`rni” konfrensiyasi 2-qism, pp 16-19, Toshkent, 2019.

24. Stephen Gilbert and Bill McCarty Object-Oriented Design in Java, ISBN: 1571691340 1998, p. 72

25. "JAVASOFT SHIPS JAVA 1.0". *Archived from* the original *on March 10, 2007*. Retrieved 2008-02-05.

26. "JAVASOFT SHIPS JAVA 1.0". Retrieved 2018-05-13.

IV. Dissertatsiya va dissertatsiya avtoreferatlari:

27. Б.А. Скрипаль диссертация на соискание ученой тепени МАГИСТРА Тема: “Разработка аспектно-ориентированного расширения для языка Kotlin”, Санкт-Петербургский политехнический университет, 2017.

28. Elmahdi Omar dissertation of “CONSTRUCTING SUBTLE HIGHER ORDER MUTANTS FROM JAVA AND ASPECTJ PROGRAMS”, Department of Computer Science, 2015.

V. Internet manbalari

29. https:// www.webofsience.com

30. https://www.scopus.com

31. https://www.siencedirect.com

32. http://www.eclipse.org

33. https://dasturchi.uz

34. https://dasturim.uz