TUZUVCHI

G’ayratjon Rayimjonov

Ilm bu boylik uni o’rgatish zakot.....

Java nima

Java **dasturlash tili** va **platform**

Java yuqori himoyalangan va obyektga yo’naltirilgan dashturlash tili

Platforma: dastur bajarila oladigan ixtiyoriy apparat yoki dasturiy muhit platformadir. Javaning ham o`zini mahsus bajarilish muhiti - platformasi mavjud (JRE - Java Runtime Environment).

Javadan qayerda foydalaniladi

Sun firmassining ma’lumotlariga qaraganda 3 mlr atrofidagi qurulma javadan foydalanadi

Mana ulardan ba`zilari:

-Shaxsiy kompyuter dasturlari (Desktop Applications) - acrobat reader, media-pleyer, antiviruslar va h.k.

-Web-dasturlar

-Korxona-tashkilotlar dasturlari (Enterprise Applications) - bank yoki ishlab chiqarishga oid dasturlar

-Mobil dasturlar

-Smart kartalar

-Robotlar

-O`yinlar

Asosiy konsepsiyalari:

Java quyidagi 5 maqsad uchun qurilgan, u shunday til bo'lishi kerakki:

1. Oddiy, obyektga mo'ljallangan, taqsimlangan va o'rganishga oson bo'lsin.

2. Mustahkam va xavsiz bo'lsin

3. Qaysidir qurilma platformasidan yoki uning arxitekturasidan mustaqil bo'lsin(ya'ni qaysidir platformaga tobe bo'lmasin).

4. Juda samarali bo'lsin.

5. Dasturlash tili uchun tarjimon(interpreter) yozish mumkin bo'lsin. Shuningdek dasturlash tili parallel ishlashni va dinamik tiplashda foydalanishni ta'minlay olsin.

Javada qanaqa dasturlar yoziladi: asosan 4 tipdagi

1) **Standalone applications** – Linux, Mac yoki Windowsga o`rnatib, kundalik ishladatigan dasturlarimiz: mp3 pleyer, ofis, antivirus kabilar. Ular AWT, Swing yoki JavaFX texnologiyalari orqali tuziladi

2) **Web Applications** – tarmoq orqali ishlovchi ixtiyoriy dasturlar. Eslatma: web dasturlar ikki qismdan, server tomon hamda klient tomon (brauzer) dan iborat. Javada faqat server tomoni uchun yoziladi. Bunda servlet, jsp, jsf kabi fundamental texnologiyalardan boshlab Spring, Play kabi freymworklar qo`l keladi. Umuman olganda brauzer uchun HTML, CSS hamda Java Scriptdan boshqa tilda yozib bo`lmaydi.

3) **Enterprise Applications** – bu dasturlar yirik salmoqqa ega bo`lib, odatda katta jabhalarda ishlatiladi. Banklar, tashkilotlar yoki astronomiya kabi sohalarda. Ular yuqori xavfsizlik, yuklamani (nagruzka) serverlarga teng taqsimlash (load balancing) yoki klasterlash (clustering – katta tizimdan xuddi yagona obyekt sifatida foydalanish) kabi sifatlarni talab qiladi. Javada bular bor.

4) **Mobile Applications** – Mobil qurilmalarga mos dasturlarni ham Javada yozish mumkin. Androiddan boshlab, Java ME (JME – Java Micro Edition) gacha. JME ga misol qilib, Nokia telefonlarimiz uchun ishlangan JAR o`yinlarni misol keltirish mumkin.

Java dasturlash tili - eng yaxshi dasturlash tillaridan biri bo'lib unda korporativ darajadagi mahsulotlarni(dasturlarni) yaratish mumkin. Bu dasturlash tili OAK dasturlash tili asosida paydo bo'ldi. Oak(ma'nosi eman daraxti) dasturlash tili 90-yillarning boshida Microsystems (hozirda Oracle nomidan ish yuritadi) tomonidan platformaga(operatsion tizimga) bog'liq bo'lmagan holda ishlovchi yangi avlod aqlli qurilmalarini yaratishni maqsad qilib harakat boshlagan edi. Bunga erishish uchun Sun hodimlari C++ ni ishlatishni rejalashtirdilar, lekin ba'zi sabablarga ko'ra bu fikridan voz kechishdi. Oak muvofaqiyatsiz chiqdi va 1995-yilda Sun uning nomini Java ga almashtirdi, va uni WWW rivojlanishiga hizmat qilishi uchun ma'lum o'zgarishlar qilishdi.
Java 1990 yillarda ishlab chiqarila boshlangan bo'lsa ham, uning birinchi versiyasi(Java 1.0) 1996 yil ommaga taqdim etilgan. Undan so'ng keyingi versiyalar sekin-astalik bilan chiqa boshladi: **1998 yil - Java 2, 2004 yil - Java 5.0, 2006 yil - Java 6, 2011 yil - Java 7, 2014 yil - Java 8.**
Java **Obyektga Yo'naltirilgan Dasturlash** (OOP-object oriented programming, ООП) tili va u C++ ga ancha o'xshash. Eng ko'p yo'l qo'yildigan xatolarga sabab bo'luvchi qismalari olib tashlanib, Java dasturlash tili ancha soddalashtirildi.
Java texnologiyasi o'ta sodda, xavfsizlikni yuqori darajada ta'minlab bera oladigan, kuchli, to'la obyektga yo'naltirilgan dasturlash tili bo'lib, muhit (**platforma**)ga bo'liq bo'lmagan holda ishlaydi. U bilan xatto eng kichik qurilmalarga ham dasturlar yozish mumkin. Java texnologiyasi to'laligicha Javaning sintaksisi C++ ga asoslangan. Shuning uchun C++ tilini biladiganlar Javani oson o`rganishadi. Lekin undagi ko`pchilik xususiyatlar olib tashlangan. Masalan: Pointer(ko`rsatkich)lar bilan to`g`ridan to`g`ri ishlash, ya`ni Javada alohida ko`rsatkich tushunchasi yo`q. Operatorlarni qayta yuklash ham olib tashlangan. Yana eng muhimi, ishlatilmaydigan xotira (unreferenced objects) avtomatik tozalanadi. Buni Javadagi Garbage Collector (GC – chiqindi yig`ishtirgich) amalga oshiradi. C++ da bu destruktorlar orqali qo`lda (manual –

ruchnoy) qilingan. Bundan tashqari Java har bir yangi versiyada bundan qulay imkoniyatlarni qo`shib kelmoqda. Ayni paytda oxirgi versiya 8-sidir. 2014-yil 18-martda ommaga e'lon qilindi.

Javaning qisimlari

Java bir necha qisimlarni o’z ichiga oladi ular :

1. Simple(oddiy)
2. Object-Oriented(obyektag yo’naltirilgan)
3. Platform independent(mustaqil platformalar)
4. Secured(himoyalngan)
5. Robust(kuchli)
6. Architecture neutral(netral arxitektura)
7. Portable(ixcham)
8. Dynamic(dinamik)
9. Interpreted
10. High Performance(yuqori platforma)
11. Multithreaded()
12. Distributed

Simple

Javaning sintaksisi C++ ga asoslangan. Shuning uchun C++ tilini biladiganlar Javani oson o`rganishadi. Lekin undagi ko`pchilik xususiyatlar olib tashlangan. Masalan: Pointer(ko`rsatkich)lar bilan to`g`ridan to`g`ri ishlash, ya`ni Javada alohida ko`rsatkich tushunchasi yo`q. Operatorlarni qayta yuklash ham olib tashlangan

Object-Oriented(obyektag yo’naltirilgan)

Object – oriented dasturda turli xil obyektlarini ularning turli xil hususiyatga ega ma’lumotlarni qoshish tushuniladi.

Ob'ekt yo'naltirilgan dasturlash (oops) metodologiyasi ba'zi qoidalar bilan ta'minlash orqali dasturiy ta'minot ishlab chiqishni xizmatni soddalashtiradi

Oop asosiy tushunchalar:

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

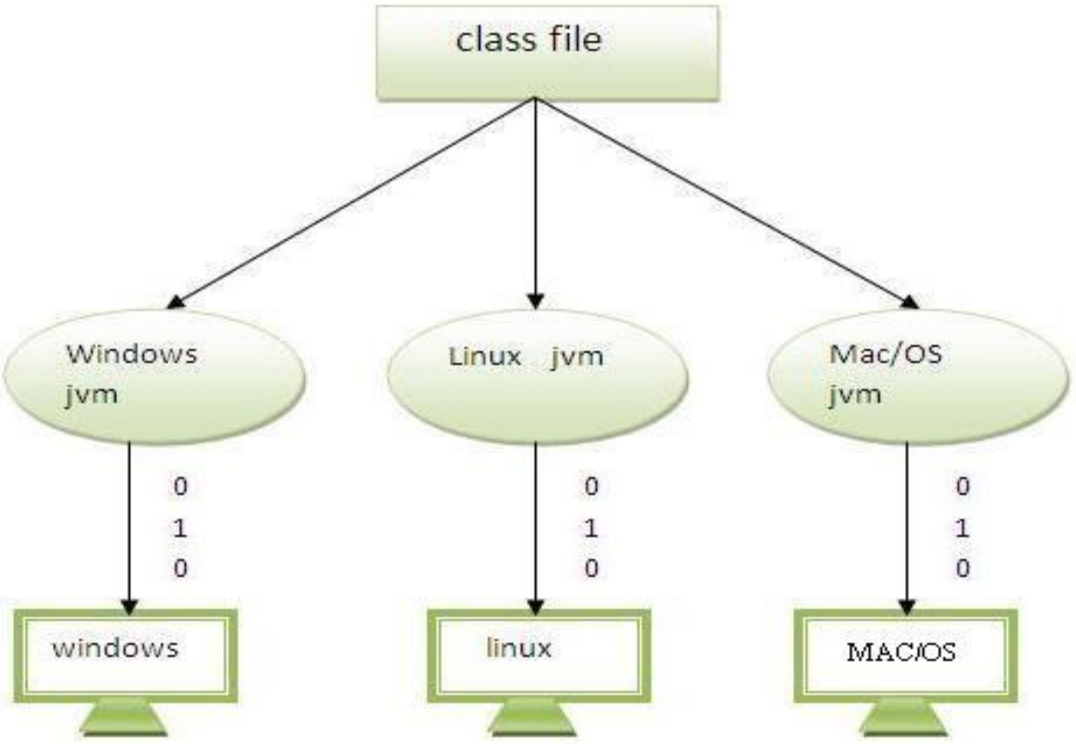
Platform Independent

Dastur bajarila oladigan ixtiyoriy apparat yoki dasturiy muhit platformadir. Platformani 2 turi mavjud

Software-based va **hardware-based.** Java Software-based platformasini tamininlab beradi. Java platformasi ikki komponentdan iborat boshqa dastur bajarilishi taminlovchi platform va qurulma platformasi bularga

1. Runtime Environment
2. API(Application Programming Interface)

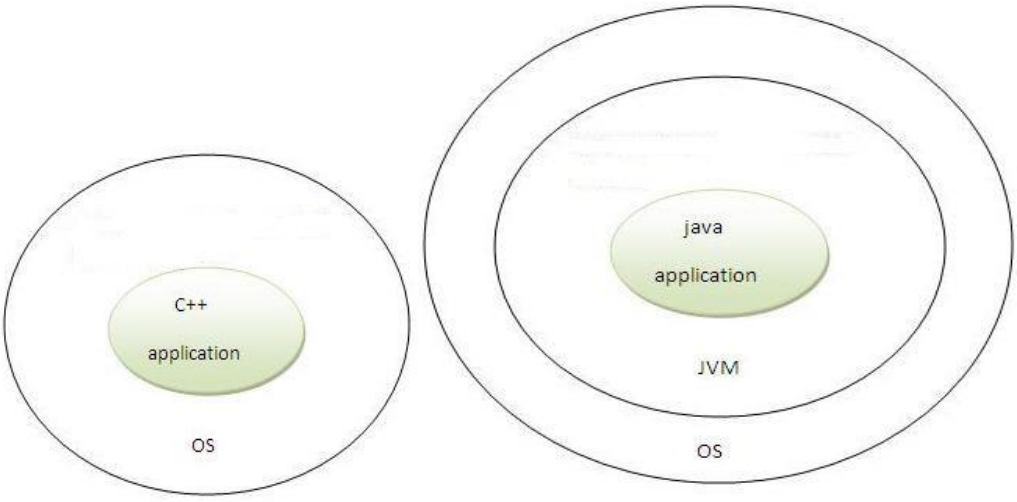
Java codalari birnecha platformalarda ishlash qobilyatiga ega misol uchun: Windows,Linux,Sun Solaris,Mac/OS va.h.k. java code compile bo'lganda platforam uni bayt kodga o'girib beradi bayt kod esa mustaqil kod bo'lib boshqa platformalarda ishlash qobilyatiga ega.



Secured

Java xafsiz chunki:

- Ochiq ko'rsatkichga ega emas
- Dastur virtual mashina muhitida ishlaydi



Robust

Robust kuchli degan ma’noni bildiradi. Java kuchli xotira boshqarividan foydalanadi.

Javada oddiy dastur

Eng birinchi oddiy dasturdi yaratish uchun talab qilinadigan dasturlar

- JDK o’rnatiladi agar bo’lmasa, download the JDK and install it.
- set path of the jdk/bin directory. http://www.javatpoint.com/how-to-set-path-in-java
- dasturlash muhiti eclips yokiy netbeabs
- java dasturi yaratiladi
- dasturni compail qilinadi

```
1. class Simple{
2.     public static void main(String args[]){
3.      System.out.println("Hello Java");
4.     }
5. }
```

Ekranda :Hello Java

class javada kalit so’z **Simple** klass nomi
public – murojoat huquqi

java dasturlash tilida 4 ta murojat huquqi bor

1. public- biz yaratgan proyektimizda hohlagan paketdan murojat qilishimiz mumkin
2. protected – faqat yaratgan paketimizdan murojatga ruxsat
3. privte – faqat class ichida murojat
4. paket

static -bu yaratgan funksiyamiz umumiyligini bildiradi
void – bu funksiya qiymat qaytarmasligini bildiradi
dasturlashda funksilar ikki xil bo'ladi qiymat qaytardigan va qiymat qaytarmaydigan void funksilar qiymat qatarmaydi

main – esa dastur ishlashi bilan **main** funksiyasiga murojat qiladi

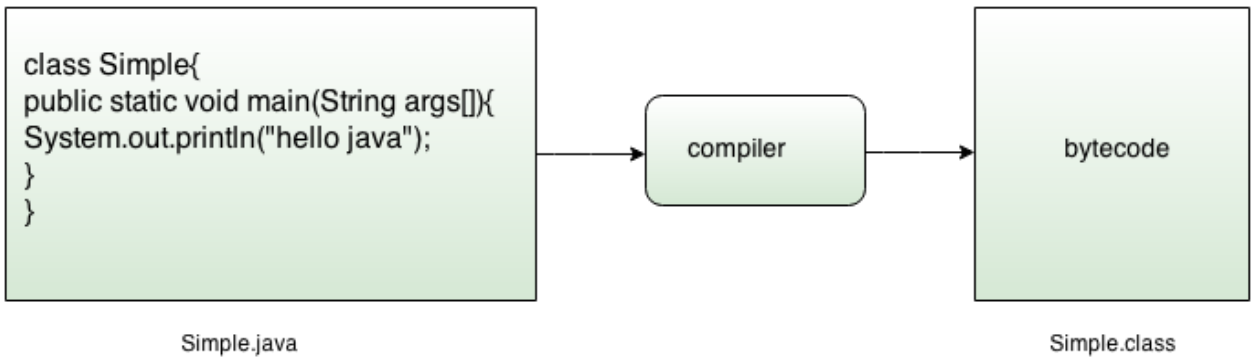
foydalanish mumkun bo'lgan main funnksiylara

1. **public static void** main(String[] args)
2. **public static void** main(String []args)
3. **public static void** main(String args[])
4. **public static void** main(String... args)
5. **static public void** main(String[] args)
6. **public static final void** main(String[] args)
7. **final public static void** main(String[] args)
8. **final strictfp public static void** main(String[] args)

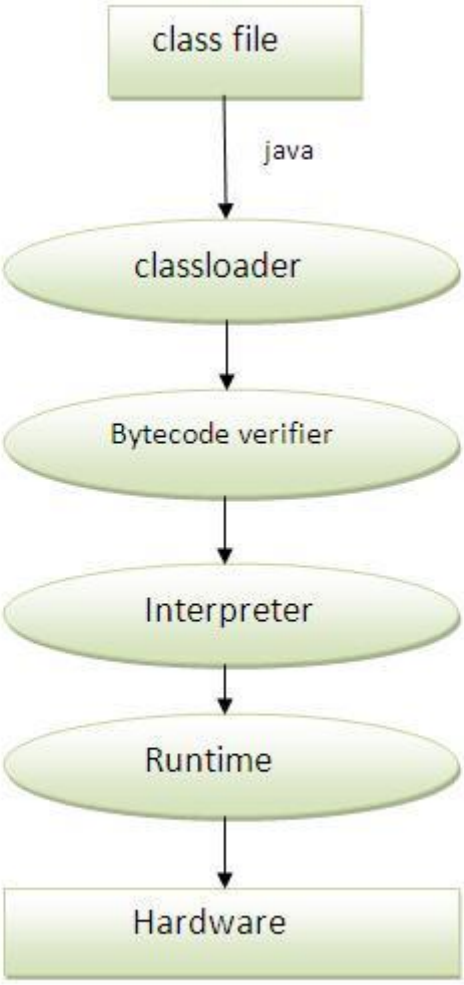
foydalanish mumkun bo'lmagan **main** funksiyalar

1. **public void** main(String[] args)
2. **static void** main(String[] args)
3. **public void static** main(String[] args)
4. **abstract public static void** main(String[] args)

Compile vaqtida java file compile bo'lib baytcode ga o'giriladi



Dastur bajarilish vaqtida sodir bo'ladigan jarayonlar



Classloader : JVM ning quyi tizimi hisoblanadi class fayllarni yuklash uchun ishlatiladi

Bytecode Verifier: checks the code fragments for illegal code that can violate access right to objects.

Interpreter: baytkodni o'qib ko'rsatmalarni amalga oshiradi

JDK, JRE va JVM farqlari

JVM

JVM ning vazifasi tarjimonlik ya'ni, dastlab biz yozgan ***.java** fayl kompilyator yordamida **bayt kod** ga o'giriladi va JVM yordamida esa mashina tiliga aylantiriladi. Bu degani JVM qaysi platformaga tegishli bo'lsa, kodlarni ham o'sha platformaga moslab beradi.

JVM ni ko’pgina qurulmalar va dasturiy taminotlar uchun ishlatish mumkin. Har bir OS uchun JVM JRE va JDK lar konfugratsiyasi farq qiladi chunkiy bular platformaga bog’liq.Lekin java mustaqil platform hisoblanadi.

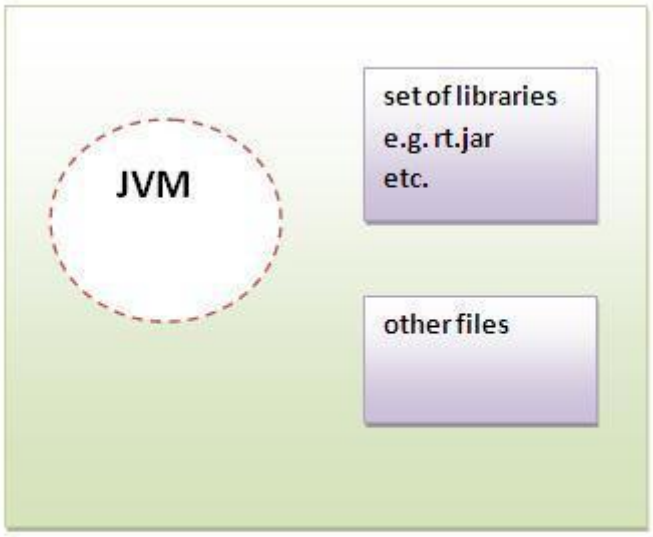
JVM ning amalga oshiradigan asosiy vazifalari

- Loads code(kod yuklanishi)
- Verifies code(tekshirilgan kod)
- Executes code(bajarilgan kod)
- Provides runtime environment(dasturni bajarilishini tamnilash)

JRE

JRE – Java Runtime Environment

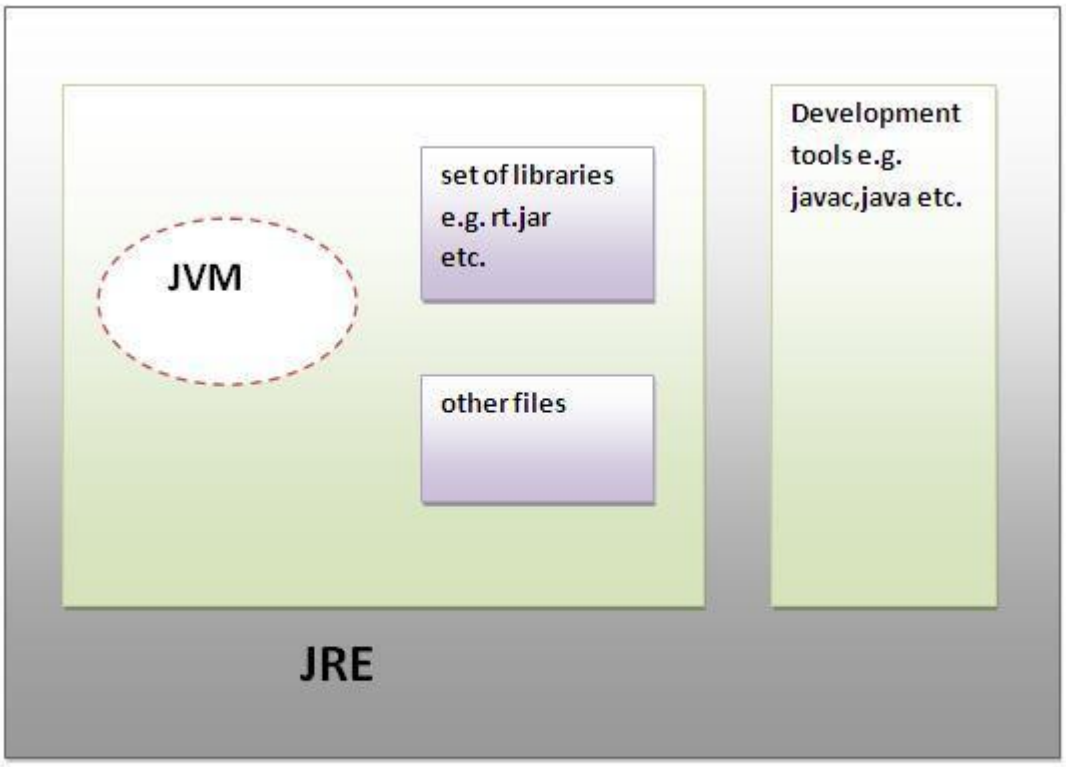
JRE bu faqat dastur bajarilishi uchun kerak bo`lgan muhit, xolos. Dasturchi bo`lmagan oddiy foydalanuvchilarga Java dasturlari bajarilishi uchun JRE yetarli



JRE

JDK – Java Development Kit

JDK = JRE + dasturlashga oid qo`shimcha instrumentlar. Bu esa dasturchilarga kerak. Formuladan ko`rinib turibdiki, JDK ning hajmi JREdan katta bo`ladi. U bir marta ko`chirib o`rnatiladi xolos.



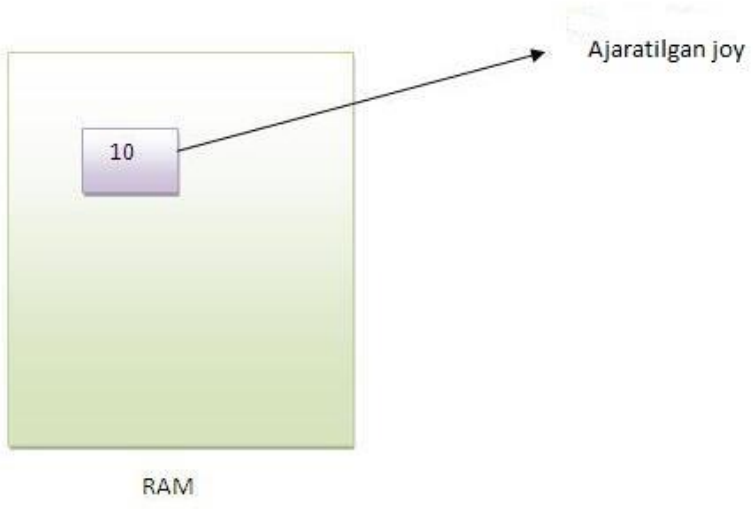
Javada o’zgaruvchilar va ma’lumot turlari

O’zgaruvchi xotiradan ajratilgan himoyalangan maydon nomidir.

Oddiy qilib tushuntiradigan bo’lsak o’zgaruvch ma’lum bir turdagi ma’lumotni o’zida saqlovchi va o’lchami chegaralangan idish. Tushunarliroq bo’lishi uchun bir ikkita hayotiy misollar keltiraman: meva solish uchun tayyorlagan savatga suv sola olmaymiz o’zgaruvchilar ham shunday bir turdagi o’zgaruvchi uchun ajratilgan joyga boshqa turdagi o’zgaruvchini saqlay olmaymiz.

4 litirlik idishga 5 litir suv quay olmaymiz chunkiy 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’ulotning o’lchami xotiradan ajratilgan joydan oshib ketsa dastur xato beradi.

5 baytlik butun sonni ma’lumot turi int bo’lgan o’zgaruvchiga saqlay olmaymiz chunkiy 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

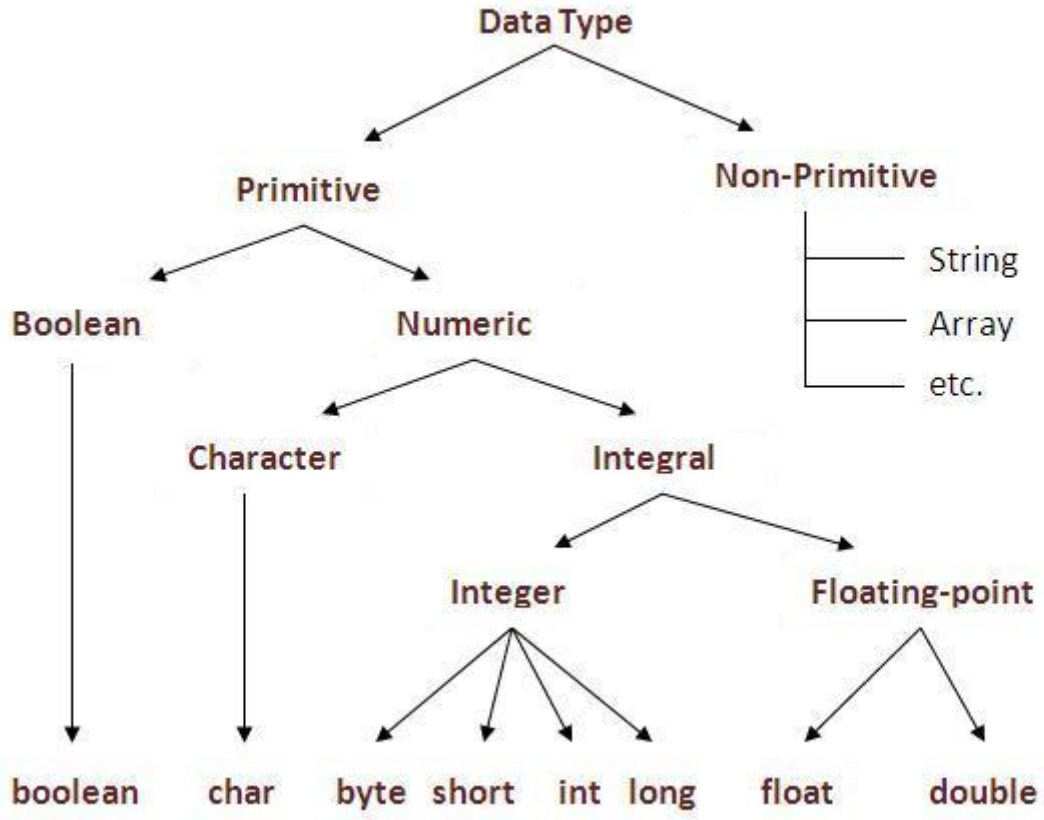
- local variable
- instance variable
- static variable

local variable - funksiya ichida e'lon qilinadi bu o'zgaruvchilar lakal(mahaliy) o'zgaruchilar deyiladi.
instance variable – class ichida e'lon ilinadi
static variable static deb e'lon qilingan o'zgaruvchi static o'zgaruvchi deyiladi. Bu local(mahalliy) bo'lishi mumkin emas.

misol uchun:

```
1. class A{
2. int data=50;//instance variable
3. static int m=100;//static variable
4. void method(){
5. int n=90;//local variable
6. }
7. }//end of class
```

Javada ma’lumot turlari (data types)
Javada ma’lumotlar turlari 2 ta
1. Soda(**primitive**)
2. Soda bo’lmagan (non primitive)



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

```
public class Varibles {
```

```
    public static void main(String[] args) {
// TODO Auto-generated method stub
        System.out.println(Character.SIZE / 8 + " byte");
        System.out.println(Byte.SIZE / 8 + " byte");
        System.out.println(Short.SIZE / 8 + " byte");
        System.out.println(Integer.SIZE / 8 + " byte");
        System.out.println(Long.SIZE / 8 + " byte");
        System.out.println(Float.SIZE / 8 + " byte");
        System.out.println(Double.SIZE / 8 + " byte");
    }
}
```

Java dasturlash tilida eng ko’p ishlatilardigan operatorlar (**solishtirish operatorlari, mantiqiy operatorlar, o‘zlashtirish operatori**)

Solishtirish operatori

Ikki operatorni bir-biri bilan solishtirishda ishlatialdi. Odatda, solishtirish operatorlar, shart berish operatori(**if**) va sikl(**while for**) ammalari bilan ishlatialdi.
Bu aperatrlar ikki xil natija qaytarishi mumkin **true** yokiy **false**
Java dasturlash tilida quyidagi solishtirish operatorlari mavjud:
== -> teng
!= -> teng emas
> -> katta
< -> kichik
>= -> katta yokiy teng
<= -> kichik yokiy teng

```
public class CompareOperators {
public static void main(String[] args) {
int a = 3;
int b = 6;
boolean c;
c = (a==b);
System.out.println(c);
}
}// false
```

int tipida a va b sonlariga 3 va 6 sonlarini o’zlashtirdik
Boolean tipida **c** ni yaratdik **a** va **b** sonlarini solishtirish

operatori yordamida (**a==b**) solishtirdik solishtirishimiz natijasida biz **true** yokiy **false** qiymatini olamiz.Demak **3** va **5** sonlari bir biriga teng emas natijada biz **false** qiymatini olamiz va u qiymatni **c** ga o'zlashtirib qo'ydik.Ekranda false qiymati chiqdi.

Mantiqiy operatorlar.

Mantiqiy operatorlar natijasi true yoki false bo'lgan operandlar ustida amalga oshiriladi. Bu operatorlar quyidagilardan iborat.

&&(&) - mantiqiy VA(AND).
||(|) - mantiqiy YOKI(OR).
^ - mantiqiy XOR(YOKI inkori)
! - mantiqiy YO'Q(NOT)
|| - qisqartirilgan YOKI(OR)
&& - qisqartirilgan VA(AND)

Yuqoridagi operatorlar orqali ikki operand qiymatni solishtiramzi

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

Shu opratorlaga oid misol ko'ramiz:

```
public class CompareOperators {

    public static void main(String[] args) {
        boolean a = true;
        boolean b = false;
        boolean c, d, e, f;
        c = a & b;
        d = a | b;
        e = a ^ b;
        f = !a;
        System.out.println("a&b = " + c);
        System.out.println("a|b = " + d);
        System.out.println("a^b = " + e);
        System.out.println("!a = " + c);
    }
}
//a&b = false
```

```
//a|b = true
//a^b = true
//!a = false
```

O‘zlashtirish operatori.

o’zlashtirish operatori tenglik(=) bilan ifodalanadi.Tepada ko’rgan misolimizga qaraydigan bo’lsak, **c =(a==b)** a va b ni tekshirganimizdan chiqqan qiymatni c ga o’zlashtiryapmiz.
Hulosa qilib shuni aytish mumkunkiy o’zlashtirish bilan solishtirish operatorlari farqi (==) bo’lsa solishtirish (=) bo’lsa o’zlashtirish bo’ladi.

If tanlash operatori

Java dasturlash tilida ikkita tanlash operatori bo’lib ular **if** va **switch** lardur masalani qo’yilishiga qarab ularning birini ishlatish mumkin.
If operatori kodlashni ikkita yo’ldan biriga burub yuboradi

Hayotda shart tekshirish operatorlarini shunchalik ko’p ishlatamizkiy. Hato ishlatganimizni ham sezmaymiz.Tasvur qiling siz bekatda turupsiz sizga 11-yo’nalishdagi aftobus kerak. Uzoqdan kelyotgan aftubusga 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 11 bo’lsa aftobusga chiq aks holda keyingisini kut degan buyruq asosida ma’lumotni tekshiradi. 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.

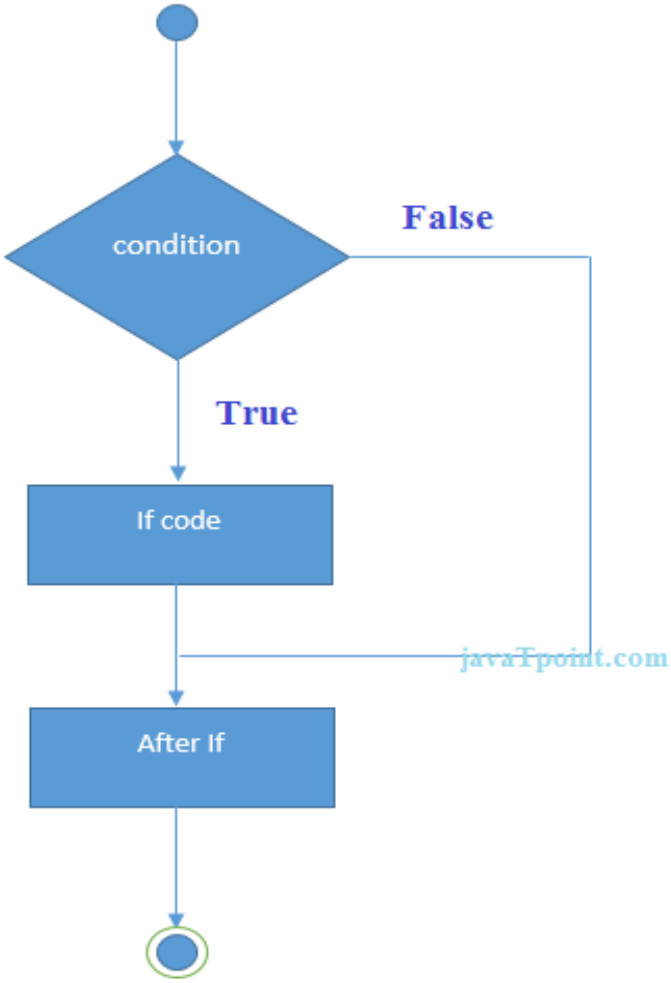
`If` tanlash operatori turlari

- if statement
- if-else statement
- nested if statement
- if-else-if ladder

Java IF tanlash aperatorining tarifi

If operatori shart tekshiradi agar shart true(rost)bo’lsa amal bajariladi aks holda shart bajarilmaydi.

```
1. if(shart){
2. //code to be executed
3. }
```



Misol:

Tasavur qiling do'konga xaridor kirib tamaki maxsulotini sotib olmoqchi bo'lib sotuvchidan tamaki mahsulotini so'radi. Sotuvchi xaridordan yoshini so'radi va xaridor sotuvchiga Yoshi 17 da ekanligini aytdi.

Keling endi sotuvchining miyyasida qanday jarayon bo'layotganini dasturga o'girib chiqsak:

Quloqlari orqali xaridorning yoshi nechida ekanligi miyaga kiruvchi qiymat bo'lib kirib keldi.

Qonunga ko'ra 18 yoshdan kichik bo'lgan xaridorlaga tamaki sotish taqiqlanadi.

```
1. public class IfExample {
2. public static void main(String[] args) {
3.    int xaridorningYoshi=17;
4.     if(idorningYoshi < 18){
5.         System.out.print("Tamaki mahsuloti sotilmasin");
6.     }
7. }
8. }
```

Ekranda:

```
Tamaki mahsulot sotilmasin
```

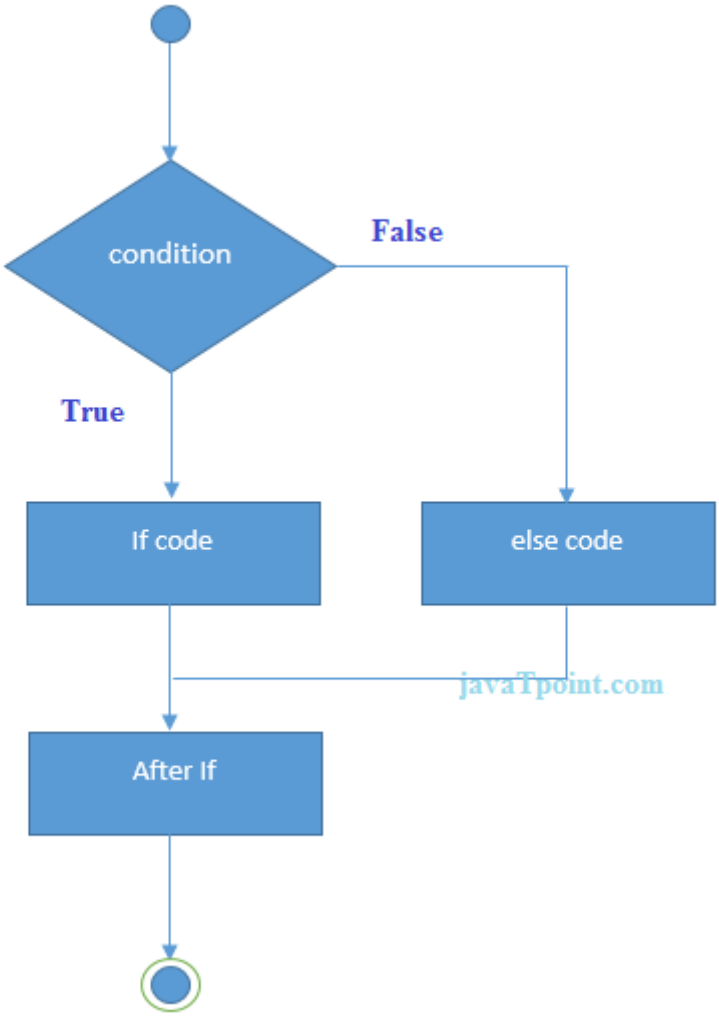
int xar idorningYoshi=17; -> kiruvchi qiymat

if(idorningYoshi < 18) -> tekshiruv jarayoni

Java IF-else operatori tarifi

Shart teksherish jarayonida shart true bo'lsa if blokdagi amal bajarladi aks holda else block dagi amal bajariladi

```
1. if(condition){
2. //code if condition is true
3. }else{
4. //code if condition is false
5. }
```



Misol:

```
1. public class IfElseExample {
2. public static void main(String[] args) {
3.     int number=13;
4.     if(number%2==0){
5.         System.out.println("juft son");
6.     }else{
7.         System.out.println("toq son");
8.     }
9. }
10.     }
```

Ekranda:

```
Toq son
```

Java IF-else-if ketma-ketlik operatori tarifi

O’zbekcha qilib aytganda unisi bo’lmasa bunis, bunisi bo’lmasa keyingisi degan ma’noni bildiradi.

if-else-if ketma ketligi shartlar ko’p bo’lgan hollarda ishlatiladi.

```
1. if(condition1){
2. //code to be executed if condition1 is true
3. }else if(condition2){
4. //code to be executed if condition2 is true
5. }
6. else if(condition3){
7. //code to be executed if condition3 is true
8. }
9. ...
10.     else{
11.     //code to be executed if all the conditions are false
12.     }
```

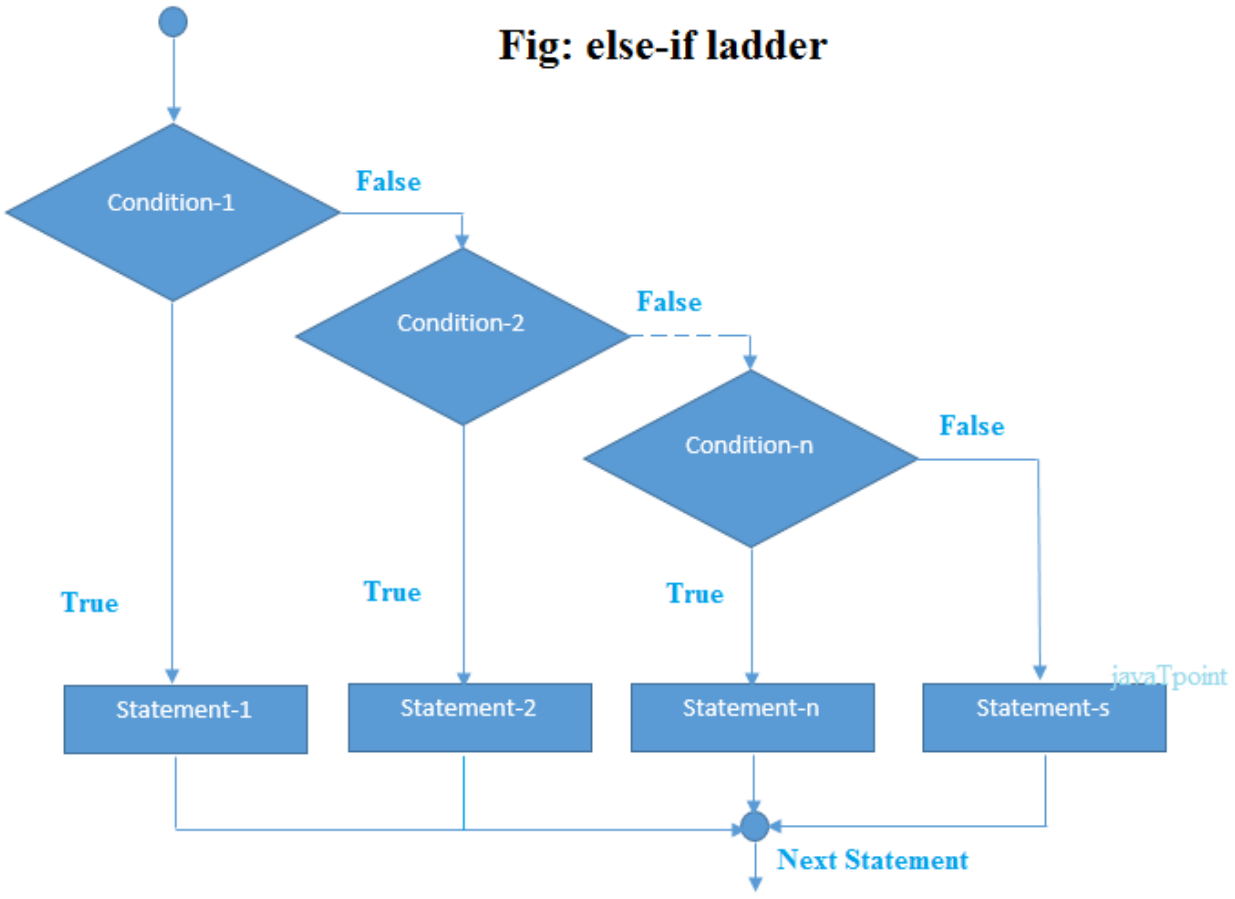


Fig: else-if ladder

```
1. public class IfElseIfExample {
2. public static void main(String[] args) {
```

```
3.    int marks=65;
4.
5.    if(marks<50){
6.       System.out.println("fail");
7.    }
8.    else if(marks>=50 && marks<60){
9.       System.out.println("D grade");
10.        }
11.        else if(marks>=60 && marks<70){
12.           System.out.println("C grade");
13.        }
14.        else if(marks>=70 && marks<80){
15.           System.out.println("B grade");
16.        }
17.        else if(marks>=80 && marks<90){
18.           System.out.println("A grade");
19.        }else if(marks>=90 && marks<100){
20.           System.out.println("A+ grade");
21.        }else{
22.           System.out.println("Invalid!");
23.        }
24.    }
25.    }
```

Ekranda :

```
C grade
```

Java Switch Statement

Javada **switch** operatori bir necha shartlarni tekshiradi .Huddi if-else-if ga o'xshaydi. Farqi u faqat bir marta tekshiradi. Shart to'g'ri bo'lgandan keyin switch operatoridan butunlay chiqib ketadi.

```
1. switch(shart){
```

```
2. case value1:
3.  //code to be executed;
4.  break;  //optional
5. case value2:
6.  //code to be executed;
7.  break;  //optional
8. ......
9.
10.      default:
11.       hech qaysi shartni qanoantlantirmasa;
12.      }
```

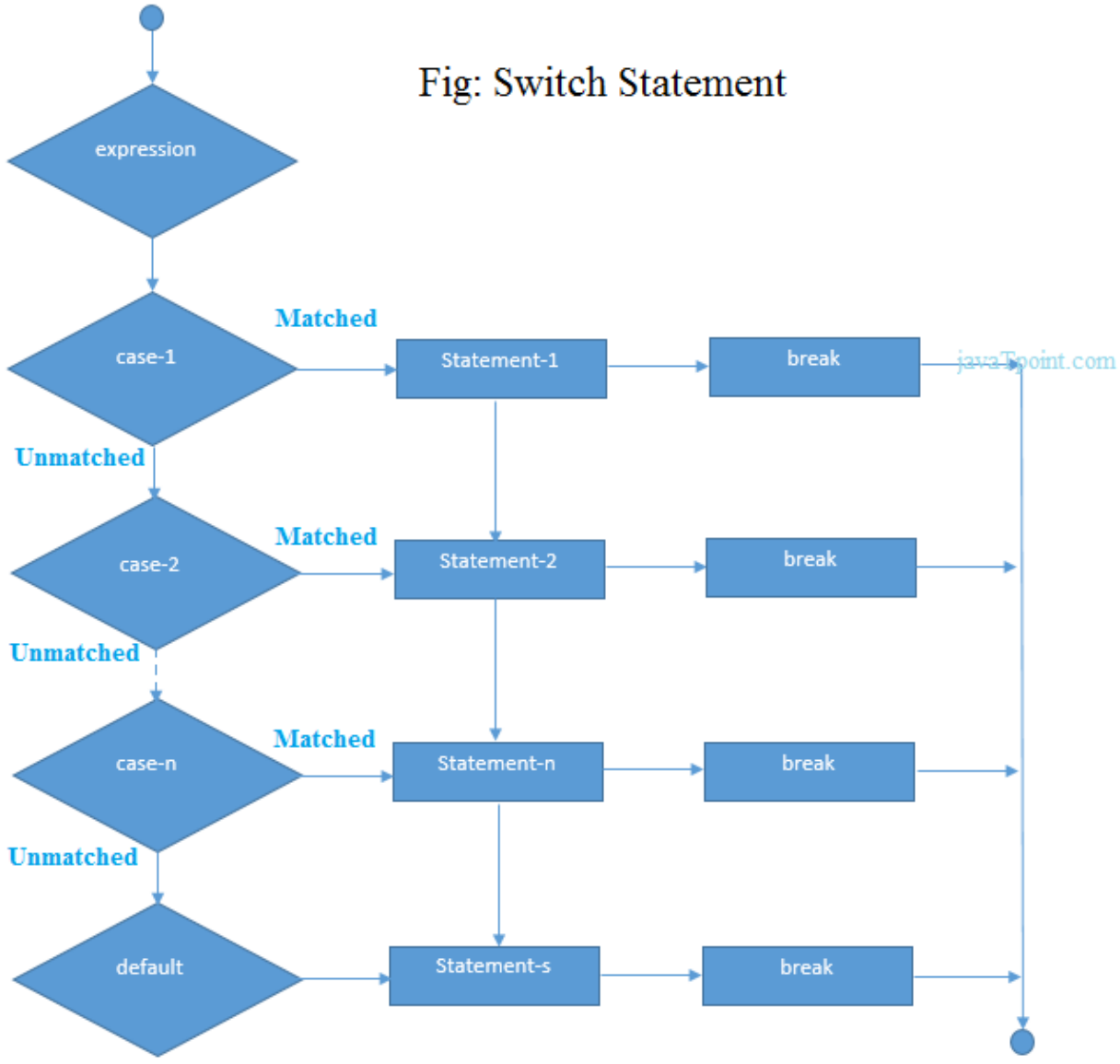


Fig: Switch Statement

```
1. public class SwitchExample {
2. public static void main(String[] args) {
3.     int number=20;
4.     switch(number){
5.     case 10: System.out.println("10");break;
6.     case 20: System.out.println("20");break;
7.     case 30: System.out.println("30");break;
8.     default:System.out.println("Not in 10, 20 or 30");
9.     }
10.       }
11.       }
```

Ekranda:
20

```
public class MainClass {


  public static void main(String[] args) {
    char yesNo = 'N';


    switch(yesNo) {
      case 'n': case 'N':
           System.out.println("No selected");
           break;
      case 'y': case 'Y':
           System.out.println("Yes selected");
           break;
    }
  }
}
```

Ekranda :

```
No selected
```

Javada For Loop(takrollanuvchi sikl operatoiri)

Takrollanuchi sikl operatorlari - O'z nomi bilan ma'lum bo'lib turubdikiy takronlanuchi yani qandaydur jarayonni qayta va qayta takronlanishidur. Yerning quyosh atrofida aylanishi yil fasillarning almashinishi va.h.k misol qilib keltirish mumkin. Takronlanuvchi ish harakatlar qandaydur shartlar asosida bajariladi. Ularning boshlang'ich, oxirgi nuqtalari va bajarilish davriligi mavjud bo'ladi. Misol uchun yerning quyosh atrofida aylanishini olsak boshlang'ich nuqta 1 yani 1-yanvar oxirgi nutqra 365 bu esa 31-dekabr takrollanishi 1 kunga teng. Atrofimizda bunga o'xshash takronlanuchi ish harakatlar shunchalik ko'pkiy ularni sanab tugata olmaymiz.
Misol tariqasida kundalik hayotimizda doimiy sodir bo'ladigan bir jarayoni tahlil qilib uning java da dasturiy ko'rinishi qanday bo'lishini ko'rib chiqsak.
Misol:
Tasavur qiling ovqat tayyorlash uchun 10 ta kartoshkani tozalashga oldik. Qo'limizda 10 ta kartoshka bor ularni tozalash uchun birinchi kartoshkadan boshlab tokiy 10-kartoshka tozalanmaguncha xar bir kartoshkani birma bir tozalab chiqamiz. O'z o'zidan ko'rinib turubdiky kartoshka tozalanish jarayoni ketma ket sodir bo'lyapti(sikl)

- Tozalangan kartoshkalar o'sib borish qiymati 1 ga teng(chunkiy ikki yokiy undan oshiq kartoshkani birdaniga tozalab bo'lmaydi :)))))))
- Boshlang'ich qiymati ham 1 ga teng
- Agar kartoshkani tozalash jarayonida kartoshkaning buzulgan, aynigan joylari bo'lsa kesib tashlanadi yani miyyamiz shu shartlar bilan shart tekshiradi.
- Bu jarayonlarni kartoshkalar soni 10 ga teng bo'lguncha amalga oshiramiz.

Keeling endi bu jarayonlarni java da dasturini tuzib chiqamiz.

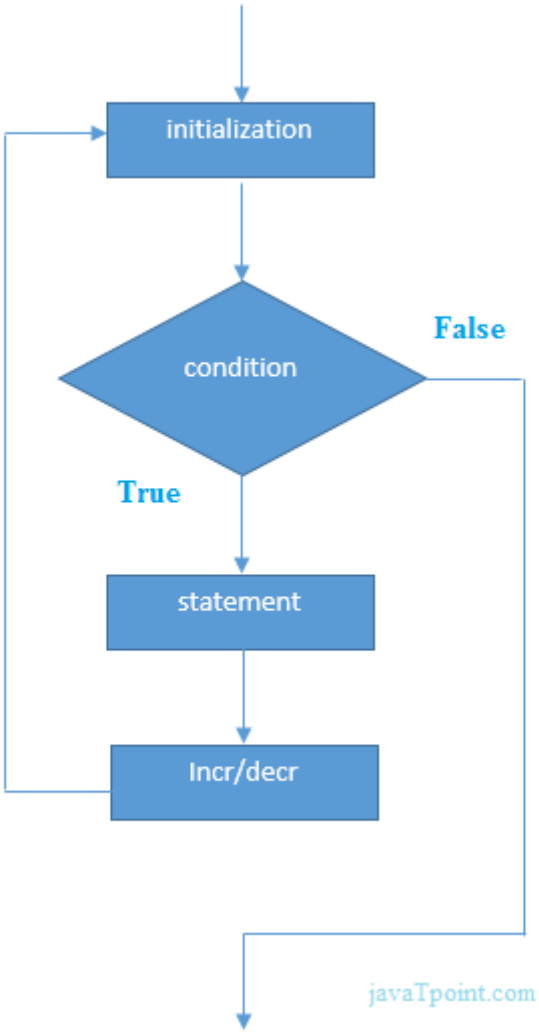
```
public class Tozalash {
public static void main(String[] args) {
   Scanner in = new Scanner(System.in);
      int aynigan = 1; // shartli ravishda shart tekshirish uchun shu qiymatlar kiritildi
      int buzulgan = 2; // shartli ravishda shart tekshirish uchun shu qiymatlar kiritildi
      for (int i = 1; i <= 10; i++) {
         System.out.println(i+"-kartoshkani Holatini kiriting \n yaxshi bo'lsa 0 ni \n aynigan bo'lsa 1 mi \n buzulgan bo'lsa 2 ni   ");
         int kartoshkaHolati = in.nextInt(); //kartoshkani holatini ko'rsatuvchi qiymatlar
```

```
            if (kartoshkaHolati == aynigan || kartoshkaHolati == buzulgan) {

                System.out.println("Buzulgan joyi kesib tashlansin ");
            }
             System.out.println(i +"-kartoshka tozalandi \n");
        }
        System.out.println("10 ta kartoshka tozalandi");
    }

}
```

Ekranda :

```
1-kartoshkani Holatini kiriting
 yaxshi bo'lsa 0 ni
 aynigan bo'lsa 1 mi
 buzulgan bo'lsa 2 ni

1 // krivchi qiymat
Buzulgan joyi kesib tashlansin
```

Bu jarayon 10 ta kartoshka tozalanib bo’lmaguncha davom etadi.
Tushunarliroq bo’lishi uchun blok sxema orqali ko’rib chiqamiz



Sikil boshlangandan shart tokiy false(yolg'on) bo'lmaguncha davom etadi. Shart false bo'lgan sikil tugaydi.

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

```
1. public class ForExample {
2. public static void main(String[] args) {
3.    for(int i=1;i<=10;i++){
4.       System.out.print(i+ ",");
5.    }
6. }
7. }
```

Ekaranda :

```
1,2,3,4,5,6,7,8,9,10
```

Sikilning boshlang'ich qiymati 1 ga ten takrollanishi ham 1 ga teng i ning qiymati o'sib borib tokiy qiymat 10 ga teng bo'lmaguncha davom etadi.
Mustahkamroq bo'lishi uchun yana bitta misol ko'rsak.

Berilgan sondan shun songacha bo’lgan raqamlar orasida 3 va 5 ga bo’linadigan raqamlar sonini topish

```
public class RaqamlarSoni {
  public static void main(String[] args) {
        int k = 0;
        for (int i = 1; i < 50; i++) {
            if (i % 3 == 0 || i % 5 == 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 chunkiy bunda sikilning boshlang’icha va oxirgi qiymatlari ko’rsatilmaydi. Bunda elementlar birma bir qaytarib uning qiymatlari ko’rsatiladi .

```
1. for(Type var:array){
2. //code to be executed
3. }
```

```
1. public class ForEachExample {
2. public static void main(String[] args) {
3.     int arr[]={12,23,44,56,78};
4.     for(int i:arr){
5.         System.out.print(i+”,”);
6.     }
7. }
8. }
```

Ekranda :

```
12,23,44,56,78
```

Yana biz **break/continue** lardan ham foydalanamiz

«Break» operatori.

Bu operator bilan, «**switch**» haqidagi maqoladan uchrashgan 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. Keling ikkala holni ham ko‘rib chiqaylik, dastlab bu operator orqali qanday qilib sikldan chiqish mumkinligini ko‘rsataman.

Oldingi maqolalardan ma’lumki, 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 qilayotgan bo‘lsa ham.

```
class test{
public  static  void main(String args[]){
for(int i=0; i<100; i++){
 System.out.println("Son"+i);
 if (i==10) break;
}
System.out.println("Sikl tugadi");
}
}
```

Dasturdan ko‘rinib turibdiki, sikl yordamida **100** ta qiymat ekranga chiqarilishi lozim(**i<100**, 0 dan 99 gacha), lekin unday bo‘lmaydi, chunki «**break**» operatori mavjud(5-qatorda). Bu operator «**if**» operatori bilan birga kelgan. Demak, shart qo‘yilgan, agar «**i**» o‘zgaruvchi qiymati «**10**» ga teng bo‘lsa, «**break**» operatorini ishlat, ya’ni siklni tugat.

«Continue» operatori.

Bu operator, ishlayotgan sikl qadamini tashlab ketib, navbatdagi qiymat bilan siklni boshidan boshlab beradi. Sikl tanasida ishlatilgan «**continue**» operatori, o‘zidan keyin kelgan operatorlarni ishlatmaydi.

Tup sonlarni chiqarib beruvhi dastur

```
public class MainClass {
  public static void main(String[] args) {
    int nValues = 50;


    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+",");
    }
  }
}
```

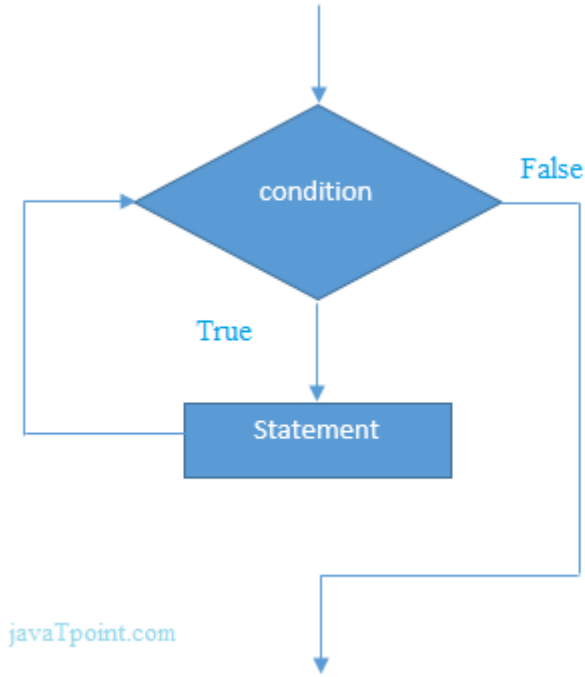
Natija

```
2,3,5,7,11,13,17,19,23,29,31,37,41,43,47
```

Shartga ko'ra I ni j ga bo'lganimizda qoldiq 0 ga teng bo'lsa **`continue ni`** amalni to'xtatib siklning boshiga qaytatib yuboradi.

Java While Loop

«**While**» operatori dastur tuzishda ko'p ishlitiladigan 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.



Sintaksisi 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 bajarilardi

«**While**» operatoridan raqamlar ketma ketligi doimiy bo’lmagan hollarda ishlatiladi.
While operatori takrollanish davri xar xil bo’lgan sikllar ustida ammalar bajarishda qo’l keladi. Tasavur qiling siz bozorda 3kg pomidor harid qilyapsiz, sotuvchi tarozida turgan idishga pomidor solyapti bir safar ikkita bir safar 3 ta qo’liga siqqanicha olib idishga solyapti.Sotuvchining idishga pomidor solishi doimiy takrollanyapti lekin miqdori xar safar xar xil tokiy bu jarayon pomidorlar 3 kg bo’lgunicha sodir bo’ladi. Endi bu jarayonni tahlil qilib chiqsak

- Idishdagi boshlang’ich pomidor miqdori 0 kg
- Chegaraviy miqdor 3 kg
- Ortib boorish miqdori xar safar xar xil

Sotuvchining miyyasidagi jarayonlarni dasturiy ko’rinishini qanday bo`lishini ko’rsak

1. Sotuvchi xaridordan necha kg pomidor olishini so`rab chegaraviy miqdorni aniqlaydi
2. Idishni taroziga qo`yib boshlang`ich miqdor (massa=0) 0 deb oladi
3. Idishga pomidor solishi bilan miqdorni hissoblab boradi.
4. Agar massa 3 kg dan oshib ketsa ortiqcha pomidorlarni olib tashlaydi

Bu misolning javada dasturiy ko’rinishi:

```
public class Xarid {

    public static void main(String[] args) {
        Scanner in  = new Scanner(System.in);

      int massa = 0;// boshlang'ich miqdor
      int chegaraviyMiqdor = 3;
        while (massa < chegaraviyMiqdor) {
            System.out.println("Pomidor miqdorini kiriting");
            int m = in.nextInt();
            massa +=m;
        }
        if (massa > chegaraviyMiqdor)
            System.out.println(massa - chegaraviyMiqdor +" kg oshib ketdi");
        else
        System.out.println("Pomidor massasi 3 kg bo'ldi");

    }

}
```

Ekranda:

Pomidor miqdorini kiriting
2 // kiruvchi qiymatlar
Pomidor miqdorini kiriting
1 // kiruvchi qiymatlar
Pomidor massasi 3 kg bo'ldi

Yana bitta oddiy misol 1 dan 10 gaxcha bo'lgan sonlarni chiqarish dasturi

```
1. public class WhileExample {
2. public static void main(String[] args) {
3.     int i=1;
4.     while(i<=10){
5.         System.out.print(i+",);
6.     i++;
7.     }
8. }
9. }
```

Natija

```
1,2,3,4,5,6,7,8,9,10
```

«**int**» tipida «**i**» nomli o'zgaruvchi e'lon qilinib, «**1**» soni o'zlashtirildi. So'ng sikl boshlandi, bu sikl kiritilgan i(**1**), «**10**» sonidan katta bo'lguncha bajariladi. Sikl tanasida, «**i**» o'zgaruvchi qiymati ekranga chiqarilib, bu o'zgaruvchi bittaga oshirilmoqda(qiymat 2 ga teng bo'ladi). Shundan so'ng, kompilyator yana sikl boshiga chiqib, shartni tekshiradi(**2<10** rost). Agar, shart rost(true) bo'lsa, sikl yana ishini davom ettiradi, endi ekranga «**2**» soni chiqadi va yana «**i**» o'zgaruvchi qiymati bittaga oshiriladi(o'zgaruvchi qiymati 3). Shu tariqa shart yolg'on bo'lguncha sikl aylanaveradi.
Qachonki, «**i**» o'zgaruvchi qiymati «**10**«ga teng bo'lsa, shart yolg'on qiymatni qaytaradi(**10<10** yolg'on) va sikl o'z ishini tugatadi.

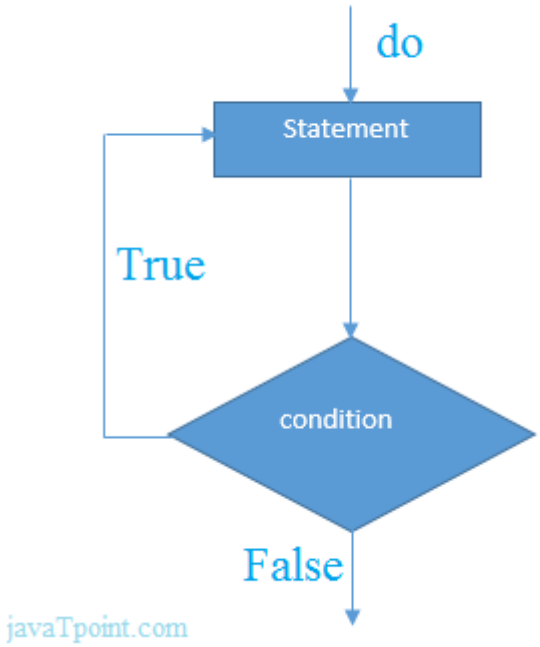
Agar shart boshidanoq yolg'on bo'lsa, sikl umuman ishlamasligi ham mumkin, ya'ni kompilyator sikl tanasiga umuman kiraolmaydi.

```
class test{
public static void main(String[] args){
    int d=10;
    while (d<10){
        System.out.println(d);
        d++;
    }
}
}
```

Bu misolda sikl umuman ishlamaydi, chunki (**10<10**) shart yolg‘on(10=10 bo‘lsa, rost bo‘lar edi) va dastur hech qanday amal bajarmaydi.

Java do-while Loop

Yuqoridagi «**while**» sikl operatorida, agar shart yolg‘on bo‘lsa sikl umuman ishlamasligini ko‘rib o‘tdik. Agar shart yolg‘on bo‘lsa ham, sikl tanasidagi operatorlar bir marotaba bajarilishi kerak bo‘lsa, «**do-while**» sikl operatoridan foydalanamiz. Bu operatorda oldin operatorlar bajariladi, so‘ng siklga qo‘yilgan shart tekshiriladi. Shu sababli sikl tanasi kamida bir marotaba ishlaydi. Bunday siklni «**sharti keyin tekshiriladigan**» sikl deyish mumkin.



«**do-while**» dan ham odatda ketma ketlik tartibi doimiy bo’lmagan va bir marta bo’lsa ham amal bajarilishi kerak bo’lgan hollarda foydalaniladi.

Misol uchun choy damlash ketma ketligini olsak

1. Quruq choy solinadi
2. Choynak to’lguncha suv quyiladi

Choy damlashdan oldin eng avvalo bitta ishni qilamiz yani choynak ichida qanday holat choy bormi yo’qligiga bir nazar tashlaymiz.

```
public class MainClass {
  public static void main(String[] args) {
    int limit = 20;
    int sum = 0;
    int i = 1;

    do {
      sum += i;
      i++;
    } while (i <= limit);

    System.out.println("sum = " + sum);
  }
}
```

Natija

```
sum = 210
```

Java Break Statement

Bu operator bilan, «**switch**» haqidagi maqoladan uchrashgan 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. Keling ikkala holni ham ko‘rib chiqaylik, dastlab bu operator orqali qanday qilib sikldan chiqish mumkinligini ko‘rsataman.

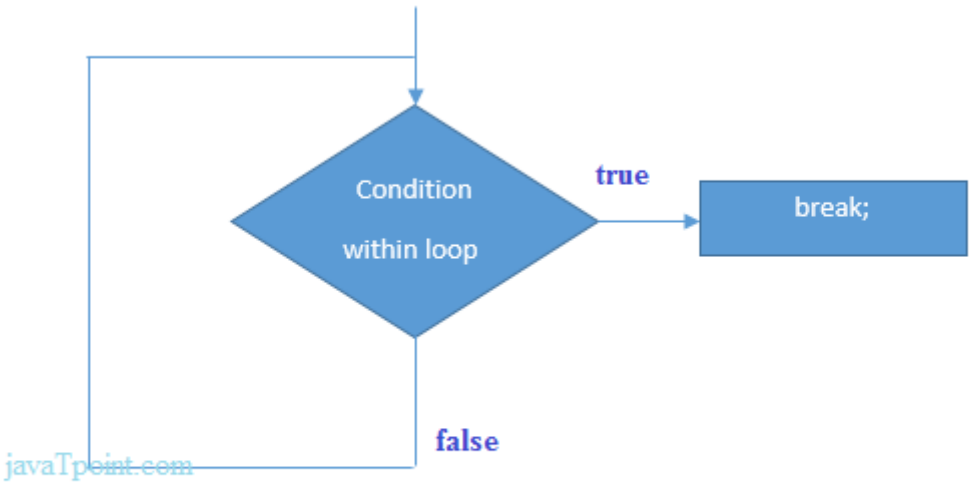


Figure: Flowchart of break statement

```
1. public class BreakExample {
2. public static void main(String[] args) {
3.     for(int i=1;i<=10;i++){
4.         if(i==5){
5.             break;
6.         }
7.         System.out.println(i);
8.     }
9. }
10. }
```

Natija:

```
1

2

3

4
```

Bu misolda boshlang'ich qiymati 1 ga oxirgi qiymati 10 ga teng bo'lgan for sikl operatori berilagan. Bunda i==5 bo'lganda **braek** yani sikl to'xatatilsin deb shart qo'yildi va sikilda buntunlay chiqib natija 1 2 3 4 lar chiqdi.

Java Continue Statement

Bu operator, ishlayotgan sikl qadamini tashlab ketib, navbatdagi qiymat bilan siklni boshidan boshlab beradi. Sikl tanasida ishlatilgan «**continue**» operatori, o‘zidan keyin kelgan operatorlarni ishlatmaydi.

```
1. public class ContinueExample {
2. public static void main(String[] args) {
3.     for(int i=1;i<=10;i++){
4.         if(i==5){
5.             continue;
6.         }
7.         System.out.println(i);
8.     }
9. }
10.      }
```

Narija

```
1
2
3
4
6
7
8
9
10
```

Bu misolda shartga binoan i==5 bo’lganida sikini to’tatib siklning boshiga qaytarib yuboradi natijada System.out.println(i); amali bajarilmaydi. Shuning uchun ekrada 5 chop etilmaydi.

```
public class MainClass {


  public static void main(String[] arg) {
    int limit = 10;
    int sum = 0;
    for (int i = 1; i <= limit; i++) {
```

```
      if (i % 3 == 0) {
        continue;
      }
      sum += i;
    }
    System.out.println(sum);
  }

}
```

Natija

```
37
```

Faktarila

```
public class MainClass {
  public static void main(String[] args) {
    int limit = 20;
    int factorial = 1;

    OuterLoop: for (int i = 1; i <= limit; i++) {
      factorial = 1;
      for (int j = 2; j <= i; j++) {
        if (i > 10 && i % 2 == 1) {
          continue OuterLoop;
        }
        factorial *= j;
      }
      System.out.println(i + "! is " + factorial);
    }
  }
}
```

Natija

```
1! is 1
2! is 2
3! is 6
4! is 24
```

```
5! is 120
6! is 720
7! is 5040
8! is 40320
9! is 362880
10! is 3628800
12! is 479001600
14! is 1278945280
16! is 2004189184
18! is -898433024
20! is -2102132736
```

Javada bazibir masalalar ishlanmasi

Fibonachi

```
1. class FibonacciExample1{
2. public static void main(String args[])
3. {
4.  int n1=0,n2=1,n3,i,count=10;
5.  System.out.print(n1+" "+n2);//printing 0 and 1
6.
7.  for(i=2;i<count;++i)//loop starts from 2 because 0 and 1 are
    already printed
8.  {
9.   n3=n1+n2;
10.       System.out.print(" "+n3);
11.       n1=n2;
12.       n2=n3;
13.      }
14.
15.     }}
```

Natija

```
0 1 1 2 3 5 8 13 21 34
```

Fibonachi sonlarini rekursiya orqali toppish

```
class FibonacciExample2{
 static int n1=0,n2=1,n3=0;
 static void printFibonacci(int count){
    if(count>0){
        n3 = n1 + n2;
        n1 = n2;
        n2 = n3;
        System.out.print(" "+n3);
        printFibonacci(count-1);
    }
 }
 public static void main(String args[]){
  int count=10;
  System.out.print(n1+" "+n2);// 0 va  1 ni chiqaradi
  printFibonacci(count-2);//n-2 chunki 2 ta raqam oldindan chop etilgan
 }
}
```

Berilagn sonni tup yokiy tup emasligini aniqlovchi dastur

```
class PrimeExample{
 public static void main(String args[]){
  int i,m=0,flag=0;
  int n=17;//it is the number to be checked
  m=n/2;
  for(i=2;i<=m;i++){
   if(n%i==0){
   System.out.println("raqam tup emas");
   flag=1;
   break;
   }
  }
  if(flag==0)
  System.out.println("raqam tup");
 }
 }
```

Armstong raqami

```
49.     class ArmstrongExample{
50.       public static void main(String[] args)  {
51.         int c=0,a,temp;
52.         int n=153;    temp=n;
53.         while(n>0)
54.         {
55.         a=n%10;
56.         n=n/10;
57.         c=c+(a*a*a);
58.         }
59.         if(temp==c)
60.         System.out.println("armstrong raqimi");
61.         else
62.             System.out.println(" armstrong raqami emas");
63.        }
64. }
```

Natija

```
armstrong raqami
```

Javada massivlar

Massiv bu — bir turdagi o'zgaruvchilarni o'zida saqlovchi qandaydur nom bilan nomlangan o'lachami chegaralangan gruh yani oddoyoq qilib aytganda maxsus idishdur. Bitta yokiy ikkita o'zgaruvcgu ustuda qandaydur ammalarni bajarmoqchisiz bununig uchun bir yokiy ikkita o'zgaruchi yaratib olib hahlagan ammalarni bajarish mumkin lekin bu o'zgaruvcilar ko'p bo'lsachi? Agar har bir ishlatmoqchi bo'gan o'zgaruvchi uchun har safar yangitan o'zgaruvchilarni yarataversak bu bir muncha noqulayliklar tug'duradi. Bu vaziyatda esa massivlarni ishlatish qo'l keladi.

Oddiy tushunarli bo'lishi uchun har doimgidek hayotiy bitta misol keltirib o`taman. Tasavur qiling siz do'stingizga bitta yoki ikkita olma bermoqchisiz. Siz bu ishni to'g'ridan to'g'ri qilishingiz mumkun chunkiy ikkita olma qo'lingizga bemalol sig'adi. Agar bu olmalar soni ko'p bo'lsachi bunday vaziyatda nima qilasiz. Albatda bu vaziyatda bermoqchi bo'lgan olmalarizni miqdoriga qarab o'sha miqdorni ko'tarish qoblyatiga ega

bo’lgan idishga solib berasiz. Massiv ham huddu shunday vaziyatlarda o’zgaruvchilar uchun idish vazifasini bajarib beradi. Tasavur qilling hamma olmalarni hechqanday ishishga solmay qo’lingizda ko’tatib ketyapsiz. Keling endi bu vaziyatda qanday noqulayliklar tug’ulishini ko’rib chiqaylik

65. Ko’tarishga juda noqulay
66. O’zimiz mustaqil holda hamma olmalarni ko’tara olmaymiz chunkiy ikkala qo’limiz ham band bo’ladi .
67. Yo’lda ketayotgan paytimizda yaqin tanishimiz chiqib qoldi unga olmalarning eng kattasini bermoqchi bo’lsangiz qo’lingizdagi olmalar orasidan eng katasini tanlab berishingiz juda qiyin bo’ladi.

Bunday noqulayliklarni juda ko’p keltirib o’tishimiz mumkun. Olmalarimizni hammasi idishda bo’lasachi? Bunday noqulayliklar aslo kelib chiqmaydi. Bunday vaziyatlarni bir ikki harakat bilan osongina hal qilish mumkun bo’ladi.

Massivlarning bizga qanday imkoniyatlarni beradi :

Optimal kodlar:kodlarni optimal darajaga keltirishimiz , qayta yozish yokiy saralashlarni juda oson amalga oshirishimiz mumkin

Ixtiyoriy kirish: har qanday pozitsiyada turgan indexdagi qiymatni olish imkoniyati yani qidirish

Kamchiligi: Massivning o’lchami oldindan beriladi shuning uchun uning o’lchami statikdur. Hohlagancha qiymat qabul qila olmaydi. Bunday holatlarda to’plamlardan foydalaniladi(collection)

Yuqorida keltirgan holatlar uchun bitta misol keltiraylik

```
class test{
public static void main(String args[]) {
int jan=21;
int feb=26;
int mar=32;
int apr=24;
int may=28;
int jun=43;
int jul=35;
int aug=29;
int sep=23;
int oct=19;
```

```
int nov=21;
int dec=11;
float rez;
rez = (jan+feb+mar+apr+may+jun+jul+aug+sep+oct+nov+dec)/12;
System.out.println(rez);
}
}
```

Dasturda, 12 ta bir xil tipli o‘zgaruvchi e’lon qilinmoqda va ularga qiymatlar berilgan. So‘ng, bu qiymatlar qo‘shilib, «**rez**» nomli o‘zgaruvchiga yozildi. Shundan so‘ng, bu o‘zgaruvchi qiymati ekranga chiqarilmoqda. Natija esa, **26.0** ga teng.

Yuqoridagi misolni massiv orrqali ifodalaymiz

```
class test{
public static void main(String args[]) {
int temp_mes[]= {21,26,32,24,28,43,35,29,23,19,21,11};
float rez = 0;
for(int i = 0; i < 12; i++){
  rez += tem_mes[i];
}
System.out.println (rez/12);
}
}
```

Ko’rib tuganingizdek kodlar anchq qisqargan va optimallshgan.

Bu misolning yechimini qadamma qadam ko’rib chiqsak

Binchi qadam: **rez = 0 , i=0** bo’lganida **temp_mas[0] = 21** ga teng bo’ladi natijada **rez = 0 + 21 , rez = 21**

Ikkinchi qadam : **rez = 21 , i=1, temp_mas[1] = 26 , rez = 21 + 26**

Uchinchi qadam: **rez = 47 , i=2, temp_mas[2] = 32 , rez = 47 + 32**

……..

o’nikkinchi qadam: **rez = 301 , i=11, temp_mas[11] = 11 , rez = 301 + 11**

natija **rez = 312** va sikl qadami tugaydi.

Massiv tuzulishi

Massivlar 2 xil turda bo‘lishi mumkin: **bir o‘lchamli** va **ko‘p o‘lchamli**.
Bir o‘lchamli massivlar bitta tipdagi o‘zgaruvchilar ro‘yxatidan iborat bo‘ladi. Bir o‘lchamli massivni e’lon qilish uchun, dastlab, massiv tipi, so‘ng massiv nomi ko‘rsatiladi(undan so‘ng qavslar).

```
int a[]=new int[5];//massivni e’lon qilish
a[0]=10;
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
```

bu yerda uzunligi 5 ga teng bo’lgan massiv e’lon qilinyapti va va ularning indexiga mos ravishda qiymatlar o’zlashtirilyapti.

```
68. class Testarray{
69. public static void main(String args[]){
70.
71. int a[]=new int[5];//massiv e’lon qilinyapti
72. a[0]=10;
73. a[1]=20;
74. a[2]=70;
75. a[3]=40;
76. a[4]=50;
77.
78. //massivni ekrada chiqarish
79. for(int i=0;i<a.length;i++)//length massivning uzunligi
80. System.out.println(a[i]);
81.
82. }}
```

Natija :

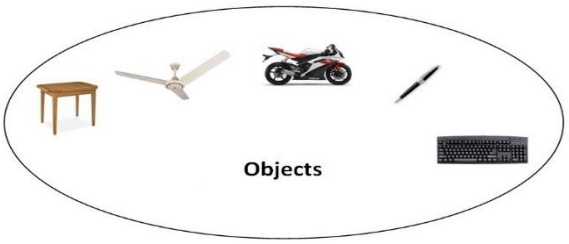
```
10
20
70
40
50
```

Massiv elementlari ichida eng kichigin aniqlash dasturi

```
83.class Testarray2{
84.public static void main(String args[]){
85.
86.int a[]={33,3,4,5};
87.int min=a[0];
88.for(int i=1;i<a.length;i++)
89. if(min>a[i])
90.  min=a[i];
91.
92.System.out.println(min);
93.
94.}}
95.Natija:
96.3
```

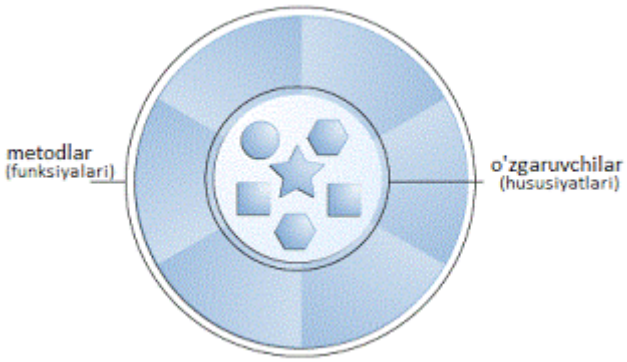
***Obyekt* nima?**

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



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

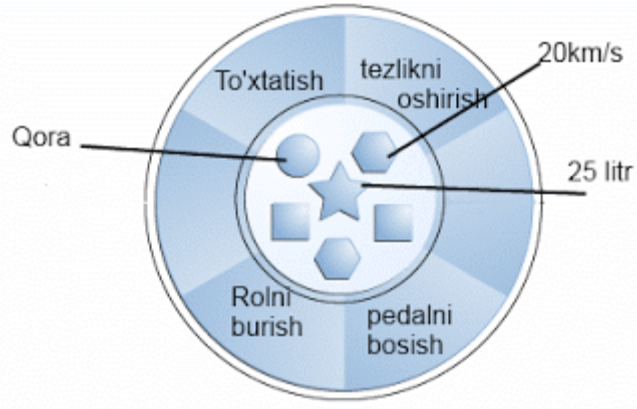
Bir daqiqaga to'xtang va hozirda atrofingizdagi biror narsalarni analiz qiling. Har bir obyekt uchun o'zingizdan so'rang: “Bu obyektning qanday hususiyatlari bor?” , “Qanday vazifalarni bajaradi?” kabi. Va kuzatish natijalaringizni yozib oling, sezgan bo'lsangiz tuziladigan ro'yxat obyektning murakkabligiga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



Dasturlashdagi obyekt.

Dasturlashdagi obyekt(bundan keyin oddiygina *obyekt* deb ketiladi) ham haqiqiy hayotdagi obyektlarga o'xshash: Ular ham qandaydir hususiyatlar va bajaradigan funktsiyalardan iborat bo'ladi. Obyektning hususiyatlari har xil dasturiy *o'zgaruvchilardan* iborat bo'ladi va ularning o'zgartirish uchun qandaydir *funktsiyalar* bajariladi. Bunday *funktsiyalar* bilan o'zgaruvchilarning holatini berkitish mumkin ya'ni aynan o'sha o'zgaruvchini tashqaridan o'zgartirish uchun albatta maxsus funktsiyadan foydalanish kerak bo'ladi. Bu jarayon “**Enkapsulatsiya**” deb atalib, OYDning eng muxim tushunchalaradian biridir.

Mashinani tasavvur qiling,



Uni dasturlash obyekit sifatida modellashtiramiz:

Uning o'zgaruvchilari(hozirgi tezligi, qolgan benzini, va h.k) va uning funktsiyalari(to'xtatish, tezlikni oshirish, rulni burish, va h.k.). Bu yerda uning bakidagi benzini yurishi tufayli kamayib boradi demak uning qiymatining o'zgarishi 0dan bakning sig'imigacha bo'ladi, yoki uning tezligi ham shu kabi aynan qaysidir funktsiyalarning 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 turiptiki 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 hatto minglab qatorli kod bo'lishi mumkin. Undan biror narsani topib-o'zgartrish ancha mashaqqat bo'ladi.

Qayta foydalanish: Yana boshqa plyus tarafi biz bo'laklagan mashinaning detallarini boshqa obyektlarda ham ishlatishimiz mumkin. Masalan, 2 xil mashina ularning shunday qismlari borki aynan bir xil, ana o'shalar uchun ikki marta alohida kod yozmasdan, bitta yozganimizni qayta ishlatishimiz mumkin.

Uzib-ulanuvchanligi: buni tushunish uchun yuqoridagi misoldan foydlanamiz, aytaylik, mashinaning biror qismi ishlamayapti, xo'sh nima qilinadi? yoki ishlab turgan boshqasiga almashtiramiz, yoki tuzatamiz. Mashinaning biror bolti buzilsa uni boshqa ishlab turgani bilan almashtirasiz yoki tuzatamiz lekin mashinani almashtirmaymiz.

OYDning asosiy tushunchalari

Obyekga yo'naltirilgan dasturlash yoki *OYD* – dasturlarni haqiqiy hayotiylikka asoslangan holda gi dasturlash usulidir. Yana protsedurali dasturlash tillari (masalan, Paskal, Basic, Fortan) ham mavjud. OYD ning undan asosiy farqi shundaki, OYD asosan obyektlar asoslangan holda ishlasa, *protsedurali dasturlash*

tillari esa asosan funktsiyalarga 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 quydagilardur

97. Object
98. Class
99. Inheritance
100. Polymorphism
101. Abstraction
102. Encapsulation

Object – bu class bilan farqli tushuncha xisoblanadi. Object biz yozgan classimizdagi har xil qoidalarga bo'ysunadigan ma'lumot bo'lib, u tezkor hotirada saqlanadi, class esa qattiq diskda saqlanadi. Har bir yasalgan Object tezkor hotiraning ma'lum bir honachalariga joylashadi. Hayotiy bir misol, masalan, ko'p qavatli binoni tezkor hotira deb qarasak. Unda istiqomat qiluvchi insonlar esa unda saqlanuvchi obyektlar bo'lib. Agar biz Insonning hususiyatlari, bajaradigan ishlari va hokazo hususiyatlari haqidagi bilimlarni qog'ozga tushursak bu qoralamani class deb qaralishi mumkin garchi u texnik usulda yozilmagan bo'lsa ham. Biz ana o'sha qoralamani classimizda kompyuter tushunadigan tilga keltiramiz.

Class- OYDning marzkazi hisoblanadi va u har xil kodlar, ma'lumotlar va shu ma'lumotlar qay tarzda o'zgarishini ifodalovchi hususiyatlar saqlanadi.Boshqacharoq qilib aytadigan bo'lsak hayotiy obyektlarning qanday faoliyat yuritishi, nimalardan iborat ekanligi, qanday hususiyatlarga ega ekanligini tavsiflovchi kichik bir hujjat sifatida qarash ham mumkin. Javada hamma narsa Class ichida sodir bo'ladi. Class o'z ichiga o'zgaruvchilar va metodlar(funktsiyalar) va qiymati o'zgarmaydigan konstantalarni oladi. Yana shuni ham ta'kidlash kerakki, har bitta klass bitta o'zgaruchi tipi bo'lib ham hizmat qiladi. Xuddi Integer, String yoki boshqa tiplar kabi har bir class ham ma'lum bir tip sifatida qaralishi mumkin.

Javada class lar quydagilardan tashkil topadi

103. **data member**
104. **method**
105. **constructor**
106. **block**
107. **class and interface**

Javada o’zgaruvchilar

O'zgaruchilar – obyektlarimizni o'zgaruvchi sifatida ham qarashimiz mumkin bunda uning qiymatlari o'zgarishi mumkin. Masalan Inson classimizda 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” compilatsiya vaqtida xotiradan joy olmaydi.O’zgaruvchilarimizni obyekt sifatida qaraganimizda runtime paytida tezkor xotiradan joy oladi.

Javada method

Metodlar(Funktsiyalar) – 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.

Method larning imkoniyatlari

108.Kodni qayta ishlash

109.Kondni optimallashtirish

Obyekt va class ga misol

Bu misolda **student** classidan ikkita obyekt yaratildi va insertRecord funnksiyasi orqali boshlang'ich qiymatlarni o'zlashtirildi.
displayInformation funsiyasi dan foydalanib ekrada qimatlarni chop etiladi.

```
110.    class Student{
111.     int rollno;
112.     String name;
113.
114.     void insertRecord(int r, String n){  //method
115.      rollno=r;
116.      name=n;
117.     }
118.
119.     void displayInformation(){System.out.println(rollno+" "+name);}//metho
    d
120.
121.     public static void main(String args[]){
122.      Student s1=new Student();
123.      Student s2=new Student();
124.
125.      s1.insertRecord(111,"Karan");
126.      s2.insertRecord(222,"Aryan");
127.
128.      s1.displayInformation();
129.      s2.displayInformation();
130.
131.     }
132.    }
```

Ekranda : 111 Karan

222 Aryan

Inheritance(meros ho’rlik)

Ma'lum obyekt asosida boshqa obyektni yaratish jarayoniga aytiladi. Bunda obyektning barcha hususiyatlarini meros qilib oladi yani **private(shaxsiy)** bo’lmagan o’zgaruvchilari funnksiyalri konstantalari ni bemalol foydalanish. Inheritance dan runtime polymorphism da foydalaniladi

Polymorphism

OOPning uchinchi ustuni. Yuqorida berilgan ustunlar bilan doim birga yuradi. Ma'lum obyekt asosida boshqa obyektni yaratish jarayoniga aytiladi. Bir classning boshqa classdan 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 classda bo'lgan 4 g'ildirak farzand classlarda ham mavjud bo'ladi. Ya'ni poyga mashinasi, avtoulovdan g'ildiraklarni meros oladi

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 telifon jiringlasa qolsa telifonga qo’ng’iroq bo’layotganini bilamiz lekin uning ichida nima jarayon bo’layotganini bilmaymiz.

Encapsulation

Ma'lumotlar va funksiyalarni bir komponent ichiga yig'ishga atyiladi. Buning uchun classlardan foydalaniladi. Enkapsulatsiya tanlov asosida classning ba'zi hususiyatlarini foydalanuvchidan yashirish imkonini beradi. Bunga misol qilib ustidan maxsus modda bilan o’ralgan dorilar

Javada static kalit so’zi (keyword)

Static kalit so’zidan asosan xotirani boshqarishda foydalaniladi. Biz **static** kalit so’zidan foydalanib o’zgaruvchilar, methodlar, blocklar va nested class lar yarata olamiz.

Javada static o’zgaruvchilar

Agar siz o’zgaruvchini **static** deb e’lon qilgan bo’lsangiz unda bu o’zgaruvchi **satatic o’zgaruvchi** bo’ladi.

Static o’zgaruvchilar hamma oyektlar uchun umumiy bo’ladi. Misol uchun firma nomi barcha hodimlar uchun umumi, unverset nomi barcha talabalar uchun umumiy bo’ladi.

Static o’zgaruvchilar xotiradan bir marta joy oladi.

Misol uchun :

```
class Student{
    int id;
    String name;
    String college="TIU";
}
```

Student nomli class yaratdik va unda id, nome, va collage o’zgaruvchilari bor.Tasavur qiling unversetda 500 ta talaba o’qiydi xar bir talaba uchun dastur xotiradan joy ajratadi, id va name takronlammas bo'ladi va bu yaxshi holat lekin 500 talaba uchun unvetet nomi bir xil va xar safar unverset nomi uchun xotiradan joy ajratish yaxshi emas.Bu holatda unverset nomi uchun bir marotaba xotiradan foydalanish uchun **satatic kalit** so’zidan foydalanamiz.

Misol:

```
class Student8{
          int id;
          String name;
          static String college ="TIU";

          Student8(int r,String n){
          id = r;
          name = n;
          }
        void display (){System.out.println(id+" "+name+" "+college);}

        public static void main(String args[]){
        Student8 s1 = new Student8(111,"Zafar");
        Student8 s2 = new Student8(222,"Aziz");

        s1.display();
        s2.display();
        }
         }
```

Ekranda :

```
111 Zafar TIU
222 Aziz TIU
```

Yana bitta misol ko’rsak

Bu misolda (instance) o’zgaruchisi bor uni counter deb nomlaymiz constructr yaratib o’zgaruvchini qiymatini oshirib natijani ko’ramiz.

```
class Counter{

int count=0;


Counter(){

count++;

System.out.println(count);

}


public static void main(String args[]){


Counter c1=new Counter();

Counter c2=new Counter();

Counter c3=new Counter();


 }

}
```

```
Ekranda:1
        1
        1
```

Natija chiqadi ko’rib turganimzdek har safar yangi obyekt olganimizda counter o’zgaruvchisiga xotiradan yangidan joy ajratyapti.

End shu o’zgaruvchini static deb e’lon qilib natijani ko’ramiz

```
class Counter2{

static int count=0;


Counter2(){

count++;

System.out.println(count);

}


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

```
Counter2 c1=new Counter2();
Counter2 c2=new Counter2();
Counter2 c3=new Counter2();


 }
}
Ekranda:1
        2
        3
```

static methodlar

Agar funnksiya static kalit so’zi bilan bo’lsa bu funksiya static funksiya deyiladi.

Javada Method Overloading

Class ichida nomi va turlari(data type) bir xil bo’lgan ikki yokiy undan ortiq bo’lgan methodlar bo’lasa bunday methodlar overloading methodlar deyiladi.

Overloading methodlar

133. Parametrlari bilan farq qiladi
134. Ma’lumot turi
135. Ma’lumot turlari ketma ketligi bilan

Bir biridan farq qiladi

Method overloading ni **static polymorphism** deb ham qarashimiz mumkun

Tariff:

136. **Static polymorphism** compile time dagi bog’lanish yokiy erta bog’lanish(binding) sifatida ham ma’lum
137. **Static bog’alanish(binding)** compile time da sodir bo’ladi. Method overloading static bog’lanishga(binding) misol bo’ladi.

Yuqorida keltirib o’tilgan tariflar barchasi **method overloading** ni argumetlari xar xil bo’lgan holarlar uchun mos keladi. Endi sizlar bilan har bir holatni birma bir ko’rib chiqamiz

138. Overloading parametiri va argumentlari bilan faq qilishga misol

Method nomlari o’xshash lekin argumetlari bilan farq qiladi

```
class DisplayOverloading
{
    public void disp(char c)
    {
         System.out.println(c);
    }
    public void disp(char c, int num)
    {
         System.out.println(c + " "+num);
    }
}
class Sample
{
   public static void main(String args[])
   {
       DisplayOverloading obj = new DisplayOverloading();
       obj.disp('a');
       obj.disp('a',10);
   }
}
```

Ekrada :

```
a
a 10
```

yuqoridagi misolda method **disp()** method overload asosida yaratilyapti. Bunda methodlar bir biridan argumetlar soni bilan farqlanyapti.

Overloading – ma’lumot turlari bilan farq qilishi

```
class DisplayOverloading2
{
    public void disp(char c)
    {
        System.out.println(c);
    }
    public void disp(int c)
    {
       System.out.println(c );
    }
}

class Sample2
{
    public static void main(String args[])
    {
        DisplayOverloading2 obj = new DisplayOverloading2();
        obj.disp('a');
        obj.disp(5);
    }
}
```

Ekranda:

```
a
```

```
5
```

Overloading- ma’lumot turlarining ketma ketligi bilan faqr qilishi.

```
class DisplayOverloading3
{
   public void disp(char c, int num)
   {
       System.out.println("Bu birinchi disp() methodi ");
   }
   public void disp(int num, char c)
   {
       System.out.println("Bu ikkinchi disp() methodi " );
   }
}
class Sample3
{
   public static void main(String args[])
   {
       DisplayOverloading3 obj = new DisplayOverloading3();
       obj.disp('x', 51 );
       obj.disp(52, 'y');
   }
}
```

Ekranda:

```
Bu birinchi disp() methodi
Bu ikkinchi disp() methodi
```

. Bu misolda overload methodi asosida yaratilgan disp() methodlari bor ular bir biridan ma’lumot turlarining ketma ketligi bilan farq qilyapti. Yani ma’lumot turlarini joylashuvi.

Ishlatish mumkun bo’lgan va mumkun bo’lmagan overload methodlarni qisqacha tariflari

139. Holar:

```
int mymethod(int a, int b, float c)
int mymethod(int var1, int var2, float var3)
```

Natija: Compile time error. Chunkiy bu methodlarni argumentlri, ma’lumot turlarlarini ketma-ketligi va ma’lumot turlari bir xil. Dastur ishga tushgan paytida qaysi bir methodga murojat qilishni tanlay olmaydi yani bu methodlarni farqlay olmaydi.

140. Holat:

```
int mymethod(int a, int b)
int mymethod(float var1, float var2)
```

bu methodalr ishalatishga yaroqli chunkiy methodlar bir biridan ma’lumot turlari bilan farq qilyapti.

141. Holat:

```
int mymethod(int a, int b)
int mymethod(int num)
```

bu methodlar ishalatishga yaroqli chunkiy methodlar bir biridan raqamlar argumenti bilan farq qilyapti.

142. Holat:

```
float mymethod(int a, float b)
float mymethod(float var1, int var2)
```

bu methodlar ishalatishga yaroqli chunkiy methodlar ma’lumotlar ketma-ketligi bilan farq qilyapti(joylashuvi)

143. Holat:

```
int mymethod(int a, int b)
float mymethod(int var1, int var2)
```

Natija: Compile time error. Chunkiy bu methodlar turli xil turdagi ma’lumotlarni qaytaryapti birinchi method int ikkinchisi float.

144. `void sum(int a,long b)`

145. `void sum(long a,int b)`

146.

Natija: Compile time error. Bu holatda ma’lumot turlari xar xil bo’lgani bilan ikkala method ham butun sonlarni saqlovchi tur shuning uchun dastur xato beradi.

Javada Constructor

Javada constructlar obyektni ishga tushurush uchun maxsus usuldur

Java konstruktor obyekt yaratish vaqtida chaqiriladi.

Javada construkor yaratish qoydalari

147.Construktr class nomi bilan o’xshash bo’lishi kerak

148.Constructor no aniq ma'lumot turini ega bo'lishi kerak

Construktorni ikkita turi mavjud

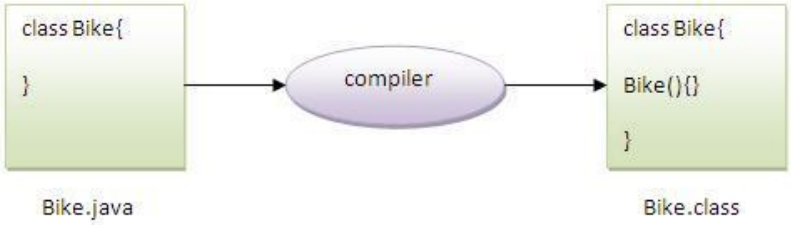
149.Argumentsiz construktrlar

150.Parametrlik construktrlar

Constructorga oddiy misol

Bu misolda argumentsiz constructr yaratildi Bike1 clasida.Obyekt yaratilishi vaqtida construktrni ishlatadi.

```
151.    class Bike1{
152.    Bike1(){System.out.println("Bike is created");}
153.    public static void main(String args[]){
154.    Bike1 b=new Bike1();
155.    }
156.    }
```



Rasimda ko’srsatilganidek java file compiler bo’lgandan keyin aftomatic default construktr yaratadi.

default constructor misol

```
157.    class Student3{
158.    int id;
159.    String name;
160.
161.    void display(){System.out.println(id+" "+name);}
162.
163.    public static void main(String args[]){
164.    Student3 s1=new Student3();
165.    Student3 s2=new Student3();
166.    s1.display();
167.    s2.display();
168.    }
169.    }
```

ekranda :

```
0 null
0 null
```

Parametrga ega bo'lgan constructr

```
170.    class Student4{
171.       int id;
172.       String name;
173.
174.       Student4(int i,String n){
175.       id = i;
176.       name = n;
177.       }
178.       void display(){System.out.println(id+" "+name);}
179.
180.       public static void main(String args[]){
181.       Student4 s1 = new Student4(111,"Karan");
182.       Student4 s2 = new Student4(222,"Aryan");
183.       s1.display();
184.       s2.display();
185.      }
186.    }
```

Ekrada:

```
111 Karan
222 Aryan
```

Bu misolda parametrga ega bo'lgan constructr yaratildi.

Constructr va method orasidagi farqlar

Constructor	Method
Constructr dan obyektni dastlabki holatini bilish uchun foydalaniladi	Method obyektni harakterini ifodalashda foydalaniladi
Constructor ma'lumot qaytarmaydi	Method ma'lumot qaytaradi
Constructr bilvosita chaqiriladi	Method oshkora chaqiriladi
Java kompilyator default constructr yaratadi agar constructr bo'lmasa	Kompilyatsiyada method yaratilmaydi
Constructr nomi class nomi bilan bir xil bo'ladi	Method nomi o'xshashi yokiy o;xshamasligi mumkun

Constructordan nusxa(copy) olish

```
187.    class Student6{
188.       int id;
189.       String name;
190.       Student6(int i,String n){
191.       id = i;
192.       name = n;
```

```
193.          }
194.
195.          Student6(Student6 s){
196.          id = s.id;
197.          name =s.name;
198.          }
199.          void display(){System.out.println(id+" "+name);}
200.
201.          public static void main(String args[]){
202.          Student6 s1 = new Student6(111,"Karan");
203.          Student6 s2 = new Student6(s1);
204.          s1.display();
205.          s2.display();
206.         }
207.       }
```

Javada Inheritance(mereosxo’rlik)

Ma'lum obyekt asosida boshqa obyektni yaratish jarayoniga aytiladi. Bir classning boshqa classdan 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 classda bo'lgan 4 g'ildirak farzand classlarda ham mavjud bo'ladi. Ya'ni poyga mashinasi, avtoulovdan g'ildiraklarni meros oladi.

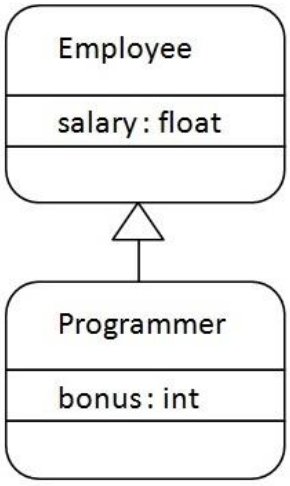
Javada inheritance dan

```
208.        Methodni overridinggi uchun (runtime da polimorfizimdan foydalanishda )
209.        Kodni qayta ishlash uchun
```
Foydalaniladi.

```
210.       class Subclass-name extends Superclass-name
211.       {
212.         //methods and fields
213.       }
```

extends kalit so’zi yangi class yaratayotganimzda Super class dan meros olish uchun ishlatiladi.

Java tehnalogoyasida quyi class super classdan meros oladi.Yani yangi class quyi class ni chaqiradi.



Yuqorida figurada Programmer subclassi va Employee supper classi ko'rasatilgan. Bunda shu ikkita classni orasidagi bog'lanish tasvirlangan.

```
214.    class Employee{
215.     float salary=40000;
216.    }
217.    class Programmer extends Employee{
218.     int bonus=10000;
219.     public static void main(String args[]){
220.       Programmer p=new Programmer();
221.       System.out.println("Programmer salary is:"+p.salary);
222.       System.out.println("Bonus of Programmer is:"+p.bonus);
223.    }
224.    }
```

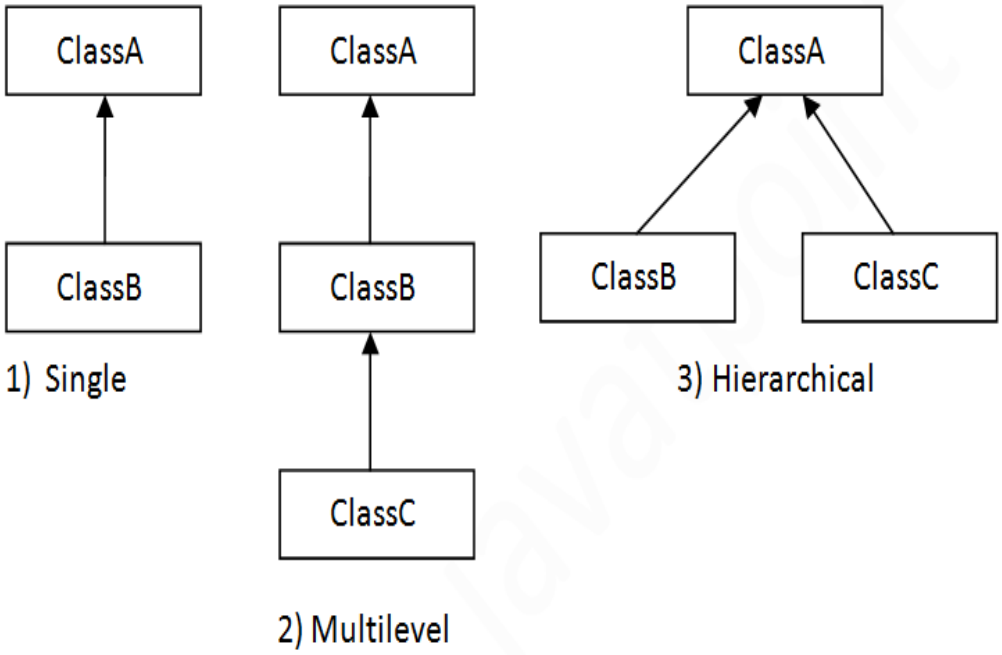
Natija :

```
Programmer salary is:40000.0
 Bonus of programmer is:10000
```

Yuqoridagi misolda Programmer classi Employee classidan meros olyapti va undagi **float salary** o'zgaruvchisini o'ziga o'zlashtirib olyapti.

Javada inheritance turlari

Javada asosiy classdan 3 xil meros olish mumkin ular **single**(bir tomonlama), **multilevel**(bir necha class orqali) va **hierarchical**(iyerarxiya).



Javada bir vaqtning o’zida bir necha class dan meros olib bo’lmaydi

Misol:

```
225.    class A{
226.    void msg(){System.out.println("Hello");}
227.    }
228.    class B{
229.    void msg(){System.out.println("Welcome");}
230.    }
231.    class C extends A,B
232.
233.     Public Static void main(String args[]){
234.       C obj=new C();
235.       obj.msg();//}
236.    }
```

Bu yerda

`Compile Time Error`

Bo’ladi chunkiy bir vaqtning o’zida **C** classi **A, B** classlaridan meros olyapti.Yuqorida aytganimizdek javada bir vaqtning o’zida bir necha classdan meros olib bo’lmaydi.

Method Overriding

Agar subclassimizdagi(bola class) method supper(ota class) clasimizdagi method bilan bir xil bo'lsa bu methodlar **overriding methodlar** deyiladi.

Javda overriding methoddidan foydalanish

Method overridning dan avvaldan supper classda yaratib qo'yilgan methodning o'ziga xos amalga oshirilishini taminlashda foydalaniladi.
Method overridning runtime polymorphism da foydalaniladi

Javda overriding method uchun qoydalar

Method super class(ota) dagi method nomi bilan bir xil bo'lishi kerak
Method super class(ota) dagi method parametrlari bilan bir xil bo'lishi lozim
Class meros olgan bo'lishi kerak

```
class Vehicle{
void run(){System.out.println("Vehicle is running");}
}
class Bike2 extends Vehicle{
void run(){System.out.println("Bike is running safely");}

public static void main(String args[]){
Bike2 obj = new Bike2();
obj.run();
}
```

```
Ekranda :Bike is running safely
```

Bu misolda **run()** methodi sub class va sepper claslarda yaratilgan. Bu methodlarni nomi va parametrlari o'xsha va claslar bir biri bilan bo'g'langan shuning uchun overriding methodlar hosil bo'lyapti.

Method overriding ga hayotiy misol olsak

Faraz qilaylik O'zbekistonda birnecha banklar bor va ularning foiz stafkalari har xil masalan agro bank 4% , hamkor bank 6%, xalq bank 7% va h.k. Shularning foiz satfkalarini chiqaruvchi dastur yozamiz.

```
class Bank{
int getOfInterest(){return 0;}
}

class AgroBank extends Bank{
int getOfInterest(){return 4;}
}

class HamkorBank extends Bank{
int getOfInterest(){return 6;}
}
class XalqBank extends Bank{
int getOfInterest(){return 7;}
}

class Test2{
public static void main(String args[]){
AgroBank  s=new AgroBank  ();
HamkorBank  i=new HamkorBank  ();
XalqBank  a=new XalqBank  ();
System.out.println("Agro bank Rate of Interest: "+s.getOfInterest ());
System.out.println("Hamkor bank Rate of Interest: "+i.getOfInterest ());
System.out.println("Xalq bank Rate of Interest: "+a.getOfInterest ());
}
}
```

```
Ekranda:
Agro bank Rate of Interest: 4
Hamkor bank Rate of Interest: 6
Xalq bank Rate of Interest: 7
```

Static method lar hech qachon override method bo'lmaydi. Chunkiy static method lar class bilan bo'glangan va u obyekt bilan bog'liq.

Main method lari ham override bo'lmaydi chunkiy u static method

Method Overloading	Method Overriding
Method Overloading class ichida bo'ladi	Method Overriding ikkita classs ichida bo'ladi yani bir biri bilan bog'langan claslarda.
Method overloading parametrlari xar xil bo'ladi	Method Overriding parametrlari bir xil bo'lishi shart
Method Overloadingda polemorfizm compile vaqtida ishlatadi	Method Overridingda polemorfizmdan runtime da foydalaniladi
Method Overloadingda qaytarayotgan ma'lumot turlari o'xshashi yokiy o'xshamasligi mumkin	Method Overridingda qaytarayotgan ma'lumot turlarti o'xshash bo'lishi kerak

Javada polemorfizim

Obyektga abstrakt darajada qarash hususiyati. Masalan, turli xil oynalar mavjud: deraza oynasi, eshik oynasi, mashina oynasi, telifon oynasi. Bularning barchasi bir biridan ishlatilish sohasi tuzulishi shakli bilan farq qiladi. Lekin barchasini umumiy qilib oyna deb qarash mumkun. Polemorfizim turli hil obyektlar bilan bir hil uniformada ishlash imoniyatini beradi.

Polimorfizm orqali bir jarayonni turli yo'llar bilan tashkillashtirishimiz mumkin. Polimorfizm so'zi yunoncha ikki so'zning birikmasidan tashkil otpgan «poly» — Ko'p va «morphs» — formalar. Polimorfizm ham *ko'p formalar* degan ma'noni anglatadi.

Javada polimorfizmni ikki turi mavjud: programma kompilatsiya bo'layotgan vaqtda sodir bo'ladigan polimorfizm (**compile time polymorphism**, **static polymorphism**) va programma ishlab turgan vaqtda sodir bo'ladigan polimorfizm (**runtime polymorphism**, **dynamic polymorphism**). Javada polimorfizm metodlarni qayta yuklash (overload) va qayta yozish (override) bilan amalga oshiriladi.

Compile time polimorfizm

Agar metodni *qayta yuklansa (**overload**)* bu — compile time polimorfizmga misol bo'ladi. Method overload haqida oldingi maqolalarimizda to’xtalib o’tganmiz

Quyidagi misolga qarang:

```
class X
{
   void metodA(int num)
   {
       System.out.println ("metodA:" + num);
   }
   void metodA(int num1, int num2)
   {
       System.out.println ("metodA:" + num1 + "," + num2);
   }
```

```
    double metodA(double num) {
        System.out.println("metodA:" + num);
        return num;
    }
}

class Y
{
    public static void main (String args [])
    {
        X Obj = new X();
        double natija;
        Obj.metodA(20);
        Obj.metodA(20, 30);
        result = Obj.metodA(5.5);
        System.out.println("Javob:" + result);
    }
}
```

Natija:

```
metodA:20
metodA:20,30
metodA:5.5
Javob:5.5
```

Yuqoridagi misolda, X classning 3 ta metodi bor. Uchala metodlarning xam nomlari bir xil — *metodA*, ammo metodlarning argumentlarida farqlar mavjud. Kompilyatsiya vaqtida kompilyator metodga yuborilayotgan argumentlarning soni va ma'lumot tipiga qarab kerakli metodni tanlab oladi. Qaysi metodlarni chaqirish kompilatsiya vaqtida aniqlanishi tufayli ham bu polimorfizm turi Compile time polimorfizm deb nomlanadi.

Compile time polimorfizmni yana Static binding, Early binding deb ham ataladi.

Runtime polimorfizm

Dastlab casting, up-casting va down-casting tushunchalari bilan tanishib olaylik (Aslida, runtime polimorfizmni tushunish un faqat up-casting ni o'zini bilish yetadi, ammo men to'lqiroq yozdim). Obyektni bir toifadan boshqa toifaga o'tkazishga *casting* deyiladi.

Agar ota classning obyekti bola class toifasga o'tkazilsa bu *down-casting* deyiladi. Misol:

```
Object satrObject = "Bu shunchaki satr"; // Object - ota class
String satrString = (String)ObyektSatr; // String - Object classning bola classi
```

Yuqoridagi misolda Object toifasidagi o'zgravchi (satrObject) String toifasiga o'zgartirilayapti. Bilamizki, Obyekt class barcha classlarning otasi hisoblanadi.

Agar bola classning obyekti ota class toifasga o'tkazilsa bu up-*casting* deyiladi. Misol:

```
String satrString = "Bu shunchaki satr";
Object satrObject = (Object)satrString;
```

Agar ota classning o'zgaruvchisi bola class *obyektini ko'rsatsa* (xotiradagi adresini ko'rsatsa) ham up-casting deyiladi. Misol:

```
class A{}
class B extends A{}
```

```
A a = new B(); //upcasting
```

Yuqoridagi misolda A class toifasidagi a o'zgaruvchi (reference variable) B bola classning *obyektini ko'rsatadi* (hotiradagi adresini ko'rsatadi).

Runtime polimorfizmda bola classdagi override qilingan metod, ota class toifasidagi o'zgaruvchi orqali chaqiriladi. Quyidagi misolga qarang:

```
class Chevrolet{
  void run(){System.out.println("running");}
}
```

```
class Malibu extends Chevrolet{
  void run(){System.out.println("CHevrolet Malibu");}

  public static void main(String args[]){
    Chevrolet ch = new Malibu();//upcasting
    ch.run();
  }
}
```

Naija:

```
CHevrolet Malibu
```

Yuqoridagi misolda biz ikkita class yaratayapmiz. Malibu class Chevrolet class dan nasl olayapti va run() metodni qayta yozayapti (override). Qayta yozilgan run metod ota class toifasidagi o'zgaruvchi ch orqali chaqirilayapti. ch o'zgaruvchi Malibu toifasidagi obyektga ko'rsatayotgani uchun Malibu classda qayta yozilgan metod chaqiriladi.

Javada abstract class

Javada ikki xil class mavjud ular **abstract, va abstract-bo'lmagan** classlar

Abstract kalit so'zi bilan yaratilgan classlar abstrakt classlar deb nomlanadi. Abstract classlar abstakt metodlar (tanasi yozilmagan metod) yoki abstrakt bo'lmagan metodlardan (tanasi yozilgan metod) tashkil topgan bo'ladi.

Abstrakt class quyidagicha e'lon qilinadi:

```
abstract class A{}
```

237. Abstrakt klaslar odatda super(ota) klaslar yaratishda ishlatiladi.
238. Abstract kalslardan yaratilayotgan klaslarimizni sxemasni yani skletini yaratishda foydlaniladi.
239. Abstract klasimizda undan meros oladigan klaslar uchun umumiy bo'lgan metodlar, konstantlar, o'zgaruvchilar e'lon qilinadi.

Tanasi (implementation) yozilmagana va abstract kalit so'zi bilan yaratilgan metod — **abstrakt metod** deyiladi.

```
abstract void funksiya();
```

Abstrakt metodi bor bo'lgan abstrakt classga misol:

```
public abstract class Mashina {

    abstract void madeli();

    abstract void mashinaTuri();

    abstract void uzunligi();

    abstract void rangi();

}

     public class Malibu  extends Mashina{

    @Override
    void madeli() {
        System.out.println("Chevrolet");
    }

    @Override
    void mashinaTuri() {
```

```
        System.out.println("Malibu");
    }

    @Override
    void uzunligi() {
        System.out.println("4.7 metr");
    }

    @Override
    void rangi() {
        System.out.println("Qora");
    }
    public static void main(String[] args) {
        Mashina mashina = new Malibu();
        mashina.madeli();
        mashina.mashinaTuri();
        mashina.uzunligi();
        mashina.rangi();
    }

}
```

Natija :

```
Chevrolet
Malibu
4.7 metr
Qora
```

Yuqoridagi misolda Malibu classi abstrakt metodlari (modeli, mashinaTuri, uzunligi, rangi) bor bo'lgan Mashina abstarkt classidan nasl olayapti va uning abstrakt classi tanasini realizatsiya(**`Override`**) qilayapti.

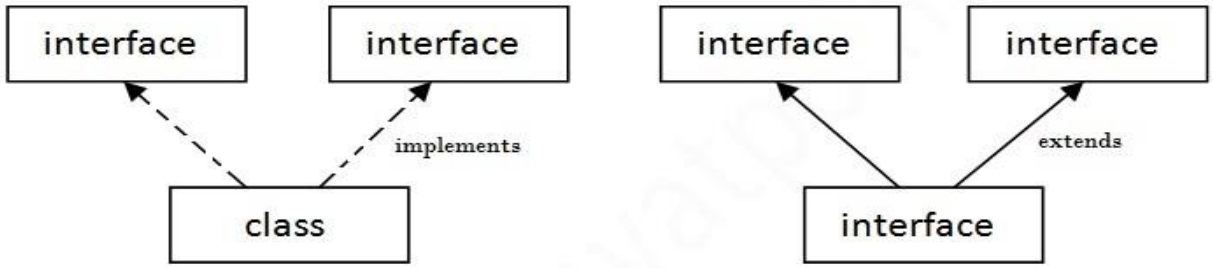
Amaliyotda abstrakt class bir nechta classlar uchun ota class vazifasini bajaradi va abstrakt metodlar turli bola classlarda turlicha realizatsiya qilinadi.

Shuningdek abstrakt class konstruktor, tanasi bor metod, ma'lumotlar va hattoki main() metodga ega bo'lishi mumkin. Quyidagi misolga qarang:

Esda tuting: Agar class tarkibida bitta bo'lsa ham abstarkt metod mavjud bo'lsa, class ham abstract bo'lishi kerak.

Esda tuting: Agar biror class abstrakt classdan voris olsa uning barcha *abstrakt metodlar*ini realizatsiya qilishi kerak yoki o'zi ham abstrakt class bo'lishi kerak.

Javada Interface



Multiple Inheritance in Java

Javada interfeyslar orqali *to'liq abstraktsiya*ni tashkillashtirish mumkin. Interfeys tarkibidagi metodlarning faqat nomlari bo'lishi mumkin, ularning tanasi bo'lishi mumkin emas(java 8 dan boshlab **default** funksiyalarga funksiyaning bajarilish qismi yozilishi qo'shildi). Shuningdek, Interfeys tarkibida ma'lumotlar bo'lishi ham mumkin.

Interfayslarda o'zgaruvchilar e'lon qilinmaydi unada fqat o'zgarmas qiymatlar **final** qiyamtlar e'lon qilinadi.

Interfacega bitta misol keltirsak.

```
public interface Dasta {

    void ishlatilishSohasi();

    void tayorlanganMatirial();

    void size();
```

```
}
```

Dasta nomli interface yaratildi unda ishlatilishSohasi(),tayorlanganMatirial(),size() nomli funksiyalari bor.

```
public class Pichoq implements Dasta{

    @Override
    public void ishlatilishSohasi() {
        System.out.println("Pichoq uchun");
    }

    @Override
    public void tayorlanganMatirial() {
        System.out.println("Yog'och");
    }

    @Override
    public void size() {
        System.out.println("15 sm");
    }
    public static void main(String[] args) {
        Pichoq pichoq = new Pichoq();
        pichoq.ishlatilishSohasi();
        pichoq.tayorlanganMatirial();
```

```
        pichoq.size();
    }


}
```

Ko’rib turganingizdek class yaratildi va **Dasta** interfacidan implement olindi. Bu holatda dasta barcha dastasi bor obyektlar uchun umumiy.

default funksiyaga misol :

```
interface Formula {
    double calculate(int a);

    default double sqrt(int a) {
        return Math.sqrt(a);
    }
}
Formula formula = new Formula() {
    @Override
    public double calculate(int a) {
        return sqrt(a * 100);
    }
};

formula.calculate(100);     // 100.0
formula.sqrt(16);
```

Esda tuting: Java kompilyatori interfeys tarkibidagi *metodlarga public va abstract*kalit so'zlarni, interfeys tarkibidagi*ma'lumotlarga esa public, static va final* kalit so'zlarni qo'shadi.

Bir class bir vatda bir nechta interfeyslarndan foydalanishi (implementation) mumkin. Quyidagi misolga qarang:

```
interface Printable{
void print();
}

interface Showable{
void show();
}

class A implements Printable,Showable{

public void print(){System.out.println("Salom");}
public void show(){System.out.println("Hush kelibsiz");}

public static void main(String args[]){
A obj = new A();
obj.print();
obj.show();
 }
}
```

Quyidagi misolda class interfeydan foydalanadi, interfeys esa boshqa interfeysdan nasl oladi:

```
interface Printable{
void print();
}

interface Showable extends Printable{
void show();
}
```

```
class A implements Showable{

public void print(){System.out.println("Salom");}
public void show(){System.out.println("Hush kelibsiz");}

public static void main(String args[]){
A obj = new A();
obj.print();
obj.show();
 }
}
```

Shunindek, bir interfyes tarkibida boshqa bir interyes bo'lishi ham mumkin:

```
   interface Xat{
      void xatKeldi();
      interface xatHabar{
        void xatHabarKeldi();
      }
     }
```

abstract class va interface o’rtasidagi farqlar

Abstract class	Interface
Abstract classda abstract va abstract bo’lmagan methodlar bo’ladi	Interface da faqat abstract methodlar bo’ladi
Classlar faqat bitta abstract classdan meros olishi mumkin	Classlar birnecha interfayslardan meros olishi mumkin
Abstract classlarda **final, non-final static va non-static** bo’lgan o’zgaruvchilar bo’lishi mumkin	Interface da esa faqat **final va static** o’zgaruvchilar bo’ladi
Abstract claslarda **static methodlar, main methodlar va constructrlar** yaratish mumkin	Interface da esa **static methodlar, main methodlar va constructrlar** yaratib bo’lmaydi

Interface va abstract class larga misol:

```
//interface da 4 ta method yaratilyapti
interface A{
void a();abstract
void b();
void c();
void d();
}

// B abstract classi yaratilyapti va A interface implement qilyapti
abstract class B implements A{
public void c(){System.out.println("Men c man ");}
}
class M extends B{
public void a(){System.out.println("Men a man ");}
public void b(){System.out.println("Men d man ");}
public void d(){System.out.println("Men d man ");}
}
class Test5{
public static void main(String args[]){
```

```
A a=new M();
a.a();
a.b();
a.c();
a.d();
}}
```

Natija :

```
Men a man
Men b man
Men d man
Men c man
```

Collection

Javada Collectionlar(to'plamar) freamwork dir u o'zida obyektlarni saqlaydi.

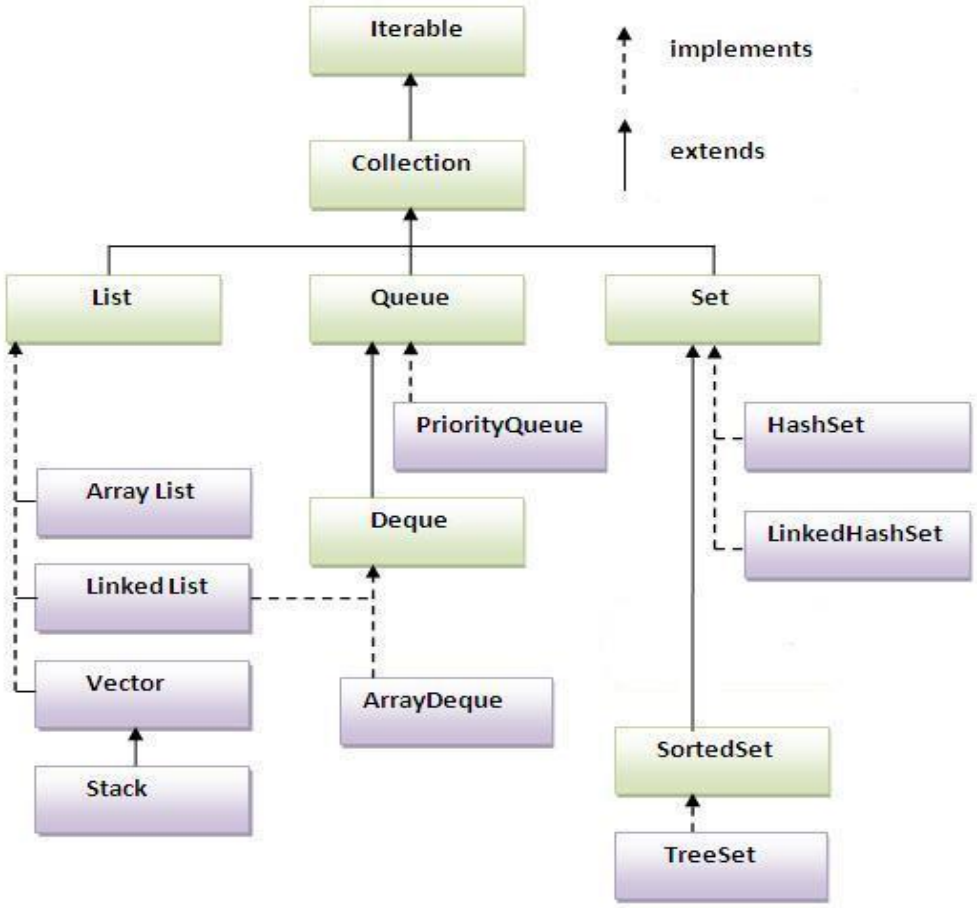
Collection larda barcha jarayonlar (**saralash, qidirish, yozish, o'chirish** va.h lar)ni oson amalga oshirish mumkun.

Collection lar bir necha interface lardan tashkil topgan. Ular (**Set, List, Queue, Deque etc.**) va class lar (**ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc**).

Collection obyektning alohida ko'rinishi.

Collection framework lar tuzulishi noma'lum bo'lgan obyet guruhlarini o'zida yig'adi. Collection lar ma'lumot yig'ishi bilan massivlarga o'xshab ketadi. Farqi masiv o'lchami satik collection larniki esa dinamik, masiv bilan collectionlarni qurulishi bir biridan tubdan faq qiladi va.k.

Collectionlarning tuzilish sxemasi:



Collectionlar **java.util** paketida saqlanadi.

List

List dublikat elementlarni o’z ichiga olishi mumkun. Elementlar joylashtirilgan yoki ro'yxatda o'z holatiga ruxsat etilishi mumkin.

Listdan implement qiladigan klaslar

ArrayList

LinkedList

Vector

Set

Set ham collection u o’zida dublikat elementlarni saqlamaydi. Setdan asosiy 3 ta class implement oladi ular: HashSet, TreeSet, and LinkedHashSet, HashSet elementlarni hashlab hash jadvaliga yig’adi.Treeset elementlarni daraxtsimon va saralangan holda saqlaydi saqlaydi.

Setdan implement qiladigan classlar

- **HashSet**
- **LinkedHashSet**
- **TreeSet**

Map

Map key ham value si ham obyekt bo'lgan collection. U o'zida dublkat bo'lagan key larni saqlamaydi. Mapdan e ta class implement oladi HashMap, TreeMap, and LinkedHashMap. HashMap : elementlarni hashlab yozadi, TreeMap : elementlarni qiymati(value) asosida saralab yozdadi.

Mapdan implement oladigan claslar

- **HashMap**
- **TreeMap**
- **LinkedHashMap**

Iterator/ListIterator

Iterator va ListIteratorlardan collection larning elementlarini qaytarishda foydalaniladi.

Collectionlarninig bazi funksiyalari

N	Funksiya nomlari	Tarifi
1	public boolean add(Object element)	Collectionga ma'lumot yizishda foydalanilari
2	public boolean addAll(Collection c)	Belgilangan collection larni yozishda ishlailadi
3	public boolean remove(Object element)	Collectionni elementini o'chirishda ishlatiladi
4	public boolean removeAll(Collection c)	Belegilangan collectionni o'chirishda ishlariladi
5	**public boolean retainAll(Collection c)**	
6	public int size()	Collectionlar elementlar sonini qaytaradi
7	public void clear()	Collectionni elementlarini o'chiradi
8	public boolean contains(Object element)	Element qidirishda ishlatilari
9	public boolean containsAll(Collection c)	Belgilangan colleectionlarni qidirshda ishlatiladi
10	public Iterator iterator()	Iterator qaytaradi
11	public Object[] toArray()	Collection ni massivga o'giradi

12	public boolean isEmpty()	Collectionni bo’sh yokiy bo’sh emasligiga tekshiradi
13	public boolean equals(Object element)	Ikkita collectionni bir biri bilan solishtiradi
14	public int hashCode()	Collectionni hash code dagi raqamini qaytaradi

Javada ArrayList class

1. Javada ArrayList class u ma’lumotlarni yig’ishda dinamik massivdan foydalanadi. ArrayList AbstractList ni extend va List ni implement qiladi.
2. Javada ArrayList class dublikat elementlarni ham saqlasi mumkun
3. Javada ArrayList class tartib bilan elementlarni yozadi(index ketma-ketligi).
4. Javada ArrayList class sinxron emas
5. Javada ArrayList classi ixtiyori kirishga (ixtiyoriy indexdagi elementni olish)ruxsat beradi chunkiy u asosi index bo’lgan massivda ishlaydi.
6. Java ArrayList class ida ma’lumotlarni boshqarish(o’chirish yozish) sekin chunkiy elementlarni ko’chirish o’tish jarayonlari ko’p sodir bo’ladi.

Javada jenerik bo’lmagan va jenerik bo’lgan collectionlar

Javada collectionlar jenerik bo’lmagan JDK 1.5 dan boshlab jeneriklar qo’shilgan.

Jenerik collectionlar sizga faqat bir turdagi obyektlar bilan ishlashga ruhsat beradi.

Jeneric bo’lmagan arraylist yaratishga misol:

```
ArrayList al=new ArrayList();//jenerik bo’lmagan array list yaratish
```

Jeneric bo’lgan arraylist yaratishga misol:

```
240.        ArrayList<String> al=new ArrayList<String>();//jeneric arraylist
```

Jenerik collection lar maxsus (<>) belgi bilan yaratiladi. Bunda ArrayList faqat bir turdagi obyektlar bilan ishlashligini bildiradi. Agar biz boshqa turdagi obyektni qo'shsak ***compile time error.*** Xatolik beradi

ArrayListga misol:

```
import java.util.*;
class TestCollection1{
 public static void main(String args[]){

  ArrayList<String> al=new ArrayList<String>();// arraylist yaratildi
  al.add("Zafar");//obyekt qo'shilyapti
  al.add("Jalol");
  al.add("Fayzullo");



  Iterator itr=al.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }
 }
}
```

Ekranda:

```
Zafar
Jalol
Fayzullo
```

Javada collectionlarni iterate() qilishni ikki xil usuli bor

1. Iterator interface orqali
2. For-eaach orqali

Yuqoridagi misol iterator orqali elementlarni chiqarishga misol bo'ladi.

For-each oraqil iterate qilish

```
import java.util.*;
class TestCollection2{
 public static void main(String args[]){
  ArrayList<String> al=new ArrayList<String>();
  al.add("Zafar");
  al.add("Jalol");
  al.add("Fayzullo");
```

```
      for(String obj:al)
       System.out.println(obj);
    }
   }
```

Ekaranda :

```
Zafar
Jalol
Fayzullo
```

ArrayList da saralash usullari

1. Yuqoridan quyiga qarab
2. Quyidan yuqoriga qarab

Yuqoridan quyiga qarab saralshga misollar

Sting turdagi ArrayList yaratamiz va undagi ma’lumotlarni saralaymiz

```
import java.util.*;
public class Details  {

        public static void main(String args[]){
           ArrayList<String> listofcountries = new ArrayList<String>();
           listofcountries.add("India");
           listofcountries.add("US");
           listofcountries.add("China");
           listofcountries.add("Uzbekistan");

           /*Unsorted List*/
           System.out.println("Saralashdan oldin:");
           for(String counter: listofcountries){
                        System.out.println(counter);
                }

           /* Sort statement*/
           Collections.sort(listofcountries);

           /* Sorted List*/
           System.out.println("Saralashdan keyin:");
           for(String counter: listofcountries){
                        System.out.println(counter);
                }
        }
}


Ekranda
Saralashdan oldin:
India
```

```
US
China
Uzbekistan
Saralashdan keyin:
China
India
US
uzbekistan
```

Integer sortga misol:

```
import java.util.*;
public class ArrayListOfInteger  {

        public static void main(String args[]){
           ArrayList<Integer> arraylist = new ArrayList<Integer>();
           arraylist.add(11);
           arraylist.add(2);
           arraylist.add(7);
           arraylist.add(3);
           /* ArrayList before the sorting*/
           System.out.println("Saralashdan oldin:");
           for(int counter: arraylist){
                        System.out.println(counter);
                }

           /* Sorting of arraylist using Collections.sort*/
           Collections.sort(arraylist);

           /* ArrayList after sorting*/
           System.out.println("Saralashdan keyin:");
           for(int counter: arraylist){
                        System.out.println(counter);
                }
        }
}
```

Ekranda:

```
Saralashdan oldin::
11
2
7
3
Saralashdan keyin:
2
3
7
11
```

Quyidan yuqoriga qarab saralash

Bu usulda elementlar quyi qisimdan yuoqri qisimga qarab saralanadi yani **z dan a** ga qarab

```
import java.util.*;
public class Details  {

        public static void main(String args[]){
           ArrayList<String> arraylist = new ArrayList<String>();
           arraylist.add("AA");
           arraylist.add("ZZ");
           arraylist.add("CC");
           arraylist.add("FF");

           /*Unsorted List: ArrayList content before sorting*/
           System.out.println("Saralashdan oldin:");
           for(String str: arraylist){
                        System.out.println(str);
                }

           /* Sorting in decreasing order*/
           Collections.sort(arraylist, Collections.reverseOrder());

           /* Sorted List in reverse order*/
           System.out.println("Saralangandan keyin:");
           for(String str: arraylist){
                        System.out.println(str);
                }
        }
}
Ekranda:

Saralashdan oldin:
AA
ZZ
CC
FF
Saralangandan keyin:
ZZ
FF
CC
AA
```

ArrayListda obyektlar bilan ishlash

```
class Student{
  int rollno;
  String name;
  int age;
```

```
  Student(int rollno,String name,int age){
   this.rollno=rollno;
   this.name=name;
   this.age=age;
  }
}
import java.util.*;
public class TestCollection3{
 public static void main(String args[]){

  Student s1=new Student(101,"Zafar",23);
  Student s2=new Student(102,"Aziz",21);
  Student s2=new Student(103,"Jalol",25);

  ArrayList<Student> al=new ArrayList<Student>();
  al.add(s1);// Student class object qo'shilyapti
  al.add(s2);
  al.add(s3);

  Iterator itr=al.iterator();
    while(itr.hasNext()){
    Student st=(Student)itr.next();
    System.out.println(st.rollno+" "+st.name+" "+st.age);
  }
 }
}
```

Ekranda:

```
101 Zafar 23
102 Aziz 21
103 Jalol 25
```

addAll() funksiyasi yani ikkita listni bir biri bilan qo'shishga doir misol:

```
import java.util.*;
class TestCollection4{
 public static void main(String args[]){

  ArrayList<String> al=new ArrayList<String>();
  al.add("Aziz");
  al.add("Zafar");
```

```
  al.add("Jalol");

  ArrayList<String> al2=new ArrayList<String>();
  al2.add("Fayzullo");
  al2.add("Jamshid");

  al.addAll(al2);
  Iterator itr=al.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }
 }
}
```

Ekranda :

```
Aziz
Zafar
Jalol
Fayzullo
Jamshid
```

Massivni ArrayListga qo’shish ga doir bazi usullarni

```
import java.util.*;

public class ArrayToArrayList {
     public static void main(String[] args) {

          /* Array Declaration and initialization*/
          String citynames[]={"Andijon", "Namangan", "Farg’ona", "Qo’qon"};

          /*Array to ArrayList conversion*/
          ArrayList<String> citylist= new ArrayList<String>(Arrays.asList(citynames));

          /*Adding new elements to the converted List*/
          citylist.add("Pop");
          citylist.add("Chust");

          /*Final ArrayList content display using for*/
          for (String str: citylist)
          {
                System.out.println(str);
          }
      }
}
```

Ekranda:

```
Andijon
Namangan
Farg’ona
Qo’qon
Pop
Chust
```

Collections.addAll method didan foydalanib massivni Arraylistag qo’shish

```
import java.util.*;

public class Example2 {
        public static void main(String[] args) {

            /* Array Declaration and initialization*/
            String array[]={"Hi", "Hello", "Howdy", "Bye"};

            /*ArrayList declaration*/
            ArrayList<String> arraylist= new ArrayList<String>();

            /*Conversion*/
            Collections.addAll(arraylist, array);

            /*Adding new elements to the converted List*/
            arraylist.add("String1");
            arraylist.add("String2");

            /*Display array list*/
            for (String str: arraylist)
            {
                System.out.println(str);
            }
        }
}
```

Ekranda:

```
Hi
Hello
Howdy
Bye
String1
String2
```

Collection ning methodlaridan foydalanmasdan oddiy sikil yordamida qo’shish

```
import java.util.*;

public class Details {
        public static void main(String[] args) {

            /*ArrayList declaration*/
            ArrayList<String> arraylist= new ArrayList<String>();

            /*Initialized Array*/
            String array[] = {"Text1","Text2","Text3","Text4"};

            /*array.length returns the current number of
             * elements present in array*/
            for(int i =0;i<array.length;i++)
            {

                 /* We are adding each array's element to the ArrayList*/
                 arraylist.add(array[i]);
            }
```

```
            /*ArrayList content*/
            for(String str: arraylist)
            {
                 System.out.println(str);
            }
      }
}
```

Ekranda:

```
Text1
Text2
Text3
Text4
```

Endi esa yozgan jarayonlarimizni teskarisi yani ArrayListdan massivaga o’girish usullariga doir misollar

Collection ning methodlaridan foydalanilmagan holda ArrayListni massivga o’girish

```
import java.util.*;

public class ArrayListTOArray {
        public static void main(String[] args) {

                /*ArrayList declaration and initialization*/
                ArrayList<String> arrlist= new ArrayList<String>();
                arrlist.add("String1");
                arrlist.add("String2");
                arrlist.add("String3");
                arrlist.add("String4");

                /*ArrayList to Array Conversion */
                String array[] = new String[arrlist.size()];
                for(int j =0;j<arrlist.size();j++){
                  array[j] = arrlist.get(j);
                }

                /*Displaying Array elements*/
                for(String k: array)
                {
                        System.out.println(k);
                }
        }
}
```

Ekranda:

```
String1
String2
String3
String4
```

toArray methodidan foydlanib listni masiga o’girish

```
import java.util.*;

public class Example {
        public static void main(String[] args) {

                /*ArrayList declaration and initialization*/
                ArrayList<String> friendsnames= new ArrayList<String>();
                friendsnames.add("Aziz");
                friendsnames.add("Zafar");
                friendsnames.add("Jalol");
                friendsnames.add("Fayzullo");

                /*ArrayList to Array Conversion */
                String frnames[]=friendsnames.toArray(new String[friendsnames.size()]);

                /*Displaying Array elements*/
                for(String k: frnames)
                {
                        System.out.println(k);
                }
        }
}
```

Ekranda:

```
Aziz
Zafar
Jalol
Fayzullo
```

Javada LinkedList classi

- Javada LinkedList doubly linked list dan foydalanib elementlarni yig’adi. U AbstractList clasidan meros(extends) oladi va List va Deque interface lardan implement oladi ArrayListdan farqlaridan ham biri Deque dan implement olishidur.
- LinkedList da bir xil elementlar saqalsh mumkun
- LinkedList da malumotlar krirtilish ketma ketligida saqlanadi
- LinkedList class sinxron emas
- LinkedList classda boshqarish jarayonlari tez sodir bo’ladi chunkiy LinkedList da elementlarni ko’chirib o’tkazilmaydi
- LinkedList list, steck yokiy queue lardan foydalanishi mumkun

LinkedList da ham ma’lumotlar bilan ishlash ArrayList bilan deyarli birxil amalga oshiriladi.

```
import java.util.*;
public class TestCollection7{
 public static void main(String args[]){

  LinkedList<String> al=new LinkedList<String>();
  al.add("Zafar");
  al.add("Aziz");
  al.add("Jalol");
  al.add("Zafar");

  Iterator<String> itr=al.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }
 }
}
```

ArrayList saqalagan ma’lumotlarni LinkedList ham saqlashi mumkun

LinkedListda gi ma’lumotlarni ArrayListga qo’shib qo’yishga misol

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

public class ConvertExample {
 public static void main(String[] args) {
    LinkedList<String> linkedlist = new LinkedList<String>();
    linkedlist.add("Zafar");
    linkedlist.add("Aziz");
    linkedlist.add("Jalol");
    linkedlist.add("Fayzullo");

    List<String> list = new ArrayList<String>(linkedlist);

    for (String str : list){
      System.out.println(str);
```

```
    }
 }
}
```

Ekranda:

```
Zafar
Aiz
Jalol
Fayzullo
```

ArraList va LinkedList lar orasidagi farqlar

ArrayList va LinkedList orasida unchalik farqlar ko’p emas. Ulardan bazi bir farqlarni ko’rib chiqsak.

3. **Qiriruv:** ArrayListda qiriruv jarayoni LinkedListga qaraganda tez amalga oshiriladi. ArrayListda qidiruv davomiyligi O(1) LinkeList da esa O(n) ga teng.

Tarif: ArrayList da elementlar uchun index lar biriktrilgan ma’lumotlarni saqlash strukturasida massivdan foydalanganligi uchun ArrayListda qidiruv tez amalga oshiriladi.LinkedListda esa elementlar joylashuvi boshqacha u doubly linked listni implenet qiladi.Shuning uchun qidirilayotgan elementni hamma elementlar ichidan qidiradi.

4. **O’chirish:** LinkedList da o’chirishni amalga oshirilishi O(1) ga teng ArrayListda esa ochirilishni bajarilishi O(n) ga teng.

LinkedList da o’chirish ArrayListga nisbatan tezroq amalga oshadi.

Tarif: LinkedList elementlarida ikkita nuqta mavjud yani elementnig qo’shnilari o’zidan oldigi va keyingi element o’chirilasa faqat shu qo’shnilarini joylashuvi o’zgaradi halos. ArrayListda esa element o’chirilgandan keyin barcha elementlarga index lar boshqatdan beriladi yani har bir elementni boshqa index ga ko’chrib o’tish jarayoni bajariladi.

5. **Ma’lumot yozish:** LinkedList da add funksiyasini bajarilishi O(1) teng ArrayListda esa bu holat O(n) ga teng. Yozish jarayoni o’chirish jarayonida bo’ladigan ammalar bajariladi.
6. Qisqacha qilib aytganda ArrayList ma’lumotlarni saqalash va qidirishga LinkedList esa ma’lumotlarni qayta ishlashga yaxshidur.

ArrayList va LinkedList lardan qachon foydalaniladi.

Yuqoridagi tariflarda aytib o'tilganidek yozish va o'chirishda LinkedList ArrayListga qaraganda ancha tez amalga oshiriladi.Ma'lumot o'chirib yozishda eng yaxshi tanlov bu LinkedList Hisoblanadi.

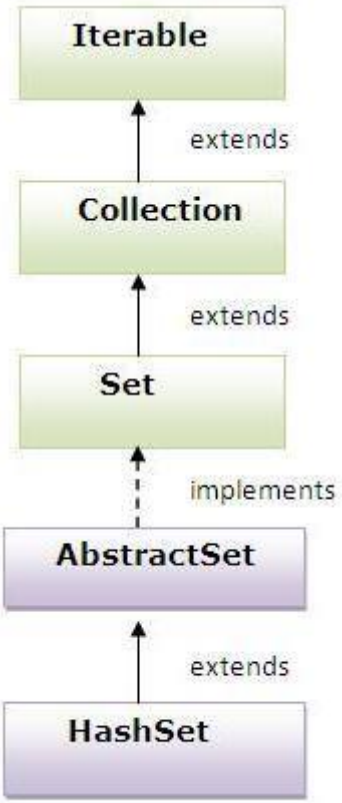
Qidirish da ArrayList Linkelistga qaraganda jarayona tezroq amalga oshiriladi. Shuning uchun qidirish jarayoni amalga oshirilganda ArrayList eng yaxshi tanlov

Javada Set

Set ham collection u ham o'zida dublicat bo'lmagan(dublictdan faqat bittasini saqlaydi) ma'lumotlarni saqlaydi. Set interface bo'lib undan **TreeSet HashSet LinkedHashSet** Abstract claslari Set dan implement oladi. Ular bilan tanishib chiqsak.

HashSet

HashSet ning ierarxiyasi



HashSet Set dan implement oladi. U o’zida null qiymatni saqlashga ruxsat beradi. Bu class sinxron emas.

1. Hashset ma’lumotlarni tasodifiy tartibda qaytaradi
2. HashSet dublikat ma’lumotlarni o’zida saqlamaydi
3. hashSet null qiymatlarni ham saqlashi mumkun lekin faqat bitta null qiymatni qaytaradi.
4. HashSet sinrxon emas

HashSetga misol:

```
import java.util.HashSet;
public class HashSetExample {
   public static void main(String args[]) {
      // HashSet declaration
      HashSet<String> hset =
               new HashSet<String>();

      // Adding elements to the HashSet
      hset.add("Apple");
      hset.add("Mango");
      hset.add("Grapes");
      hset.add("Orange");
      hset.add("Fig");
      //Addition of duplicate elements
      hset.add("Apple");
      hset.add("Mango");
      //Addition of null values
      hset.add(null);
      hset.add(null);

      //Displaying HashSet elements
      System.out.println(hset);
    }
}
```

Ekranda:

```
[null, Mango, Grapes, Apple, Orange, Fig]
```

Ko’rib turganingizdek ekranda bitta null qiymat va barcha dublicat elementlardan faqat bittasini o’zida saqalayapti.

HashSet da ishlovchi bazi bir funksiyalar bilan tanishib chiqsak

Hamma elementlarni o’chirib tashlashga doir misol:

```
import java.util.HashSet;
class EmptyHashSetExample{
  public static void main(String[] args) {
     // Create a HashSet
     HashSet<String> hset = new HashSet<String>();

     //add elements to HashSet
     hset.add("Element1");
     hset.add("Element2");
     hset.add("Element3");
     hset.add("Element4");
     hset.add("Element5");

          System.out.println("Oldin: HashSet contains: "+ hset);


     hset.clear();

     // Display HashSet content again
     System.out.println("Keyin: HashSet contains: "+ hset);
  }
}
```

Ekranda

```
Oldin: HashSet contains: [Element1, Element2, Element3, Element4, Element5]
Keyin: HashSet contains: []
```

HashSet ni massivga convert qilish ga misol:

```
import java.util.HashSet;
class ConvertHashSettoArray{
  public static void main(String[] args) {
     // Create a HashSet
     HashSet<String> hset = new HashSet<String>();

     //add elements to HashSet
     hset.add("Element1");
     hset.add("Element2");
     hset.add("Element3");
     hset.add("Element4");

     // Displaying HashSet elements
     System.out.println("HashSet contains: "+ hset);

     // Creating an Array
     String[] array = new String[hset.size()];
     hset.toArray(array);

     // Displaying Array elements
     System.out.println("Array elements: ");
     for(String temp : array){
        System.out.println(temp);
     }
  }
}
```

Ekranda:

```
HashSet contains: [Element1, Element2, Element3, Element4]
Array elements:
Element1
Element2
Element3
Element4
```

Hashsetga ArrayListga convert qilish:

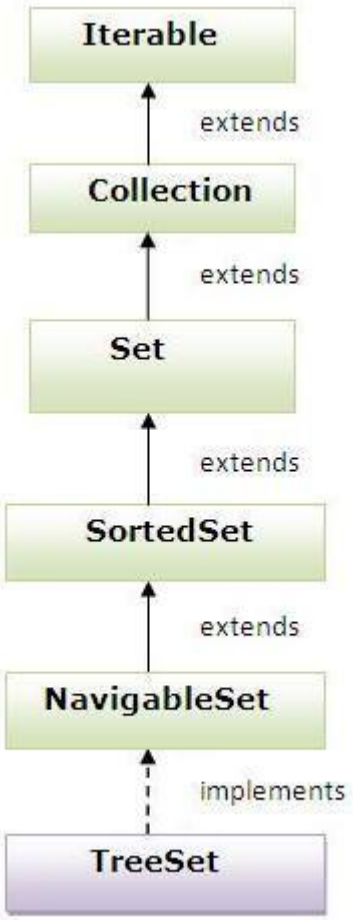
```
import java.util.HashSet;
import java.util.List;
import java.util.ArrayList;
class ConvertHashSetToArrayList{
  public static void main(String[] args) {
     // Create a HashSet
     HashSet<String> hset = new HashSet<String>();

     //add elements to HashSet
     hset.add("Steve");
     hset.add("Matt");
     hset.add("Govinda");
     hset.add("John");
     hset.add("Tommy");

     // Displaying HashSet elements
     System.out.println("HashSet contains: "+ hset);

     // Creating a List of HashSet elements
     List<String> list = new ArrayList<String>(hset);

     // Displaying ArrayList elements
     System.out.println("ArrayList contains: "+ list);
  }
}
```

Ekranda

```
HashSet contains: [Tommy, Matt, Steve, Govinda, John]
ArrayList contains: [Tommy, Matt, Steve, Govinda, John]
```

TreeSet

TreeSet ham deyarli HashSetga o’xshab ketadi. Bazibi hususiyatlari bilan ular bir biridan farq qiladi. Yani TreeSet elemetlarni saralab qaytaradi HashSet esa saralamay, TreeSet null qiymatni saqlamaydi HashSet esa saqlaydi. TreeSet sinxron emas.




```
TreeSet<String> tset = new TreeSet<String>();//treeSet ni e’lon qilinish
```

TreeSet ga misol:

```
import java.util.TreeSet;
public class TreeSetExample {
     public static void main(String args[]) {
         // TreeSet of String Type
         TreeSet<String> tset = new TreeSet<String>();

         // Adding elements to TreeSet<String>
         tset.add("ABC");
         tset.add("String");
         tset.add("Test");
         tset.add("Pen");
         tset.add("Ink");
         tset.add("Jack");

         //Displaying TreeSet
         System.out.println(tset);

         // TreeSet of Integer Type
         TreeSet<Integer> tset2 = new TreeSet<Integer>();

         // Adding elements to TreeSet<Integer>
```

```
        tset2.add(88);
        tset2.add(7);
        tset2.add(101);
        tset2.add(0);
        tset2.add(3);
        tset2.add(222);
        System.out.println(tset2);
    }
}
```

Ekranda:

```
[ABC, Ink, Jack, Pen, String, Test]
[0, 3, 7, 88, 101, 222]
```

Hashset va TreeSet orasidagi bazi bir faqlar

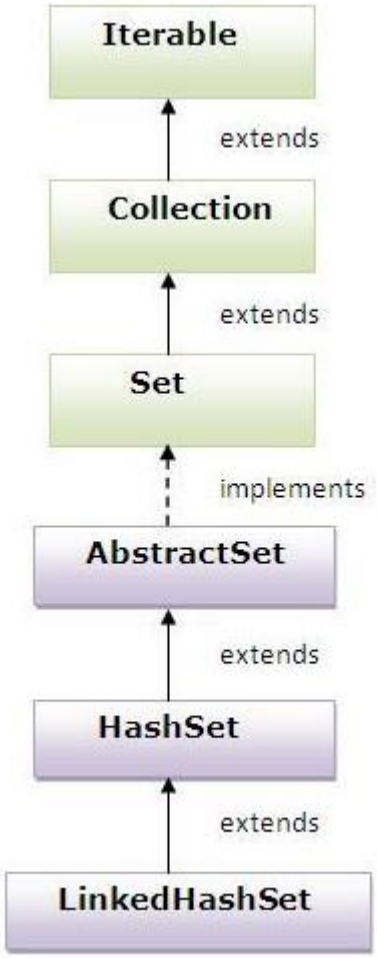
1. Hashset da remove, add contains, size lar TreeSet ga nisbatan ancha tezroq bajariladi. Hashset da bu jarayonlarni bajarilishi vaqti hashsetni bajarilish vaqtiga teng Treeset da esa vaqt log(n) ga teng.
2. HashSet elementlarni tartib bilan saqalamaydi Treeset esa elementarni saralab sqalaydi.

O’xshashliklari

1. HashSet Ham Treeset Ham dublicat ma’lumotlarni saqlamaydi
2. Agar siz Setni elementlarini saralashni hohlasangiz Hashsetni Treesetga convert qilishingiz kerak.
3. Ikkala class ham sinxron emas

LinkedHashSet

Yuqoridagi maruzalarimizda HashSet va TreeSet lar bilan tanishib o’tdik. LinkedHashSet ham Set dan implement oladi va HashSet va TreeSet larga juda o’xshab ketadi.



1. HashSet elementlari kiritilish tartibi bilan sqalamaydi
2. TreeSort esa elementlari saralab saqlaydi
3. LinkedHashSet esa elementlarni kiritilish tartibi bilan saqlaydi

```
import java.util.LinkedHashSet;
public class LinkedHashSetExample {
     public static void main(String args[]) {
          // LinkedHashSet of String Type
          LinkedHashSet<String> lhset = new LinkedHashSet<String>();

          // Adding elements to the LinkedHashSet
          lhset.add("Z");
          lhset.add("PQ");
          lhset.add("N");
          lhset.add("O");
          lhset.add("KK");
          lhset.add("FGH");
          System.out.println(lhset);

          // LinkedHashSet of Integer Type
          LinkedHashSet<Integer> lhset2 = new LinkedHashSet<Integer>();

          // Adding elements
          lhset2.add(99);
          lhset2.add(7);
          lhset2.add(0);
```

```
        lhset2.add(67);
        lhset2.add(89);
        lhset2.add(66);
        System.out.println(lhset2);
    }
}
```

Ekaranda:

```
[Z, PQ, N, O, KK, FGH]
[99, 7, 0, 67, 89, 66]
```

Ko’rib turganingizdek elementlarni saqlash tartibida qaytaryapyti(ekranka chiqaryapti)

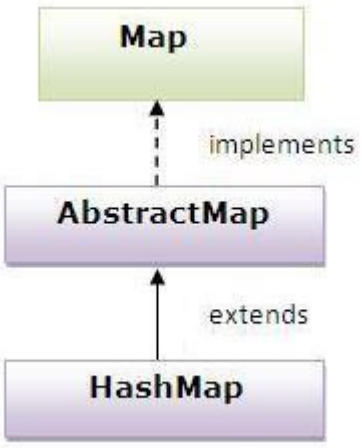
Map

Map ham to’plam bo’lib boshqalaridan farqi key(kaliti) ham obyekt saqlasnishidadur. Mapda key hech qachon dublikat bo’lishi mumkun emas. Map interface bo’lib undan HashMap TreeMap LinkedHashMap lar implement oladi.

HashMap clashi

HashMap asosi Map interface bo’lgan to’plamadur(collection) u o’zida ikki juft qiymatlardan foydalanadi ular **key** va **value** . Bu class elementlarni saqlanish tartibida saqlamaydi. Huddi HashTable class iga o’xshab ketadi va u sinxron emas o’zida key i ham value sini ham null qiymat saqalshi mumkun. hashMap da faqat bitta null ney saqlash mumkun.

E’lon qilinishi **HashMap<Key, Value>**



HahshMap ga misol:

```
import java.util.HashMap;
import java.util.Map;
import java.util.Iterator;
import java.util.Set;
public class Details {

   public static void main(String args[]) {

      /* This is how to declare HashMap */
      HashMap<Integer, String> hmap = new HashMap<Integer, String>();

      /*Adding elements to HashMap*/
      hmap.put(12, "Aziz");
      hmap.put(2, "Zafar");
      hmap.put(7, "Jalol");
      hmap.put(49, "Fayzullo");
      hmap.put(3, "Jamshid");

      /* Display content using Iterator*/
      Set set = hmap.entrySet();
      Iterator iterator = set.iterator();
      while(iterator.hasNext()) {
         Map.Entry mentry = (Map.Entry)iterator.next();
         System.out.print("key is: "+ mentry.getKey() + " & Value is: ");
         System.out.println(mentry.getValue());
      }

      /* key orqali elementni olish
      String var= hmap.get(2);
      System.out.println("Value at index 2 is: "+var);

      /* key orqalis elemetni o'chirish*/
      hmap.remove(3);
      System.out.println("Map key and values after removal:");
      Set set2 = hmap.entrySet();
      Iterator iterator2 = set2.iterator();
      while(iterator2.hasNext()) {
          Map.Entry mentry2 = (Map.Entry)iterator2.next();
          System.out.print("Key is: "+mentry2.getKey() + " & Value is: ");
          System.out.println(mentry2.getValue());
       }

   }
}
```

Ekranda:

```
key is: 49 & Value is: Fayzullo
key is: 2 & Value is: Zafar
key is: 3 & Value is: Jamshid
key is: 7 & Value is: Jalol
key is: 12 & Value is: Aziz
Map key and values after removal:
Key is: 49 & Value is: Fayzullo
Key is: 2 & Value is: Zafar
Key is: 7 & Value is: Jalol
Key is: 12 & Value is: Aziz
```

HashMap ni elementlarini olshining ikki xil usuli bor

1. For loop orqali
2. While loop + iterator

Misol:

```
import java.util.HashMap;
import java.util.Map;
import java.util.Iterator;
public class Details
{
    public static void main(String [] args)
    {
        HashMap<Integer, String> hmap = new HashMap<Integer, String>();
        //Adding elements to HashMap
        hmap.put(11, "AB");
        hmap.put(2, "CD");
        hmap.put(33, "EF");
        hmap.put(9, "GH");
        hmap.put(3, "IJ");

        //FOR LOOP
        System.out.println("For Loop:");
        for (Map.Entry me : hmap.entrySet()) {
          System.out.println("Key: "+me.getKey() + " & Value: " + me.getValue());
        }

        //WHILE LOOP & ITERATOR
        System.out.println("While Loop:");
        Iterator iterator = hmap.entrySet().iterator();
        while (iterator.hasNext()) {
             Map.Entry me2 = (Map.Entry) iterator.next();
          System.out.println("Key: "+me2.getKey() + " & Value: " + me2.getValue());
        }
    }
}
```

Ekranda:

```
For Loop:
Key: 2 & Value: CD
Key: 3 & Value: IJ
Key: 33 & Value: EF
Key: 9 & Value: GH
Key: 11 & Value: AB
While Loop:
Key: 2 & Value: CD
Key: 3 & Value: IJ
Key: 33 & Value: EF
Key: 9 & Value: GH
Key: 11 & Value: AB
```

HashMap ni key orqali saralash

Bu misolda HashMap ning key ni TreeMap dan foydalain saralash usulini ko’ramiz.

```
import java.util.HashMap;
import java.util.Map;
import java.util.TreeMap;
import java.util.Set;
import java.util.Iterator;

public class Details {

    public static void main(String[] args) {

         HashMap<Integer, String> hmap = new HashMap<Integer, String>();
         hmap.put(5, "A");
         hmap.put(11, "C");
         hmap.put(4, "Z");
         hmap.put(77, "Y");
         hmap.put(9, "P");
         hmap.put(66, "Q");
         hmap.put(0, "R");

         System.out.println("Saralashdan oldin:");
         Set set = hmap.entrySet();
         Iterator iterator = set.iterator();
         while(iterator.hasNext()) {
               Map.Entry me = (Map.Entry)iterator.next();
               System.out.print(me.getKey() + ": ");
               System.out.println(me.getValue());
         }
         Map<Integer, String> map = new TreeMap<Integer, String>(hmap);
         System.out.println("Saralashdan keyin:");
         Set set2 = map.entrySet();
         Iterator iterator2 = set2.iterator();
         while(iterator2.hasNext()) {
              Map.Entry me2 = (Map.Entry)iterator2.next();
              System.out.print(me2.getKey() + ": ");
              System.out.println(me2.getValue());
         }
    }
}
```

Ekranda:

```
Saralashdan oldin:
0: R
4: Z
5: A
66: Q
9: P
77: Y
11: C
Saralashdan keyin:
0: R
4: Z
5: A
9: P
```

```
11: C
66: Q
77: Y
```

HashMap ni Comerator dan foydalanib value sini saralashga misol:

```
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Set;

public class HMapSortingByvalues {
  public static void main(String[] args) {
      HashMap<Integer, String> hmap = new HashMap<Integer, String>();
      hmap.put(5, "A");
      hmap.put(11, "C");
      hmap.put(4, "Z");
      hmap.put(77, "Y");
      hmap.put(9, "P");
      hmap.put(66, "Q");
      hmap.put(0, "R");
      System.out.println("Saralashdan oldin:");
      Set set = hmap.entrySet();
      Iterator iterator = set.iterator();
      while(iterator.hasNext()) {
           Map.Entry me = (Map.Entry)iterator.next();
           System.out.print(me.getKey() + ": ");
           System.out.println(me.getValue());
      }
      Map<Integer, String> map = sortByValues(hmap);
      System.out.println("Saralashdan keyin:");
      Set set2 = map.entrySet();
      Iterator iterator2 = set2.iterator();
      while(iterator2.hasNext()) {
           Map.Entry me2 = (Map.Entry)iterator2.next();
           System.out.print(me2.getKey() + ": ");
           System.out.println(me2.getValue());
      }
  }

  private static HashMap sortByValues(HashMap map) {
       List list = new LinkedList(map.entrySet());
       // Defined Custom Comparator here
       Collections.sort(list, new Comparator() {
            public int compare(Object o1, Object o2) {
               return ((Comparable) ((Map.Entry) (o1)).getValue())
                  .compareTo(((Map.Entry) (o2)).getValue());
            }
       });

       // Here I am copying the sorted list in HashMap
       // using LinkedHashMap to preserve the insertion order
       HashMap sortedHashMap = new LinkedHashMap();
       for (Iterator it = list.iterator(); it.hasNext();) {
```

```
            Map.Entry entry = (Map.Entry) it.next();
            sortedHashMap.put(entry.getKey(), entry.getValue());
        }
        return sortedHashMap;
    }
}
```

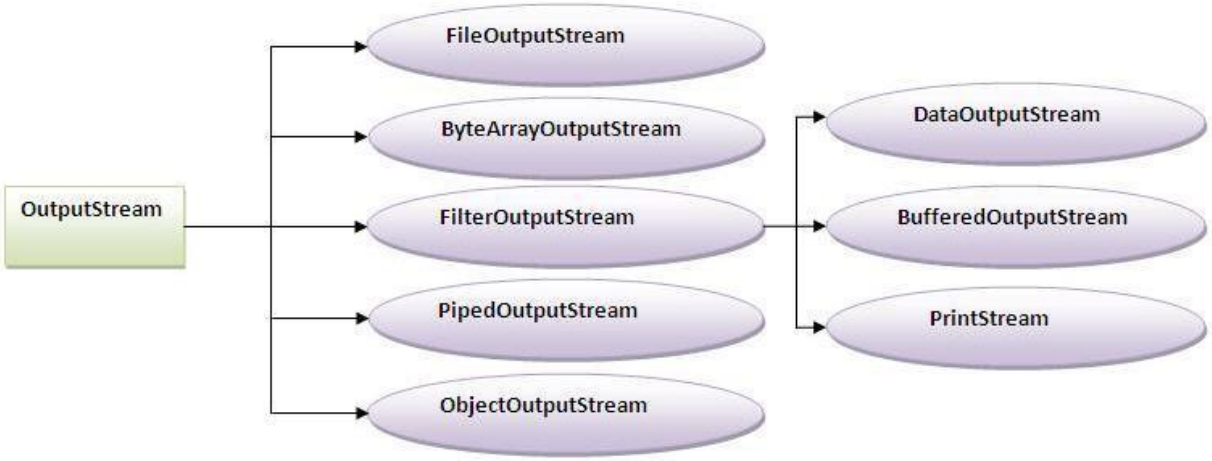
Ekranda:

```
Saralashdan oldin:
0: R
4: Z
5: A
66: Q
9: P
77: Y
11: C
Saralashdan keyin:
5: A
11: C
9: P
66: Q
0: R
77: Y
4: Z
```

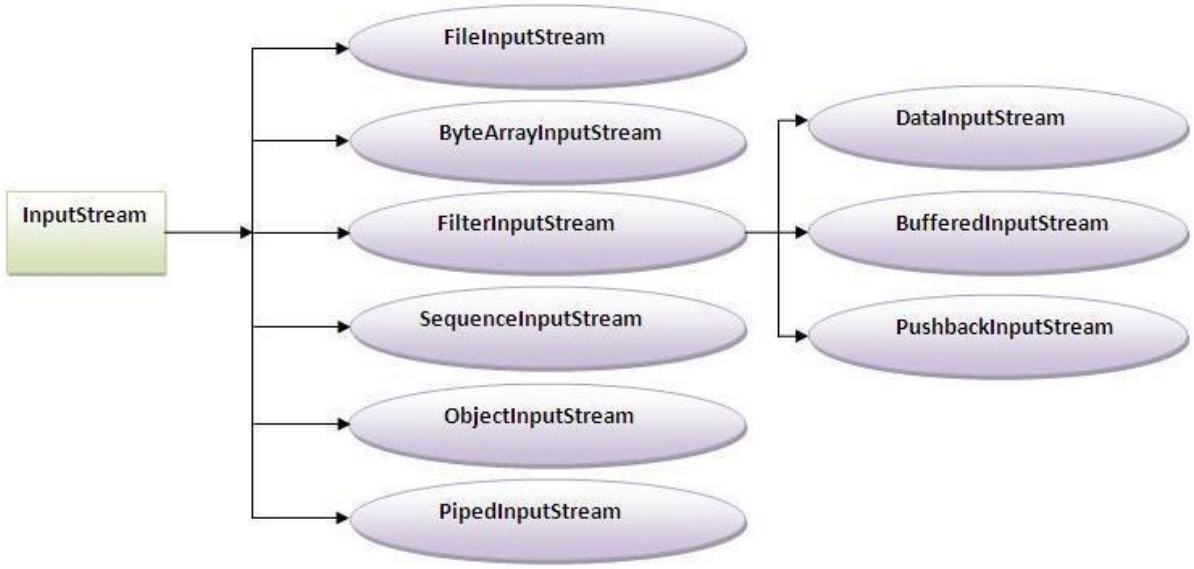
Javada I/O **(Input and Output)**

Java I/O (Input and Output) dan kiritish chiqarish jarayonlaridan foydalaniladi.

OutputStream class lar



InputStream class lashlar



FileInputStream va FileOutputStream

Java FileOutputStream class

Javda FileOutputStrimdan filega ma’lumot yozishda foydalaniladi