# Leksione në Java

# 2019

Ky set prej 15 leksionesh dhe shembujsh është një përmbledhje teorike për lëndën Java 1, për studentët e vitit te dytë Informatikë.

Sidita Duli

# Leksion 1

## Prezantim me gjuhën e programimit Java

Gjuha e programimit Java është projektuar fillimisht nga Sun Microsystems. U inicializua nga James Gosling, dhe lancuar në vitin 1995, si një komponent kryesor në platformën e Sun Microsystems. Versioni i saj i parë ishte Java 1.0 J2SE.

Versioni i fundit i Java Standart Edition është Java SE 8. Duke u nisur nga ndryshimet e Java dhe nga popullariteti i shpërndarë i saj, janë shtuar shumë konfigurime për shumë tipa të ndryshme platformash, siç janë J2EE për Enterprise Applications, J2ME për Mobile Applications, etj.

Versionet e reja të J2 janë riemëruar si Java SE, Java EE dhe Java ME.

Java garanton parimin : Write once, run anywhere.

Java është :

- E orientuar nga objektet – Në Java çdo gjë është objekt. Java është lehtësisht e zgjerueshme përderisa është e bazuar në modelin e objekteve.
- E pavarur nga platforma – Ndryshe nga gjuhë të tjera programimi, duke përfshirë edhe C apo C++, kur kompilohet në Java, nuk kompilohet në një makinë specifike, por në një kod bitesh të pavarur nga platforma. Ky kompilator për në kod bitesh interpretohet nga Java Virtual Machine (JVM).
- E thjeshtë - Java është projektuar për të qenë e thjeshtë për tu mësuar. Nëse kuptoni konceptet bazë të programimit Object-Oriented, do e keni të thjeshtë të përvetësoni Java.
- E sigurt – Java përmban cilësi që mundësojnë programimin pa ndërhyrje të viruseve. Teknikat e autentifikimit janë të bazuara në enkriptimin me çelësa publik.
- Neutrale nga arkitektura – Kompilatori i Java gjeneron një skedar në një format neutral nga arkitektura, i cili e bën kodin e kompilueshëm dhe të ekzekutueshëm në shumë procesorë, mjafton të jetë e instaluar Java runtime system.
- Portabël – Duke qenë neutrale nga arkitektura, nuk ka asnjë varësi në implementim, gjë që e bën Java portable.
- Robust – Java përpiqet të eliminojë situatat e gabimit duke u fokusuar në kontrollimi gjatë kohës së kompilimit dhe kontrolle gjatë ekzekutimit.

- Multithread – Një cilësi i Java është mundësia për multithread, pra kryerja e disa detyrave njëherësh, në mënyrë simultane. Kjo cilësi në projektim i lejon programuesit të ndërtojnë aplikacione iterative multitask.
- E interpretueshme – Kodet e biteve në Java përkthehen në instruksione të makinës dhe nuk ruhen askund tjetër ndërkohë. Procesi i zhvillimit të kodit është i shpejtë dhe analitik, sepse link-imi kryhet si një proces inkrementues.
- Me performancë të lartë - Me përdorimin e kompilatorëve Just-In-Time, Java arrin në një performancë të lartë.
- E shpërndarë - Java është projektuar për mjedise të shpërndara në internet.
- Dinamike – Java konsiderohet si dinamike, më shumë se C ose C++, përderisa është projektuar për t'iu përshtatur mjediseve ku po ekzekutohet kodi. Programi në Java mund të mbajë sasi të gjëra informacioni run-time, i cili shërben për të verifikuar dhe për të aksesuar objektet gjatë kohës së ekzekutimit.

## Pak historik

James Gosling inicializoi projektin e gjuhës së programimit Java, në Qershor të 1991. Kjo gjuhë fillimisht u quajt Oak, që në anglisht do të thotë lis, ngaqë jashtë zyrës së Gosling ishte një peme lisi. Sun lëshoi implementimin e parë si Java 1.0. Që në këtë version, Java ishte me cilësinë Write Once Run Anywhere. Në nëntor 2006, Sun i bashkëngjiti Java-s licencën GNU General Public License, si një software free dhe open source. Në maj 2007, Sun përfundoi procesin e bërjes së komplet kodit Java, free dhe open source.

## Instalimet e mjedisit të Java

Në këtë sesion do përshkruhen instalimet incializuese të mjedisit të Java. Më poshtë jepen hapat që duhen ndjekur.

Java SE mund të shkarkohet falas në linkun e mëposhtëm :

http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

Mund të shkarkoni një version në varësi të sistemit të shfrytëzimit në kompjuterin tuaj.

Ndiqni hapat e mëposhtëm për të ekzekutuar një skedar .java në kompjuterin tuaj. Pasi të keni instaluar me sukses makinën virtuale të Java, do ju duhet të caktoni variablat e mjedisit (environment variables) të sistemit tuaj.

### Caktimi i variablave në Windows

Supozojmë se keni instaluar Java në direktorinë : `c:\Program Files\java\jdk`

- Klikoni me buton të djathtë të Mouse, tek MyComputer, dhe zgjidhni Properties.
- Klikoni tek Environment Variables, dhe tek Advanced.
- Tani, ndryshoni variablin Path, në mënyrë të tillë që të përmbajë direktorinë e instaluar, psh mund ta shtoni :

```
'C:\WINDOWS\SYSTEM32;c:\Program Files\java\jdk\bin'
```

### Editorë të Java-s

Për të shkruar një program në Java, do ju duhet një editor teksti. Variante të ndryshme janë nga editori më i thjeshtë Notepad, ose Notepad++, deri tek dy IDE falas dhe open source, si Eclipse dhe Netbeans.

### Rregullat sintaksore në Java

Një program në Java mund të konsiderohet si një koleksion objektesh që komunikojnë nëpërmjet metodave. Le të zgjerojmë më tej konceptin e klasës, objektit, metodës dhe variablave të instancës (atributeve).

- Objektet kanë gjendje dhe sjellje. Për shembull një qen ka gjendjen : ngjyrë, emër, por gjithashtu ka edhe sjelljen si lojërat dhe ushqimi. Një objekt është një instancë e një klase.
- Klasa mund të përkufizohet si një shabllon që përshkruan gjendjen dhe sjelljen e objektit.
- Metoda është një sjellje. Klasa mund të përmbajë shumë metoda. Në këto metoda shkruhet veprimet që do kryhen, të dhënat që do përpunohen dhe çdo veprim që do ekzekutohet.

- Variablat e instancës, ose ndryshe atributet përmbajnë vlerat e gjendjes të objekteve.

### Programi i parë në Java

Le të shohim një program në Java i cili afishon tekstin Hello World.

### Shembull

Le të shohim se si ruhet një skedar, si kompilohet dhe si ekzekutohet. Ndiqni hapat e mëposhtme:

- Hapni programin Notepad, dhe shkruani pjesën e kodit.
- Ruani skedarin me emërtimin ProgramiPare.java
- Hapni command prompt dhe shkoni tek direktoria ku keni ruajtur këtë klasë.
- Shtypni "`javac ProgramiPare.java`" dhe shtypni Enter për të kompiluar kodin. Në qoftë se do ketë probleme gjatë kompilimit, do ju jepet mesazh për gabimin.
- Pas kompilimit me sukses, shtypni "`java ProgramiPare`" për të ekzekutuar programin.
- Në ekran do shikoni Hello World.

### Sintaksa bazë

Gjatë programimit në Java është e rëndësishme të mbani mend këto pika:

- Shkronjat e mëdha dhe të vogla dallojnë. Kjo do të thotë se Hello dhe hello apo HeLLo janë të ndryshme.
- Emërtimet kanë specifikë, gjë që do të thotë se shkronja e parë e klasës këshillohet të jetë shkronjë e madhe. Në qoftë se emërtimi përmban disa fjalë, ato duhet të ndjekin rregullin e gamiles, shkronja e parë e çdo fjale të jetë shkronjë e madhe.
- Emërtimet e metodave kanë specifikën që fillojnë me shkronjë të vogël. Në qoftë se emërtimi ka disa fjalë, secila fjalë nis me shkronjë të madhe.
- Emërtimi i programit duhet të jetë identik me emrin e klasës publike. Kur ruani këtë skedar, duhet ta ruani në bazë të këtij rregulli, si dhe të shtoni mbaresën .java, në të kundërt nuk do kompilohet.
- `public static void main(String args[])`

është metodë që duhet të jetë pjesë e çdo klase të ekzekutueshme në Java.

### Idektifikatorët në Java

Çdo komponent në Java kërkon emërtim. Emrat e përdorur për klasat, variablat dhe metodat quhen ndryshe si identifikatorë.

Në Java duhen theksuar këto rregulla për identifikatorët.

- Çdo identifikator duhet të fillojë me shkronjë (nga A deri në Z, ose nga a deri në z), simboli $ ose simboli _
- Pas karakterit të parë, identifikatorët mund të kenë çfarëdo kombinimi me simbolet e numrave.
- Fjalët çelës të gjuhës Java nuk mund të jenë identifikatorë.
- Në identifikatorët ka dallim shkronjat e mëdha ose të vogla.
- Shembuj identifikatorësh të saktë :

`age`, `$salary`, `_value`, `__1_value`.

- Shembuj identifikatorësh të gabuar :

`123abc`, `-salary`.

### Modifikuesit në Java

Në java është e mundur të modifikohen klasat, metodat, etj, duke përdorë modifikues. Ato janë të dy kategorive :

- Modifikues aksesi: `default`, `public`, `protected`, `private`
- Modifikues të tjerë : `final`, `abstract`, `strictfp`

Më gjatë në lidhje me to do shohim në leksionet ne vazhdim.

### Variablat në Java

Në Java, variablat janë në këto tre lloje :

- Variabla lokal
- Variabla të klasës (variabla statik)
- Variabla të instancës (variabla jo statik)

### Tabelat në Java

Tabelat (matricat) janë objekte që mund të ruajnë disa variabla të të njëjtit lloj. Një tabelë është në vetvete një objekt. Mënyra e deklarimit dhe e inicializimit do përshkruhet në leksionet në vazhdim.

### Enums në Java

Enums janë prezantuar në Java 5.0. Ato janë variabla të limituar në vlerat që mund të marrin. Kjo listë me vlera quhet enums. Me këtë listë të kufizuar vlerash mund të zvogëlohen gabimet në kod.

### Fjalët çelës në Java

Më poshtë jepet lista e fjalëve çelës në Java. Këto fjalë kanë rolin specifik të tyre, dhe nuk mund të përdoren si konstante, variabla, apo si ndonjë identifikues tjetër.

| | | | | |
|---|---|---|---|---|
| abstract | assert | boolean | break | byte |
| case | catch | char | class | const |
| continue | default | do | double | else |
| enum | extends | final | finally | float |
| for | goto | if | implements | import |
| instanceof | int | interface | long | native |
| new | package | private | protected | public |
| return | short | static | strictfp | super |
| switch | synchronized | this | throw | throws |
| transient | try | void | volatile | while |

### Komentet në Java

Java përfshin dy lloj komentesh, komentet rresht dhe komentet në bllok. Kjo sintaksë për komentet ngjason me C apo me C++. Çdo karakter brenda kufijve të komenteve injorohet nga kompilatori në Java.

## Leksioni 2

## Objektet dhe klasat

Objektet dhe klasat janë dy koncepte kryesore në Java. Në botën reale mund të gjejmë shumë objekte rreth nesh, si psh automjete, persona, pizza apo ëmbëlsira. Të gjithë këto janë objekte që kanë një gjendje dhe sjellje.

Psh klasa qen, gjendja është emri, raca, ngjyra, ndërsa sjellja është duke luajtur apo duke vrapuar.

Në qoftë se krahasoni objektet software me objektet e botës reale, do vini re se kanë shumë cilësi të përbashkëta. Objektet software gjithashtu kanë gjendje dhe sjellje, gjendja ruhet në atribute, ndërsa sjellja tregohet në metoda. Në zhvillimin software, metodat veprojnë mbi gjendjen e objektit, gjithashtu objektet komunikojnë me njëri tjetrin nëpërmjet metodave.

### Klasat në Java

Klasa është një përcaktim i modelit të gjendjes dhe sjelljes së objekteve të saj.

### Shembull

```
public class Ore {
	String modeli;
	String ngjyra;

	void caktoOren() {
	}
	void caktoDaten() {

	}
}
```

### Konstruktori i klasës

Në diskutimin rreth klasave, një çështje me rëndësi është konstruktori i klasës. Çdo klasë ka konstruktorin e saj. Në qoftë se nuk përcaktoni konstruktorin në mënyrë eksplicite, kompilatori në Java ndërton konstruktorin e gatshëm të klasës.

Çdo herë që krijohet një objekt i ri, përfshihet një konstruktor. Rregulli kryesor i konstruktorit është që duhet të ketë emrin e njëjtë si emri i klasës. Gjithashtu nuk ka tip të vlerës kthyese. Një klasë mund të ketë më shumë se një konstruktor. Më poshtë jepet shembulli i një konstruktori.

### Shembull

```
public class Ore {
	public Ore() {

	}
	public Ore(String modeli) {
	// Ky konstruktor ka vetem nje parameter: modeli.
	}
}
```

### Krijimi i objekteve

Një objekt krijohet si objekt i një klase. Në këtë deklarim përdoret fjala çelës new. Krijimi i objektit përfshin këto tre hapa:

- Deklarimi – Objekti deklarohet me një emër dhe me një tip. Emri do jetë referenca e këtij objekti, ndërsa tipi është klasa ku bën pjesë objekti.
- Krijimi i instancës – Fjala çelës new krijon objektin
- Inicializimi – Fjala çelës new ndiqet nga një thirrje e konstruktorit të klasës. Kjo thirrje inicializon objektin e ri.

Më poshtë jepet një shembull i krijimit të një objekti të klasës ore.

### Shembull

```
public class Ore {
	String modeli;
	String ngjyra;
```

```
      public Ore(String modeli) {
            System.out.println("Modeli i ores eshte :" + modeli);
      }
      public static void main(String []args) {
             // Ky instruksion krijon nje ore
            Ore oraIme = new Ore( "lotus" );
      }
}
```

Në console

```
Modeli i ores eshte :lotus
```

### Variablat dhe metodat e instancës

Variablat dhe metodat e instancës aksesohen nëpërmjet objekteve të krijuar. Për këtë, duhen ndjekur hapat e mëposhtëm:

```
/* se pari krijohet nje objekt */
      ReferenceObjekti = new Constructor();

/* thirret variabli */
      ReferenceObjekti.emerVariabli;

/* thirret metoda e klases */
       ReferenceObjekti.emriMetode();
```

### Shembull:

Në këtë shembull shpjegohet se si aksesohen variablat dhe metodat e instancës të klasës.

```
public class Ore {
      String modeli;
      String ngjyra;

      public Ore(String modeli) {
            System.out.println("Modeli i ores eshte :" + modeli);
            modeli=modeli;
            ngjyra="";
      }

      public String getNgjyra() {
            return ngjyra;
      }
```

```
	public void setNgjyra(String n) {
		ngjyra = n;
	}

	public static void main(String []args) {
		/* Krijimi i objektit */
		Ore obj1 = new Ore("festina");
		/* thirrja e metodave te klases */
		obj1.setNgjyra("Kuq");
		/* Aksesimi i variablave te instances,atributit mosha */
		System.out.println("Vlera e variablit :"
+obj1.getNgjyra() );
	}
}
```

Në console:

```
Modeli i ores eshte :festina
Vlera e variablit :Kuq
```

### Rregulla të skedarëve kod në java

Më poshtë le të shohim rregullat e deklarimit të skedarëve kod në java. Këto rregulla janë të rëndësishme në deklarimin e klasave si dhe në instruksionet e importimit të klasave dhe paketave.

- Në një skedar kodi në java, ka vetëm një klasë publike.
- Një skedar në java mund të ketë disa klasa me akses jo publik.
- Emri i skedarit në java duhet të jetë i njëjtë me emrin e klasës publike. Psh, skedari që përmban klasën publike Nenpunes do ketë emrin Nenpunes.java.
- Në qoftë se klasa është përcaktuar brenda një pakete, atëherë deklarimi i paketës duhet të jetë instruksioni i parë në këtë skedar në java.
- Në qoftë se klasa përmban deklarimi importimi të paketave, ato duhen shkruar mes deklarimit të paketës dhe deklarimit të klasës.
- Importimi i paketave do përfshijë çdo klasë që deklarohet në skedarin java. Nuk është e mundur të deklarohen importime të ndryshme për klasa të ndryshme në të njëjtin skedar java.

Klasat kanë disa nivele aksesimi, dhe tipa të ndryshme të klasave, siç janë klasat abstrakte, klasa finale, etj. Këto koncepte do shpjegohen në leksionet e ardhshme. Gjithashtu në Java ekzistojnë edhe klasa të veçanta siç janë klasat e brendshme dhe klasat anonime.

### Paketat në Java

Paketa është një mënyrë kategorizimi i klasave dhe ndërfaqeve. Meqë gjatë zhvillimit të aplikacioneve në Java, do të shkruhen dhjetëra klasa dhe ndërfaqe, atëherë kategorizimi i klasave do jetë një domosdoshmëri për organizimin e punës.

### Instruksionet e importimit të klasave

Për importimin e klasave, duhet dhënë emri i plotë i klasës, gjë që përfshin paketën dhe klasën që do përfshihet në kod. Në këtë mënyrë, gjendet thjeshtë kodi i klasës që do importohet. Instruksionet e importimit të klasave është shënim për kompilatorin që i duhet të lokalizojë klasën e kërkuar. Për shembull, do kërkohet importimi i klasave të paketës io.

```
import java.io.*;
```

Një importim i tillë përfshin simbolin * i cili i tregon kompilatorit se do importohet e gjithë paketa.

### Rast studimi : Klasa Nenpunes

Në këtë rast studimi, do krijohen dy klasa: klasa Nenpunes dhe klasa NenpunesTest. Kujtoni që klasa Nenpunes do ruhet ne skedarin Nenpunes.java, ndërsa klasa NenpunesTest do ruhet në skedarin NenpunesTest.java. Klasa Nenpunes do ketë katër variabla të instancës: emri, mosha, departamenti dhe paga. Kjo klasë ka një konstruktor që merr një parametër.

```
import java.io.*;
public class Nenpunes {
   String emri;
   int mosha;
   String departamenti;
   double paga;
```

```
   public Nenpunes(String e) { // Konstruktori i klases nenpunes
      this.emri = e;
   }
   // Caktimi i variablit te moshes per nenpunesin
   public void nMosha(int m) {
      mosha=m;
   }
   /* Caktimi i variablit departamenti*/
   public void nDepartamenti(String d) {
      departamenti=d;
   }
   public void nPaga(double p) {  /* Caktimi i variablit paga */
      paga=p;
   }
   /* Afishimi i te dhenave te nenpunesit */
   public void afisho() {
      System.out.println("Emri:"+ emri );
      System.out.println("Mosha:" + mosha );
      System.out.println("Departamenti:" + departamenti);
      System.out.println("Paga:" + paga);
   }
}
```

Siç është përmendur më lart, ekzekutimi i programit nis nga metoda main. Për të ekzekutuar klasën Nenpunes, duhet te shkruhet një metodë main e cila krijon objekte të klasës Nenpunes dhe thërret metodat e saj. Në vijim po krijojmë klasën NenpunesTest e cila do krijojë dy objekte të klasës Nenpunes dhe do thërrasë metodat për secilin objekt.

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

```
        /* Krijimi i objekteve nenpunes */
        Nenpunes n1 = new Nenpunes("James Smith");
        Nenpunes n2 = new Nenpunes("Mary Anne");
        // Thirrja e metodave per objektet qe jane krijuar
        n1.nMosha(26);
        n1.nDepartamenti("Applications");
        n1.nPaga(1000);
        n1.afisho();
        n2.nMosha(21);
        n2.nDepartamenti("Software Engineer");
        n2.nPaga(500);
        n2.afisho();
    }
}
```

Në console

```
Emri:James Smith
Mosha:26
Departamenti:Senior Software Engineer
Paga:1000.0
Emri:Mary Anne
Mosha:21
Departamenti:Software Engineer
Paga:500.0
```

## Leksioni 3

## Tipat bazë të të dhënave

Të dhënat e një programi bazohen në konceptin e variablit. Një variabël është një vendndodhje e memories e rezervuar për të ruajtur vlera të një tipi të caktuar. Kjo do të thotë se kur krijohet një variabël, rezervohet një hapësirë memorie, e cila do identifikohet me emrin

e variablit. Duke pasë të dhëna të tipave të ndryshme, në memorie mund të ruhen numra të plotë, numra me presje apo të dhëna karakter.

Në Java ka dy lloje të dhënash :

- Tipa të dhënash primitive
- Tipa të dhënash referencë/objekt

### Tipat primitive të të dhënave

Gjuha e programimit Java suporton tetë tipa primitive të të dhënave. Këto tipa janë të paracaktuar, dhe emërtuar me fjalë çelës të gjuhës. Më poshtë jepen detaje për secilin tip primitiv.

#### byte

- Tipi i të dhënave byte zë 8 bit memorie, dhe ruan të dhëna numra të plotë.
- Vlera minimale është -127, dhe vlera maksimale është 127. Vlera default është zero.
- Ky tip përdoret zakonisht për të kursyer hapësira memorieje në tabela një ose disa përmasore.
- Shembull : `byte a=100; byte b= -50;`

#### short

- Tipi i të dhënave short është zë 16 bit në memorie, dhe ruan të dhëna numra të plotë.
- Vlera minimale që ruan tipi short është -32,768 ndërsa vlera maksimale është 32,767. Vlera default është zero.
- Tipi i të dhënave short gjithashtu përdoret për të ruajtur hapësirë memorieje, sepse nxë relativisht pak hapësirë.
- Shembull : `short s=10000; short r=-20000;`

#### int

- Tipi i të dhënave int zë hapësirë 32 bit për çdo vlerë që ruan, të cilat janë numra të plotë.
- Vlera minimale që ruan int është -2,147,483,648. Vlera maksimale që ruan int është 2,147,483,647. Vlera default është zero.
- Tipi i të dhënave int përdoret shpesh kur nuk ka probleme kursimi të memories.

- Shembull : `int a=100000; int b=-200000;`

### long

- Tipi long kërkon 64 bit hapësirë memorieje për çdo numër të plotë që ruan.
- Vlera minimale që mban ky tip është -9,223,372,036,854,775,808, ndërsa vlera maksimale është 9,223,372,036,854,775,807. Vlera default është 0L.
- Ky tip të dhëne përdoret kur nevojitet ruajtja e numrave më të mëdhenj se sa mundëson tipi int.
- Shembull : `long a=100000L; long b=-2000000L;`

### float

- Tipi i të dhënave float ruan numra me presje. Ky tip kërkon 32 bit hapësirë për çdo vlerë.
- Float përdoret në rastet kur të dhënat janë numra me presje, dhe nuk përdoret asnjëherë kur do ruhen vlera precize si vlera çmimesh.
- Vlera default e float është 0.0f
- Shembull : `float f1=234.4f;`

### double

- Tipi double ruan të dhëna numra me presje, me precizon dyfish, dhe nxë 64 bit memorie për çdo vlerë.
- Ky tip zakonisht është tipi që ruan të dhënat për numra me presje. Ai nuk përdoret asnjëherë në rastet kur do ruhen vlera të tilla si monedhat.
- Vlera default është 0.0d
- Shembull: `double d1=123.4;`

### boolean

- Tipi i të dhënave boolean përdor një bit memorie për të ruajtur të dhënën, e cila mund të jetë vetëm true ose false.
- Ky tip të dhënash përdoret për shënime të thjeshta me vlerat logjike e vërtetë ose e gabuar.
- Vlera default është false.

- Shembull : `boolean one=true;`

**char**

- Tipi i të dhënave char ruan të dhëna karakter, në Unicode, dhe përdor 16 bit memorie.
- Vlera minimale është \u0000 ndërsa vlera maksimale është \uffff
- Ky tip ruan të dhëna karakter.
- Shembull : `char shkronjaA='A';`

### Tipat e të dhënave referencë

- Variablat referencë krijohen duke përdorë konstruktorët e klasave. Ata përdoren për të aksesuar objektet. Këto variabla deklarohen me një tip specifik që nuk ndryshon më vonë.
- Vlera default e çdo reference është null.
- Një variabël referencë mund të përdoret për të shënuar tek çdo objekt i tipit me të cilin është deklaruar variabli, ose çdo tip tjetër kompatibel me te.
- Shembull : `Ore obj1 = new Ore("festina");`

### Literal në Java

Një literal është një vlerë fikse në kod, e cila përfaqësohet direkt, pa llogaritje. Literal mund të caktohen për çdo tip primitiv. Për shembull :

```
byte a = 68;
char a = 'A';
```

Byte, int, long dhe short mund të shprehet në bazë 10, në bazë 16 ose në bazë 8. Prefiksi 0 përdoret për të treguar octal, prefiksi 0x tregon literal në hekzadecimal. Për shembull :

```
int decimal = 100;
int octal = 0144;
int hexa =  0x64;
```

Literalët string përcaktohen brenda thonjëzave, që përfshijnë brenda tyre radhë karakteresh. Për shembull :

```
"Hello World"
"dy\nrreshta"
```

```
"\"Kjo eshte brenda thonjzave\"
```

Java përmban edhe simbole të veçanta që paraqiten në tabelën më poshtë:

| Shënimi | Karakteri që përfaqëson | Shënimi | Karakteri që përfaqëson |
|---|---|---|---|
| \n | Rresht i ri | \t | Tab |
| \r | Kthim në fillim të rreshtit | \" | Thonjëzat |
| \b | Fshirje një karakter më para | \' | Apostrof |
| \s | Hapsirë bosh | \\ | Backslash |

### Variablat

Çdo variabël në Java ka një tip i cili përcakton madhësinë e memories që nxë, bashkësinë ku do marrë vlerë variabli si dhe veprimet që mund të kryhen me këtë variabël. Çdo variabël duhet deklaruar në fillim, para se të përdoret. Sintaksa e deklarimit përcakton tipin e variablit dhe emrin e variablit. Në deklarim mund të përfshihet edhe inicializimi i variablit. Për te deklaruar më shumë se një variabël, me një tip të përbashkët, emrat ndahen me presje dhe në këtë mënyrë listohen variablat e rinj të sapo-deklaruar. Më poshtë jepen shembuj deklarimi të variablave.

### Shembull

```
int a, b, c;         // Deklaron tre variabla nr te plote
int a = 10, b = 10;  // Shembull inicializimi
byte B = 22;         // Inicializon variablin B te tipit byte
double pi = 3.14159; // Deklaron dhe inicializon vleren e PI.
char a = 'a';        // Variabli karakter inicializohet 'a'
```

Një klasë përmban këto lloj variablash:

- *Variabla lokal*. Variablat që deklarohen brenda metodave, konstruktorëve apo blloqeve quhen ndryshe edhe variabla lokal. Variabli deklarohet dhe inicializohet brenda metodës, dhe me mbarimin e metodës, variabli fshihet.

- *Variabla të instancës*. Variablat që deklarohen brenda klasës por jashtë çdo metode, quhen variabla të instancës, ose ndryshe edhe atribute. Këto variabla inicializohen kur inicializohet dhe klasa. Variablat e instancës mund të aksesohen nga çdo metodë, konstruktor ose bllok brenda klasës.
- *Variablat e klasës*. Variablat e deklaruar brenda një klase, jashtë çdo metode dhe me fjalën çelës static, quhet variabël klase.

### Variablat lokal

Variablat lokal deklarohen brenda një metode, konstruktorit ose brenda një blloku instruksionesh. Ato përdoren vetëm në zonën brenda të cilës janë deklaruar. Nqs janë deklaruar brenda një metode, atëherë mund të përdoren vetëm brenda saj. E njëjta rregull vlen për zonën e konstruktorit apo të një blloku instruksionesh. Variablat lokal fshihen sapo mbaron ekzekutimimi i instruksioneve të zonës ku është deklaruar variabli. Për këto variabla nuk funksionojnë rregullat e aksesimit, sepse ato njihen vetëm brenda kësaj zone ku janë deklaruar. Këto variabla nuk kanë një vlerë të gatshme (default), prandaj duhen deklaruar dhe inicializuar pa nisë përdorimi i tyre.

### Shembull

Në shembullin e mëposhtëm, variabli modeli është një variabël lokal i deklaruar brenda metodës modeliIOres, prandaj zona e këtij variabli është kjo metodë. Në qoftë se ky variabël nuk inicializohet, do afishohet mesazh gabimi gjatë kohës së kompilimit.

```
public class TestVariables {
        public void modeliIOres() {
           String modeli="";
           modeli="Verde";
           System.out.println("Modeli i ores eshte eshte : " +
modeli);
        }
        public static void main(String args[]) {
           TestVariables test = new TestVariables();
           test.modeliIOres();
        }
}
```

Në console:

```
Modeli i ores eshte eshte : Verde
```

### Variablat e instancës

Variablat e instancës deklarohen brenda klasës, por jashtë çdo metode apo konstruktori. Zakonisht deklarohen menjëherë pas deklarimit të klasës. Kur deklarohet një objekt i kësaj klase, rezervohet hapësirë memorieje për çdo variabël të instancës. Variablat e instancës krijohen kur krijohet objekti i klasës duke përdorë fjalën çelës new, dhe fshihet me objektin e krijuar. Këto variabla ruajnë vlerat e instancës së objektit, që përfaqësojnë gjendjen e objektit, atributet e tij. Është e rekomandueshme që këto variabla të jenë të përcaktuar si privat. Këto variabla kanë vlera default. Për numrat, vlera e gatshme është zero, vlera e gatshme për variablat boolean është false, vlera default për objektet është null.

### Shembull

```
import java.io.*;
public class Nenpunes {
   // ky eshte nje variabel instance, me akses publik
   public String emri;
   // ky eshte nje variabel instance me akses privat
   private double paga;
   // emri percaktohet ne konstruktor
   public Nenpunes (String e) {
      emri = e;
   }
   // kjo metode cakton nje vlere per pagen e punonjesit
   public void setPaga(double p) {
      paga = p;
   }
   // kjo metode afishon te dhenat
   public void print() {
      System.out.println("emri : " + emri );
```

```
        System.out.println("paga :" + paga);
    }
    public static void main(String args[]) {
        Nenpunes nenpunes1 = new Nenpunes("Ransika");
        nenpunes1.setPaga(1000);
        nenpunes1.print();
    }
}
```

Në console :

```
emri : Ransika
paga :1000.0
```

### Variablat statik

Variablat e klasës, ose ndryshe variablat statik, deklarohen me fjalën çelës statik, brenda një klase, por jashtë çdo metode apo konstruktori. Ato do jenë vetëm një kopje për çdo klasë, pavarësisht se sa objekte krijohen. Variablat statik shpesh deklarohen si konstante, pra me fjalën çelës final dhe static, gjë që pengon ndryshimin e vlerës fillestare.

Variablat statik ruhen në memorie statike në mënyrë të pavarur nga ndonjë objekt i klasës. Ato krijohen sapo starton programi, dhe fshihen kur ndalon ekzekutimi i programit. Zakonisht këto variabla deklarohen publik, që te aksesohen nga klasat e tjera. Rregullat për vlerën default të tyre janë njësoj si për variablat e instancës. Këto variabla aksesohen nëpërmjet emrit të klasës.

### Shembull

```
import java.io.*;
public class Nenpunes {
   // variabli nr total i nenpunesve si variabel static
   private static int nrTotal;
   // variabli departamenti si variabel konstant dhe static
```

```
   public static final String DEPARTMENT = "Development ";
   public static void main(String args[]) {
      nrTotal = 22;
      System.out.println(DEPARTMENT + "ka nr total punonjesish:" +
nrTotal);
   }
}
```

Në console :

```
Development ka nr total punonjesish:22
```

### Rregullat e aksesimit dhe modifikues të tjerë në Java

Në Java përdoren modifikues, siç janë modifikuesit e aksesit. Për të përdorë këto modifikues, duhet shtuar fjala çelës që e përcakton atë. Modifikuesit e aksesit përcaktojnë nivelin e aksesimit për klasën, variablin, metodën apo konstruktorin. Në Java përdoren katër nivele aksesimi :

- Akses brenda paketës, kur nuk është vendosur asnjë modifikues.
- Akses privat, vetëm brenda klasës, kur vendoset fjala çelës private.
- Akses publik, për çdo klasë, në cilëndo paketë, kur vendoset fjala çelës public.
- Akses brenda paketës dhe klasave të derivuara, kur vendoset fjala çelës protected.

Modifikuesit e tjerë janë me funksionalitete të ndryshme. Më poshtë listohen disa prej tyre:

- Modifikuesi static përdoret në krijimin e metodave statike ose variablave statik.
- Modifikuesi final përdoret në deklarimin e konstanteve, qe mund te jenë klasa, metoda ose variabla.
- Modifikuesi abstract përdoret për krijimin e klasave ose metodave abstrakte.
- Modifikuesi synchronized dhe modifikuesi volatile përdoren për thread.

# Leksioni 4

# Operatorët në Java

Java ofron një set të gjerë operatorësh, të cilët mund ti ndajmë në këto grupe:

- Operatorët aritmetikë
- Operatorët relacionalë
- Operatorët e biteve
- Operatorët logjik
- Operatorët e vlerëdhënies
- Operatorë të tjerë

## Operatorët aritmetikë

Operatorët aritmetikë përdoren në shprehje matematike në mënyrë të ngjashme siç përdoren në algjebër. Në tabelën e mëposhtme jepen këto operatorë me përshkrimet përkatëse.

| Operatori | Përshkrimi |
|---|---|
| **+ (mbledhje)** | Mbledh vlerat e operandëve në dy anët e operatorit |
| - (zbritje) | Zbret vleren e operandit në të djathtë nga operandi në të majtë |
| * **(shumëzim)** | Shumëzon vlerat në të dy anët e operatorit |
| **/ (pjesëtim)** | Pjesëton vlerën në të majtë me vlerën në të djathtë të operatorit |
| % **(modul)** | Pjesëton vlerën në të majtë me vlerën në të djathtë dhe merr pjesën që mbetet |
| **++ (rritje)** | Rrit vlerën e operandit me një njësi |
| **-- (zvogëlim)** | Zvogëlon vlerën e operandit me një njësi |

### Operatorët relacionalë

Operatorët relacionalë veprojnë mbi dy vlera numerike dhe kthejnë si rezultat një vlerë logjike, në varësi të relacionit mes operandëve. Në tabelën më poshtë jepen këto operatorë me përshkrimet përkatëse.

| Operator | Përshkrim |
|---|---|
| `== (e barabartë)` | Kthen vlerën true vetëm nqs dy operandët janë të barabartë |
| `!= (e ndryshme)` | Kthen vlerën true vetëm nqs dy operandët janë të ndryshëm |
| `> (më e madhe)` | Kthen vlerën true vetëm nqs operandi majtas është me i madh se operandi djathtas |
| `>= (më e madhe, e barabartë)` | Kthen vlerën true vetëm nqs operandi majtas është më i madh ose i barabartë me operandin djathtas |
| `< (më e vogël)` | Kthen vlerën true vetëm nqs operandi majtas është me i vogël se operandi djathtas |
| `<= (më e vogël, e barabartë)` | Kthen vlerën true vetëm nëse operandi majtas është me i vogël ose i barabartë me operandin djathtas. |

### Operatorët e biteve

Java përcakton një set operatorësh që veprojnë në numra të plote, ose karaktere, të konvertuar në paraqitjen binare të tyre. Mbi to kryen veprimet me bite. Operatorët e biteve jepen në tabelën më poshtë.

| Operatori | Përshkrimi |
|---|---|
| `& (dhe)` | Ky operator kryen veprimin dhe mes biteve |
| `\| (ose)` | Ky operator kryen veprimin ose mes biteve |

| | |
|---|---|
| ^ (xor) | Ky operator kryen veprim ose përjashtuese mes biteve |
| ~ (komplementar) | Ky operator kthen bitin komplementar te bitit të dhënë |
| << (shift majtas) | Ky operator zhvendos në të majtë me numrin e përcaktuar nga operandi djathtas |
| >> (shift djathtas) | Ky operator zhvendos bitet në të djathtë, me numrin e përcaktuar nga operandi djathtas |
| >>> (shift djathtas, shton zero) | Ky operator bën zhvendosje në të djathtë, por vlerën bosh e plotëson me zero. |

### Operatorët logjik

Operatorët logjik në Java janë të ngjashëm me atë të algjebrës së Bool. Dy operandët në këto raste janë vlera logjike, ose krahasime mes vlerash që gjithashtu kthejnë një vlerë logjike. Këto operatorë paraqiten në tabelën e mëposhtme.

| Operatori | Përshkrimi |
|---|---|
| && (dhe) | Ky operator kthen vlerën true kur të dy pohimet janë të vërteta |
| \|\| (ose) | Ky operator kthen vlerën true kur të paktën njëri pohim të jetë i vërtetë |
| ! (mohim) | Ky operator unar konverton true në false dhe false në true. |

### Operatorët e vlerëdhënies

Këto operatorë i japin vlerë operandit në të majtë, i cili duhet të jetë vetëm variabël. Nuk mund të pranohet një shprehje apo konstante matematike si vlerë në të majtë të këtyre operatorëve. Këto operanda jepen në detaje në tabelën më poshtë.

| Operatori | Përshkrimi |
|---|---|
| = | Ky operator i cakton variablit majtas vleren rezultat djathtas |

| | |
|---|---|
| += | Ky operator mbledh vleren e variablit majtas me vleren në të djathtë, dhe ia jep vlerë të re variablit majtas |
| -= | Ky operator zbret vleren e variablit majtas me vleren në të djathtë, dhe ia jep vlerë të re variablit majtas |
| *= | Ky operator shumëzon vleren e variablit majtas me vleren në të djathtë, dhe ia jep vlerë të re variablit majtas |
| /= | Ky operator pjesëton vleren e variablit majtas me vleren në të djathtë, dhe ia jep vlerë të re variablit majtas |
| %= | Ky operator pjesëton vleren e variablit majtas me vleren në të djathtë, dhe pjesën e plotë ia jep vlerë të re variablit majtas |
| <<= | Ky operator bën zhvendosje majtas dhe ia cakton vlerë të re variablit në të majtë |
| >>= | Ky operator bën zhvendosje djathtas dhe ia cakton vlerë të re variablit në të majtë |
| &= | Ky operator kryen veprimin Dhe mes biteve, dhe ia jep vlerë variablit në të majtë |
| ^= | Ky operator kryen veprimin Ose përjashtuese mes biteve, dhe ia jep vlerë variablit në të majtë |
| \|= | Ky operator kryen veprimin Ose mes biteve, dhe ia jep vlerë variablit në të majtë |

### Operatorë të tjerë

### Operatori i kushtëzimit

Ky operator njihet ndryshe si operatori trior. Ai konsiston në tre operanda, i pari është një shprehje logjike, në bazë të vlerësimit të të cilës, kthen përgjigje operatori. Nqs shprehja është e vërtetë kthehet si përgjigje vlera e parë, përndryshe kthehet vlera e dytë. Sintaksa e tij është :

```
x = (shprehje logjike) ? vlera1_ : vlera2
```

### Strukturat e kontrollit

Java përdor të gjitha instruksionet e kontrollit të ekzekutimit të gjuhës C, prandaj nëse keni programuar në gjuhët C ose C++ atëherë do jeni mësuar me këto instruksione. Shumë gjuhë programimi kanë instruksione degëzimi të ngjashme. Java përdor :

- `if – else`
- `while`
- `do-while`
- `for`
- `switch`

Të gjithë instruksionet e kushtëzuara përdorin një kusht logjik për të degëzuar ekzekutimin e programit. Një shembull është shprehja e kushtëzuar

`A==B`

Kjo shprehje përdor një operator relacional "= =" për të parë nëse vlera e A-së është njëlloj me vlerën e B-së. Kjo shprehje kthen si vlerë "true" ose "false" , pra e vërtetë ose e gabuar. Secili prej operatoreve relacionalë mund të përdoret për të prodhuar një shprehje logjike. Vini re që në Java nuk mund të përdorni një numër si vlerë logjike. Në C dhe në C++ kjo gjë lejohet. Në C++ një numër jozero merret dhe si vlerë "true" dhe zero merret si "false". Nqs doni të përdorni një vlerë jo logjike në një kusht logjik, në fillim duhet ta konvertoni atë në një vlerë logjike, duke përdorë një instruksion kushtëzimi si psh `if (a!=0)`

### Struktura if – else

Instruksioni i kushtëzimit është instruksioni bazë për të komanduar radhën e ekzekutimit të programit. Instruksioni else është opsional, pra ju mund ta përdorni ose jo atë. Dy format e këtij instruksioni janë :

```
if (shprehje-logjike)
     instruksion
```

dhe

```
if (shprehje-logjike)
```

```
     instruksion1
else
     instruksion2
```

Shprehja logjike duhet të prodhojë një rezultat logjik (boolean). Instruksioni është ose një instruksion i thjeshtë që mbaron me një simbol ";" ose mund të jetë një bllok instruksionesh që kufizohet nga dy kllapat '{' dhe '}'. Çdo herë me fjalën instruksion do nënkuptojmë një instruksion të vetëm ose një bllok instruksionesh. Është mirë që gjatë shkrimit të kodit të zhvendosemi disa hapësira nga e djathta kur kemi një instruksion brenda kushtit, në mënyrë që

të dallojmë se ku nis dhe ku mbaron instruksioni i kushtëzuar.

### Strukturat ciklike

Tre ciklet që përdoren në Java janë : while, do – while dhe for. Ata bëjnë të mundur përsëritjen e instruksionit në bazë të një kushti logjik. Kur ky kusht ka vlerën " false" atëherë instruksioni brenda ciklit nuk ekzekutohet më. Cikli while ka formën :

```
while (shprehje-logjike)
     instruksion
```

Shprehja logjike vlerësohet në fillim të çdo iteracioni të ciklit.

Cikli do- while ka formën :

```
     do
        instruksion
     while (shprehje-logjike)
```

Diferenca e vetme mes while dhe do-while është se instruksioni në ciklin do-while do ekzekutohet të paktën një herë edhe kur shprehja ka vlerën "false". Kjo sepse në ciklin while nqs kushti është "false" nuk hyn asnjëherë në cikël, kurse për do-while kushti vlerësohet në fund të ekzekutimit të instruksionit.

Cikli for performon një inicializim para iteracionit të parë. Më pas paraqet një kontroll dhe në fund të çdo iteracioni ka një rritje hapi. Forma e këtij cikli është :

```
for (inicializim;shprehje-logjike;hapi)
            instruksion
```

Secila prej zonave, inicializimi, shprehja-logjike ose hapi mund të jenë edhe bosh. Shprehja testohet para çdo iteracioni, dhe sapo të rezultojë "false" ekzekutimi do ti kalohet instruksionit pasardhës të ciklit for. Në fund të çdo iteracioni ekzekutohet "hapi". Cikli for në përgjithësi përdoret kur e dimë numrin e përsëritjeve të ciklit. Ju mund të përcaktoni disa variabla brenda ciklit for, por këto variabla duhet të jenë të të njëjtit tip. Psh :

```
for (int i = 0, j = 1; i < 10 && j != 11; i++, j++)
          // trupi i ciklit
```

Deklarimi int brenda for mbulon dhe variablin i dhe variablin j. Kjo është veçanti e ciklit for dhe jo e të gjithë cikleve të tjerë.

### Instruksionet break dhe continue

Instruksioni break realizon daljen para kohe nga cikli pa ekzekutuar instruksionet që mbeten. Instruksioni continue realizon ndërprerjen e ekzekutimit nga iteracioni i ciklit dhe vazhdon me ekzekutimin e hapit pasardhës.

### Instruksioni switch

Instruksioni switch klasifikohet si një instruksion zgjedhjeje. Një instruksion switch zgjedh cilën pjesë të kodit të ekzekutojë në varësi të vlerës që ka shprehja që shqyrton.

Switch ka formën :

```
switch(shprehje) {
     case vlera1 :
          instruksion;
          break;
     case vlera2 :
          instruksion;
          break;
     case vlera3 :
          instruksion;
          break;
     case vlera4 :
          instruksion;
          break;
     case vlera5 :
          instruksion;
          break;
          // ...
     default:
          instruksion;
```

```
}
```

Instruksioni switch krahason vlerën e shprehjes me vlerat përkatese për çdo case. Nqs gjen një vlerë që i korrespondon atëherë ekzekuton instruksionet e atij blloku të vlerës përkatëse. Përndryshe ekzekutohet instruksioni default. Vini re përdorimin e instruksionit break në fund të çdo case. Kjo realizon kalimin e ekzekutimit në fund të switch. Përdorimi i break është opsionale, por shpesh ndërtohet në këtë mënyrë. Instruksioni i fundit default nuk ka një break sepse nuk do kishte ndonjë efekt , gjithsesi ekzekutimi do vazhdonte në pikën ku mbaron switch. Ky instruksion është metoda më e mirë për të implementuar një zgjedhje shumëvlereshe, por kërkon një vlerë të tillë si int ose char.

# Leksioni 5

# Klasat e numrave, karakter, dhe String

## Klasat e numrave

Zakonisht për të ruajtur të dhëna numerike, përdoren variabla që i përkasin tipave primitive, si int, float, double, etj.

### Shembull

```
int i = 5000;
float gpa = 13.65;
double mask = 0xaf;
```

Java ofron edhe klasat përkatëse për secilin tip primitiv. Emërtimet këtyre klasave përkojnë me tipat primitive : Integer, Long, Byte, Double, Float, Short. Të gjitha këto klasa janë nënklasa të klasës përfshirëse Number.

Objekti i një klasë numerike përmban numrat respektive të tipit primitiv. Gjatë përdorimit të këtyre klasave, mjafton të jepni si argument konstruktorit numrin përkatës, dhe do keni objekt që ka si vlerë po atë numër. Klasa Number dhe nënklasat e saj janë pjesë e paketës `java.lang`.

### Shembull

```
public class KlasatNumerike {
      public static void main(String args[]) {
        Integer x = 5; // vendos int ne nje objekt te klases Integer
        x =  x + 10;   // kthen ne int nga objekti Integer
        System.out.println(x);
      }
}
```

Në console :

```
15
```

### Klasa karakter

Gjatë punës me elemente karakter, zakonisht përdoret tipi primitiv char.

### Shembull

```
char ch = 'a';
// Unicode shkronjen e madhe Greek omega
char uniChar = '\u039A';
// tabele me karaktere
char[] charArray ={ 'a', 'b', 'c', 'd', 'e' };
```

Java ofron një klasë të posatshme Character, që përmban objekte me të dhënat primitive karakter. Klasa Character ofron një numër metodash për të modifikuar të dhëna karakter. Një objekt i kësaj klasa krijohet nëpërmjet konstruktorit të klasës, që merr si parametër një karakter.

```
Character ch = new Character('a');
```

Kompilatori në Java krijon një objekt karakter edhe në raste kur duhet paraqitur të dhënat si objekt dhe jo si primitivë, ai e konverton automatikisht në objekt të Character.

### Shembull

```
// primitiva char 'a'
// i vendoset nje objekti te klases Character
Character ch = 'a';
// primitiva 'x' kalon si objekt si parameter per metoden test,
// vlera kthyese do jete primitiva char 'c'
char c = test('x');
```

Më poshtë jepen metodat e klasës Character.

| Metoda | Përshkrimi |
|---|---|
| `isLetter ( )` | Përcakton nëse karakteri me të cilin thirret është shkronjë. |
| `isDigit ( )` | Përcakton nëse karakteri me të cilin thirret është shkronjë. |
| `isWhitespace( )` | Përcakton nëse karakteri me të cilin thirret është hapsire. |
| `isUpperCase( )` | Përcakton nëse karakteri me të cilin thirret është shkronjë e madhe |
| `isLowerCase( )` | Përcakton nëse karakteri me të cilin thirret është shkronjë e vogël |
| `toUpperCase( )` | Kthen si përgjigje shkronjën e madhë të karakterit me të cilen është thirrë |
| `toLowerCase( )` | Kthen si përgjigje shkronjën e vogël të karakterit me të cilën është thirrë |
| `toString ( )` | Kthen një objekt String që përfaqson karakterin me të cilin u thirr. |

### Klasa String

Radhët e karaktereve, ose ndryshe String, përdoren gjerësisht në programim. Në Java ato trajtohen si objekte. Java ofron klasën String për te kryer veprime me radhë karakteresh.

### Krijimi i objekteve String

Mënyra më e thjeshtë për të krijuar objekte String, është përcaktimi i një vlere, psh :

```
String prsh = "Pershendetje!";
```

Kurdo që kompilatori ndesh një radhë karakteresh, ai e konverton në një objekt String, në këtë rast radhën "Pershendetje".

Si me çdo objekt tjetër, mund të krijohen objekte String duke përdorë fjalën çelës new, si dhe konstruktorin e klasës. Klasa String ka 11 konstruktorë, që ofrojnë mundësi krijimi String nga disa burime të ndryshme, si psh radhë karakteresh.

### Shembull

```
public class StringDemo {
        public static void main(String args[]) {
           char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.' };
           String helloString = new String(helloArray);
           System.out.println( helloString );
        }
}
```

Në console :

```
hello.
```

### Gjatësia e stringes

Metoda që përdoret për të numëruar gjatësinë e objektit të klasës String është metoda `length ( )`. Kjo metodë kthen si vlerë numrin e karaktereve që mban stringa e dhënë.

### Shembull

```
public class StringDemo {
```

```
        public static void main(String args[]) {
           String palindrome = "Dot saw I was Tod";
           int len = palindrome.length();
           System.out.println( "String Length is : " + len );
        }
}
```

Në console :

```
String Length is : 17
```

### Lidhja e stringave

Klasa String përmban një metodë për lidhjen e dy stringave:

```
string1.concat(string2);
```

Kjo metodë concat kthen si vlerë stringën e re, bashkim të dy stringave. Kjo metodë mund të përdoret edhe për radha literal :

```
"My name is ".concat("Ruth");
```

Një mënyrë e thjeshtë për te bashkuar stringat është përdorimi i operatorit +

```
"Hello," + " world" + "!"
```

që në fakt i bashkon këto stringa në :

```
"Hello, world!"
```

### Formatimi i stringave

Metodat printf() dhe format() afishojnë rezultatin të formatuar. Klasa String ka metodën e saj format () e cila kthen si rezultat një objekt String.

Përdorimi i metodës format lejon ripërdorimin e stringes së formatuar, ndryshe nga afishimi vetëm njëherë i saj.

### Shembull

Në vend të :

```
System.out.printf("Vlera e numrit me presje eshte  " +
                  "%f, ndersa vlera e numrit te plote eshte  " +
                  "variabli %d, dhe stringa " +
                  "is %s", 2.33, 63, "a2d423s");
```

mund të shkruani :

```
String fs;
fs=String.format("Vlera e numrit me presje eshte  " +
                  "%f, ndersa vlera e numrit te plote eshte  " +
                  "variabli %d, dhe stringa " +
                  "is %s", 2.33, 63, "a2d423s");
System.out.println(fs);
```

dhe variablin fs mund ta ripërdorni përseri në kod.

## Leksioni 6

## Metodat

Një metodë në Java është një set instruksionesh të grupuara për të performuar një veprim apo përllogaritje. Në këtë leksion do trajtohen metodat më gjerësisht, duke krijuar metoda të reja, të cilat mund të kenë parametra mbi të cilët kryejnë veprimet.

### Krijimi i një metode

Jepet shembulli i mëposhtëm i cili paraqet sintaksën e një metode :

```
    public static int emriMetodes(int a, int b) {
            // instruksionet
               return 1;
    }
```

Në këtë shembull vëmë re :

- public static - është modifikues që përcakton se kjo metodë ka akses public dhe është metodë statike. Këto modifikues janë opsional
- int – tipi i vlerës kthyese. Kur metoda nuk kthen asnjë rezultat, atëherë dalja shënohet void.

- emriMetodes – emri i metodës, i cili zakonisht duhet të jetë orientues për veprimet që kryen metoda.
- a, b – parametrat për metodën. Kur metoda nuk merr parametra, atëherë brenda kllapave të rrumbullakëta nuk ka asnjë shënim.

### Shembull

Kodi i mëposhtëm përmban një metodë maximum, e cila merr dy parametrat n1 dhe n2 dhe kthen si rezultat më të madhin mes tyre.

```
/** kjo metode kthen me te madhin mes dy numrave te dhene */
     public static int maximum(int n1, int n2) {
        int max;
        if (n1 > n2)
           max = n2;
        else
           max = n1;
        return max;
     }
```

### Thirrja e një metode

Për të ekzekutuar instruksionet e një metode, duhet thirrë metoda. Ka dy mënyra si mund të thirret një metodë: ndryshe thirret kur metoda kthen një vlerë, dhe ndryshe kur metoda nuk kthen vlerë. Procesi i thirrjes së metodës është i thjeshtë : Kur programi thërret metodën, kontrolli i ekzekutimit i kalon metodës së thirrur. Kjo metodë ekzekuton instruksionet e saj, dhe mund të kalojë kontrollin në dy mënyra :

- kur ekzekutohet instruksioni return, duke kthyer një vlerë rezultat
- kur arrin në kllapën mbyllëse te metodës, dhe nuk kthehet asnjë vlerë rezultat

Jepen shembulli më poshtë kur metoda nuk ka vlerë kthyese:

```
afisho("Kjo eshte java ne Unishk");
```

dhe rasti kur metoda ka vlerë kthyese:

```
int rezultati = shuma(6, 9);
```

### Shembull

Programi më poshtë llogarit minimumin mes dy numrash dhe afishon këtë rezultat. Llogaritja kryhet nëpërmjet një metode.

```
public class Minimum {
        public static void main(String[] args) {
           int a = 11;
           int b = 6;
           int c = llogaritMin(a, b);
           System.out.println("Vlera me e vogel = " + c);
        }
        /** kthen minimumin mes dy numrave te dhene */
        public static int llogaritMin(int n1, int n2) {
           int min;
           if (n1 > n2)
              min = n2;
           else
              min = n1;
           return min;
        }
}
```

Në console:

```
Vlera me e vogel = 6
```

### Fjala çelës void

Fjala çelës void përdoret kur metoda nuk kthen asnjë vlerë si rezultat, por thjesht bën afishime ose veprime tjera pa arritë në vlerë kthyese.

### Shembull

Në këtë shembull metoda përcakton rendin e një pike të dhënë, por nuk kthen vlerë.

```
public class ShembullVoid {
```

```
        public static void main(String[] args) {
           renditje(255.7);
        }
        public static void renditje(double pika) {
           if (pika >= 202.5) {
              System.out.println("Rendi:A1");
           }else if (pika >= 122.4) {
              System.out.println("Rendi:A2");
           }else {
              System.out.println("Rendi:A3");
           }
        }
}
```

Në console :

```
Rendi:A1
```

### Parametrat e një metode

Në rastin kur metoda ka parametra, ato do jenë të përcaktuar në deklarimin të metodës. Vini re se rendi i parametrave tek deklarimi i metodës do jetë i njëjtë me rendin e argumentave në thirrje.

### Shembull

Në këtë shembull janë ndërruar vlerat e variablave a dhe b, por ky ndërrim ka ndodhë vetëm brenda metodës, jashtë saj këto vlera janë përsëri në variablat lokal a dhe b, të pa modifikuar.

```
public class Nderrim {
public static void main(String[] args) {
   int a = 30;
   int b = 45;
```

```
   System.out.println("Para nderrimit, a = " + a + " and b = " + b);
            // thirrje e metodes
   swapFunction(a, b);
   System.out.println("\n**Vlerat jane nderruar**:");
   System.out.println("Pas nderrimit, a = " + a + " dhe b = " + b);
}
public static void swapFunction(int a, int b) {
   System.out.println("Para nderrimit (ne metode), a = " + a + " b =
" + b);
   // nderrim mes n1 dhe n2
   int c = a;
   a = b;
   b = c;
   System.out.println("Pas nderrimit (ne metode), a = " + a + " b =
" + b);
   }
}
```

Në console:

```
Para nderrimit, a = 30 and b = 45
Para nderrimit (ne metode), a = 30 b = 45
Pas nderrimit (ne metode), a = 45 b = 30
**Vlerat jane nderruar**:
Pas nderrimit, a = 30 dhe b = 45
```

### Mbivendosja e metodave

Një klasë mund të ketë dy ose më shumë metoda me të njëjtin emër por me parametra të ndryshëm. Ky efekt quhet ndryshe mbivendosje metode (overloading). Kur dy metoda kanë të njëjtin emër metode, tip apo numër parametrash, por ndodhen në klasa të ndryshme, atëherë efekti është mbishkrim metode (overriding).

### Shembull

Në këtë shembull llogaritet minimumi i dy numrave të plote, dhe minimumi i dy numrave me presje. Do krijohen dy metoda të ndryshme, por me të njëjtin emër, meqë veprimi është i njëjtë.

```
public class Overloading {
        public static void main(String[] args) {
           int a = 11;
           int b = 6;
           double c = 7.3;
           double d = 9.4;
           int result1 = minimum(a, b);
           // i njejti funksion por me parametra te ndryshem
           double result2 = minimum(c, d);
           System.out.println("Vlera minimale = " + result1);
           System.out.println("Vlera minimale = " + result2);
        }
        // per numrat e plote
        public static int minimum(int n1, int n2) {
           int min;
           if (n1 > n2)
              min = n2;
           else
              min = n1;
           return min;
        }
        // per numrat me presje
        public static double minimum(double n1, double n2) {
          double min;
           if (n1 > n2)
```

```
            min = n2;
         else
            min = n1;
         return min;
      }
}
```

Në console:

```
Vlera minimale = 6
Vlera minimale = 7.3
```

### Përdorimi i argumentave command-line

Gjatë ekzekutimit të programit në Java në command-line, mund të kalojmë argumente për metodën main. Këtu marrin kuptim edhe String args[] e vendosur për çdo deklarim të main.

### Shembull

Në këtë program, do afishohen argumentet e kaluar nga command-line.

```
public class CommandLine {
      public static void main(String args[]) {
         for(int i = 0; i<args.length; i++) {
            System.out.println("args[" + i + "]: " +  args[i]);
         }
      }
}
```

Thirrja ne command-line do jete :

```
$java CommandLine this is a command line 200 -100
```

Në console :

```
args[0]: this
```

```
args[1]: is
args[2]: a
args[3]: command
args[4]: line
args[5]: 200
args[6]: -100
```

### Konstruktori

Metoda e posatshme për krijimin e një objekti të një klase quhet ndryshe konstruktor i klasës. Ai ka të njëjtin emër me emrin e klasës dhe nga sintaksa ngjan me metodë, por nuk ka tip daljeje, dhe as përcaktim void. Roli i tij është të japë vlerat fillestare variablave të instancës të klasës, ose të kryejë ndonjë procedurë tjetër inicializuese. Çdo klasë ka një konstruktor të gatshëm, i cili nuk ka parametra. Në rastin kur programuesi shton një konstruktor të një klasë, atëherë konstruktori i gatshëm nuk mund të përdoret.

### Shembull

Rasti më poshtë tregon implementimin e një konstruktori pa parametra. Zakonisht konstruktori ka parametra, mundësisht aq sa ka klasa variabla të instancës.

```
public class Ore {
	String modeli;
	String ngjyra;

	public Ore() {
		modeli="";
		ngjyra="";
	}

	public static void main(String []args) {
		/* Krijimi i objektit */
		Ore o1 = new Ore();
	}

}
```

### Fjala çelës this

Fjala this përdoret për të referencuar objektin me të cilin është thirrë metoda ose konstruktori. Kjo fjalë përdoret vetëm brenda metodave jo statike. Parimisht, ajo shërben për të diferencuar variablat e instancës prej variablave lokalë, nëse kanë të njëjtin emër brenda një metode.

```
class Student {
        int mosha;
        Student(int mosha) {
           this.mosha = mosha;
        }
}
```

### Shembull

```
public class FjalaThis {
        // variabli i instances num
        int num = 10;

        FjalaThis() {
           System.out.println("Ky eshte nje shembull perdorimi i
fjales this");
        }
        FjalaThis(int num) {
           // thirrja e konstruktorit te gatshem
           this();
           // ne te majte eshte variabli i instances, djathtas
eshte variabli lokal
           this.num = num;
        }
        public void afishim() {
            // variabli lokal num
           int num = 20;
           System.out.println("Ky objekt ka statusin "+ this.num);
```

```
        }
        public static void main(String[] args) {
           // inicializimi i nje objekti te klases
           FjalaThis obj1 = new FjalaThis();
           // thirrja e metodes afishim
           obj1.afishim();
           // Kalimi i nje vlere nepermjet konstruktorit
           FjalaThis obj2 = new FjalaThis(30);
           // Thirrja e metodes afishim
           obj2.afishim();
        }
}
```

Në console :

```
Ky eshte nje shembull perdorimi i fjales this
Ky objekt ka statusin 10
Ky eshte nje shembull perdorimi i fjales this
Ky objekt ka statusin 30
```

### Metoda finalize ( )

Metoda finalize bën finalizimin e objekteve pasi janë përdore. Ajo siguron fshirjen finale të tyre nga garbage collector. Në rastet kur hapet një skedar, duhet siguruar se do kryhet me sukses mbyllja e tij. Për të shtuar finalizer ne një klasë, duhet përcaktuar metoda, dhe gjatë ekzekutimit do e thërrasë automatikisht kur do fshihen objektet e klasës. Brenda kësaj metoda do shënohen veprime që kryhen para fshirjes së objekteve. Sintaksa e kësaj metode është :

```
protected void finalize( ) {
   // Instruksionet e metodes shenohen ketu
}
```

Fjala çelës protected shënohet për të penguar akses të plotë për këtë metodë.

# Leksioni 7

# Tabelat në Java

Java ofron strukturën e të dhënave tabelë (matricë) e cila ruan elemente të të njëjtit tip, në mënyrë sekuenciale në hapësira me madhësi fikse. Një tabelë përdoret për të ruajtur një numër të caktuar variablash të të njëjtit tip. Në vend që të deklarohen variabla individualë si numer1, numer2...numer99, deklarohet një variabël tabelë me numra, të cilët përfaqësojnë variablat numer[0], numer[1]..numer[99]. Në këtë leksion do trajtohen raste deklarimi të tabelave, krijimit të tyre, si dhe procesimi i vlerave që përmbajnë.

## Deklarimi i një tabele

Për të përdorë një tabelë në program, ajo duhet deklaruar më parë duke specifikuar tipin e të dhënave që do ruajnë tabela dhe madhe si dhe identifikuesin e saj.

### Sintaksa

```
tip[] tabele;   // versioni me i perdorur
```

ose

```
tip tabele[];  // funksionon, por jo shume i përdorur
```

## Krijimi i një tabele

Një objekt tabele krijohet nëpërmjet operatorit new, me sintaksën e mëposhtme:

### Sintaksa

```
tabele = new tip[nrElementesh];
```

Ky instruksion bën dy veprime njëherësh:

-krijon një tabelë duke përdorë new tip[nrElementesh]

-i cakton referencën tabele tabelës së re

Deklarimi i një tabele, krijimi i tabelës dhe caktimi i referencës mund të përmblidhen në një instruksion të vetëm :

```
tip[] tabele = new tip[nrElementesh];
```

Në mënyrë alternative, tabela mund të deklarohet dhe incializohet :

```
tip[] tabele = {vlera0, vlera1, ..., vlerak};
```

Elementët e tabelës aksesohen nëpërmjet indeksit të tabelës. Ky indeks nis më vlerën zero për elementin e parë në tabelë.

### Shembull

Deklarimi dhe krijimi i një tabele mund të përmblidhet në këtë instruksion :

```
double[] tab = new double[10];
```

### Përpunimi i të dhënave të tabelës

Gjatë përpunimit të elementëve të tabelës, shpesh përdoren cikli for ose cikli foreach, sepse njihet paraprakisht numri i të dhënave të tabelës.

### Shembull

Shembulli më poshtë llogarit shumën dhe maksimumin e elementeve të tabelës

```
public class TestArray {
        public static void main(String[] args) {
           double[] tab = {1.9, 2.9, 3.4, 3.5};
           // afishimi i elementeve te tabeles
           for (int i = 0; i < tab.length; i++) {
              System.out.println(tab[i] + " ");
           }
          // mbledhja e elementeve te tabeles
          double total = 0;

          for (int i = 0; i < tab.length; i++) {
              total += tab[i];
          }
          System.out.println("Shuma e elementeve eshte  " + total);
          // gjetja e elementit maksimal
          double max = tab[0];
```

```
            for (int i = 1; i < tab.length; i++) {
                if (tab[i] > max) max = tab[i];
            }
            System.out.println("Maksimumi eshte " + max);
        }
}
```

Në console:

```
1.9
2.9
3.4
3.5
Shuma e elementeve eshte  11.7
Maksimumi eshte 3.5
```

### Shembull

Në shembullin e dhënë, elementet e tabelës afishohen duke përdorë një cikël for.

```
public class TestArray {
      public static void main(String[] args) {
            double[] tab = {1.9, 2.9, 3.4, 3.5};
            // afishimi i elementeve
            for (double element: tab) {
               System.out.println(element);
            }
      }
}
```

Në console :

```
1.9
2.9
3.4
```

```
3.5
```

### Kalimi i një tabele si parametër metode

Ashtu siç mund të kalohen tipa primitive, ashtu kalohet edhe një tabelë në metodë. Shembulli i mëposhtëm afishon elementët e një tabele me numra të plotë.

**Shembull**

```
public class TestArray {
      public static void main(String[] args) {
            int[] tab = {9, 2, 4, 3};
            afishoTabelen(tab);
         }
      public static void afishoTabelen(int[] t) {
               for (int i = 0; i < t.length; i++) {
                  System.out.print(t[i] + " ");
               }
            }
}
```

Në console :

```
9 2 4 3
```

### Tabela si një vlerë kthyese e metodës

Metoda gjithashtu mund të kthejë një tabelë. Në shembullin e mëposhtëm, metoda kthen tabelën e kundërt me tabelën që hyn si parametër.

**Shembull**

```
public class TestArray {
      public static void main(String[] args) {
            int[] tab = {9, 2, 4, 3};
            afishoTabelen(tab);
            int []tabKundert=reverse(tab);
```

```
            System.out.println("\nTabela e permbysur : ");
            afishoTabelen(tabKundert);
        }
    public static void afishoTabelen(int[] t) {
            for (int i = 0; i < t.length; i++) {
                System.out.print(t[i] + " ");
            }
        }
    public static int[] reverse(int[] t) {
            int[] result = new int[t.length];
            for (int i = 0, j = result.length - 1; i < t.length;
i++, j--) {
                result[j] = t[i];
            }
            return result;
        }
}
```

Në console :

```
9 2 4 3
Tabela e permbysur :
3 4 2 9
```

### Klasa Arrays

Në paketën java.util ndodhet klasa e gatshme Arrays, e cila përmban disa metoda statike për renditjen dhe kërkim elementi në tabelë. Këto metoda janë overloaded për çdo tip primitiv të numrave. Përshkrimi i tyre bëhet në tabelën më poshtë.

| Metoda të klasës Arrays |
| --- |
| `public static int binarySearch(Object[] a, Object key)` |
| Kjo metodë kërkon në tabelën e dhënë si parametër, e cila jepet me elemente Object, por që mund të jenë Byte, Int, Double, etj, për elementin e dhënë key. Algoritmi i përdorur është |

| |
|---|
| kërkimi binar. Tabela duhet të jetë e renditur paraprakisht. Vlera kthyese është indeksi ku ndodhet elementi key, dhe nqs nuk gjendet në tabelë kthen vlerën -1. |
| `public static boolean equals(long[] a, long[] a2)`<br>Kjo metodë kthen vlerën true nqs dy tabelat a dhe a2 janë të barabarta, dmth përmbajnë të njëjtin numër elementesh, si dhe elementet korespondues janë dy e nga dy njëlloj. Tabelat mund të përmbajnë vlera primitive si byte, short, int, etj. |
| `public static void fill(int[] a, int val)`<br>Kjo metodë i cakton vlerën e dhënë val çdo elementi të tabelës. Tabela mund të përmbajë vlera primitive float, short, apo elemente të klasave Byte, Double, etj. |
| `public static void sort(Object[] a)`<br>Kjo metodë rendit tabelën me objekte sipas rendit rritës, duke u nisur nga vlera e elementeve të tabelës. Kjo metodë është overriden për Byte, short, Int, etj. |

## Leksioni 8

## Sintaksa e shprehjeve të rregullta

Java ofron paketën java.util.regex për të kontrolluar shprehjet e rregullta me një pattern (model, shablon). Një shprehje e rregullt është një sekuencë e veçantë karakteresh që ndihmon në gjetjen e një stringe ose një një grup stringash, duke përdorë një sintaksë pattern. Këto shprehje mund të përdoren për të kërkuar, edituar dhe përpunuar tekst dhe të dhëna.

Paketa `java.util.regex` konsiston kryesisht në këto tre klasa:

- Klasa Pattern - Një objekt Pattern është një paraqitje e një shprehjeje të rregullt. Kjo klasë nuk ofron konstruktor publik. Për të krijuar një pattern, duhet të thirret metodat compile( ) të cilat janë statike dhe publike, dhe kthejnë si rezultat një objekt Pattern. Këto metoda marrin një shprehje të rregullt si argument.
- Klasa Matcher - Një objekt Matcher interpreton një pattern dhe kryen një veprim kundrejt stringës input. Klasa Matcher nuk ka konstruktor publik. Objekti Matcher merret nga thirrja e metodës matcher( ) mbi një objekt Pattern.

- Klasa PatternSyntaxException – Një objekt i tillë është një përjashtim i pakontrolluar që tregon një gabim sintakse në një pattern shprehje të rregullt.

### Grupimet e karaktereve

Karakteret në një shprehje mund të trajtohen si një njësi e vetme. Këto grupe janë krijuar duke vendosur karakteret e grupuara brenda kllapave. Për shembull, shprehja e rregullt (mace) krijon një grup që përmban shkronjat m, a, c, dhe e.

Këto grupime numërohen duke numëruar kllapat hapëse, nga e majta në të djathë. Në shprehjen ((A)(B(C))) janë këto grupe :

- ((A)(B(C)))
- (A)
- (B(C))
- (C)

Për të gjetë sa grupe janë në një shprehje, thirret metoda groupCount me një objekt Matcher. Kjo metodë kthen si vlerë një numër int, i cili është numri i grupeve prezent në shprehje. Grupi zero është një grup special i cili përfaqëson krejt shprehjen. Ky grup nuk përfshihet në numrin e grupeve.

### Shembull

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class RegexMatches {
   public static void main( String args[] ) {
      // String qe kontrollohet per pattern
      String line = "Ky artikull eshte shenuar si QT3000! OK?";
      String pattern = "(.*)(\\d+)(.*)";
      // krijimi i objektit pattern
      Pattern r = Pattern.compile(pattern);
      // Krijimi i objektit matcher
      Matcher m = r.matcher(line);
```

```
      if (m.find( )) {
         System.out.println("U gjet kombinimi: " + m.group(0) );
         System.out.println("U gjet kombinimi: " + m.group(1) );
         System.out.println("U gjet kombinimi: " + m.group(2) );
      }else {
         System.out.println("Nuk ka kombinime");
      }
   }
}
```

Në console:

```
U gjet kombinimi: Ky artikull eshte shenuar si QT3000! OK?
U gjet kombinimi: Ky artikull eshte shenuar si QT300
U gjet kombinimi: 0
```

### Sintaksa e shprehjeve të rregullta

Në tabelën e mëposhtme është një listë

| Shprehjet | Matches |
|---|---|
| ^ | Match për fillimin e rreshtit |
| $ | Match për fundin e rreshtit |
| . | Match për çdo karakter përvec rreshtit të ri. Përdorimi i opsionit m, lejon të jetë match edhe për rreshtin e ri. |
| [...] | Match çdo karakter në kllapa |
| [^...] | Match per çdo karakter që nuk është në kllapa. |
| \A | Fillimi i stringes. |
| \z | Fundi i stringes. |
| \Z | Fundi i stringes përvec rreshtit të fundit të lejueshëm. |

| | |
|---|---|
| **re*** | Match 0 ose asnjë raste të shprehjes paraardhëse. |
| **re+** | Match 1 ose më tepër rastë të shprehjes paraardhëse |
| **re?** | Match 0 ose 1 raste të shprehjes paraardhëse. |
| **re{ n}** | Match saktësisht n raste të shprehjes paraardhëse. |
| **re{ n,}** | Match n ose më shumë raste të shprehjes paraardhëse. |
| **re{ n, m}** | Match të paktën n dhe të shumtën m raste të shprehjes paraardhëse. |
| **a\| b** | Match ose a ose b. |
| **(re)** | Grupon shprehjet e rregullta dhe kujton tekstin e rastisur. |
| **(?: re)** | Grupon shprehjet e rregullta pa kujtuar tekstet e rastisur. |
| **(?> re)** | Match pattern të pavarura pa shënim. |
| **\w** | Match karakteret e fjalës |
| **\W** | Match karakteret jo të fjalës. |
| **\s** | Match hapsirat, tab, rresht i ri, etj. |
| **\S** | Match karakteret jo hapsirë. |
| **\d** | Match për shifrat 0-9 |
| **\D** | Match karakteret jo shifrore. |
| **\A** | Match fillimin e string. |
| **\Z** | Match fundin e string. Nqs ekziston një rresht i ri, match përkon para këtij rreshti. |
| **\z** | Match fundin e string. |
| **\G** | Match piken ku mbaroi match paraardhës. |
| **\n** | Referencë që nxë grupimin e n'të. |
| **\b** | Match kufinjtë e fjalëve kur është jashtë kllapave. Match backspace kur është brenda kllapave. |
| **\B** | Match kufinjtë e jo-fjalëve. |
| **\n, \t** | Match rreshtin e ri, carriage return dhe tab. |
| **\Q** | Shmang çdo karakter deri tek \E |
| **\E** | Shmang cilësimet që fillojnë me \Q |

# Leksioni 9

# Klasat e brendshme dhe klasat anonime

## Klasa e ndërmjetme (nested class)

Në Java është e mundur që edhe metodat edhe variablat e klasave mund të jenë pjesë edhe të klasave tjera. Java lejon shkrimin e një klase brenda një klase tjetër. Klasat që shkruhen brenda klasës tjetër quhet klasë e ndërmjetme (nested class) dhe klasa që e përfshin atë quhet klasë e jashtme.

## Sintaksa

Sintaksa e klasave të ndërmjetme jepet më poshtë:

```
class Outer_Demo {
        class Nested_Demo {
        }
}
```

Klasat e ndërmjetme ndahen në dy tipa:

- Klasat e ndërmjetme jo statike- këto klasa kanë variabla të instancës dhe metoda jo-statike.
- Klasat e ndërmjetme statike- këto klasa kanë elementë statik.

## Klasa e brendshme (inner class)

Klasat e brendshme janë një mekanizëm sigurie në Java. Nuk mund të kemi një klasë me akses private, por mund të kemi një klasë si element i një klase tjetër. Me këtë rast, klasa e brendshme mund të bëhet private. Llojet e klasave të brendshme janë si më poshtë :

- Klasë e brendshme (basic)
- Klasë e brendshme në metodë lokale
- Klasë e brendshme anonime

### Klasa e brendshme standard

Eshte e thjeshtë të krijohet një klasë e brendshme. Mjafton të shkruhet kodi i një klase brenda një klase tjetër. Një klasë e brendshme mund të deklarohet si klasë private, që do të pengojë aksesimin e objekteve të saj jashtë klasës.

### Shembull

Në këtë shembull është krijuar një klasë e brendshme private, e cila do aksesohet nëpërmjet klasës së jashtme.

```
class Outer_Demo {
        int num;
        //klasa e brendshme
        private class Inner_Demo {
           public void afisho() {
              System.out.println("Afishime nga klasa e brendshme");
           }
        }
        // Aksesimi i klases se brendshme
        void afisho_I() {
           Inner_Demo inner = new Inner_Demo();
           inner.afisho();
        }
     }


     public class My_class {
        public static void main(String args[]) {
           // objekti i klases se jashtme
           Outer_Demo outer = new Outer_Demo();
```

```
            // aksesimi i klases se brendshme nga klasa e jashtme
            outer.afisho_I();
        }
}
```

Në console :

```
Afishime nga klasa e brendshme
```

### Aksesimi i elementëve privatë

Siç u përmend më lart, klasat e brendshme përdoren edhe për të kyçur elementët privatë të një klase. Supozoni se një klasë ka një element privat.

### Shembull

Në këtë shembull bëhet e mundur shkrimi i një klase të brendshme, me një metodë private, psh getNr, dhe një metodë që kthen këtë element privat, dhe shkrimi i një klase tjetër, që do lexojë këtë element privat të klasës së brendshme. Për të inicializuar klasën e brendshme, duhet të inicializohet klasa e jashtme, duke krijuar një objekt të klasës së jashtme.

```
class Outer_Demo {
        // Elementi privat ne klasen e jashtme
        private int num = 175;
        // klasa e brendshme

        public class Inner_Demo {
            public int getNr() {
                System.out.println("Po kthehet nga klasa e brendshme
nr : "+num);
                return num;
            }
        }
    }
    public class My_class2 {
```

```
        public static void main(String args[]) {
           // Inicializimi i klases se jashtme
           Outer_Demo outer = new Outer_Demo();
           // Inicializimi i klases se brendshme
           Outer_Demo.Inner_Demo inner = outer.new Inner_Demo();
           System.out.println("Ne klasen demo : "+inner.getNr());
        }
}
```

Në console :

```
Po kthehet nga klasa e brendshme nr : 175
Ne klasen demo : 175
```

### Klasa e brendshme në metodë lokale

Në Java është e mundur të shkruhet një klasë brenda një metode dhe kjo të cilësohet si një tip lokal. Si variablat lokal, zona e klasës së brendshme është vetëm zona brenda metodës. Një klasë e tille mund të inicializohet vetëm nga brenda metodës ku është përcaktuar klasa e brendshme.

### Shembull

Në këtë shembull është krijuar dhe inicializuar një klasë brenda një metode lokale my_Method.

```
public class Outerclass {
        // metoda e klases se jashtme
        void my_Method() {
           int num = 23;
           // klasa e brendshme ne metode lokale
           class MethodInner_Demo {
              public void print() {
```

```
                System.out.println("Kjo eshte klasa e brendshme
"+num);
            }
         } // fundi i klases se brendshme
         // aksesimi i klases se brendshme
         MethodInner_Demo inner = new MethodInner_Demo();
         inner.print();
      }



      public static void main(String args[]) {
         Outerclass outer = new Outerclass();
         outer.my_Method();
      }
}
```

Në console:

```
Kjo eshte klasa e brendshme 23
```

### Klasat e brendshme anonime

Një klasë e brendshme mund të deklarohet pa përcaktuar emrin. Kjo teknikë njihet ndryshe si deklarim i klasës së brendshme anonime. Në rastin e klasës së brendshme anonime, deklarimi dhe inicializimi kryhet në të njëjtën kohë. Zakonisht kjo teknikë përdoret kur duhen mbishkruar metodat e një klase ose një ndërfaqeje.

### Shembull

```
abstract class AnonymousInner {
      public abstract void mymethod();
   }
   public class Outter_class {
```

```
        public static void main(String args[]) {
           AnonymousInner inner = new AnonymousInner() {
              public void mymethod() {
                 System.out.println("Shembull per klasen anonime");
              }
           };
           inner.mymethod();
        }
}
```

Në console :

```
Shembull per klasen anonime
```

### Klasa anonime si argument i një metode

Në përgjithësi, në qoftë se metoda pranon si argument një objekt, një ndërfaqe ose një klasë abstrakte, atëherë mund të implementohet një ndërfaqe, zgjerohet një klasë abstrakte ose i kalohet një objekt metodës. Nqs është një klasë, atëherë mund të kalojmë tek metoda. Në të tre rastet, mund të kalohet një klasë anonime e brendshme si argument tek një metodë.

### Shembull

```
//nderfaqja
interface Mesazh {
      String sms();
}
public class My_class {
//metoda qe pranon si argument nje klase anonime
      public void afishoMesazh(Mesazh m) {
       System.out.println(m.sms() +
       "---- Shembull i klases se brendshme si argument");
      }
      public static void main(String args[]) {
```

```
      // Inicializimi i klases
      My_class obj = new My_class();
      // kalimi i klases anonime si argument
      obj.afishoMesazh(new Mesazh() {
      public String sms() {
          return "Ckemi";
       }
   });
}
}
```

Në console :

```
Ckemi---- Shembull i klases se brendshme si argument
```

### Klasa e ndërmjetme statike

Një klasë e ndërmjetme statike është klasa e cila paraqitet si një element statik i klasës së jashtme. Ajo mund të aksesohet pa inicializuar klasën e jashtme, ose duke përdorë elementë të tjerë statik. Si elementët statik të klasave, edhe kjo klase e brendshme statike nuk ka akses në variablat e instancës të klasës së jashtme.

### Shembull

```
public class Outerclass {
        static class Nested_Demo {
           public void my_method() {
              System.out.println("Afishim nga klasa e brendshme
statike");
           }
        }
        public static void main(String args[]) {
```

```
            Outerclass.Nested_Demo nested = new
Outerclass.Nested_Demo();
            nested.my_method();
        }
}
```

Në console :

```
Afishim nga klasa e brendshme statike
```

## Leksioni 10

## Trashëgimia në Java

Trashëgimia mund të përkufizohet si një proces ku një klasë merr cilësitë (metoda dhe fusha) e një klase tjetër. Kjo teknikë mundëson që informacioni të organizohet në strukturë hierarkike.

Klasa që i trashëgon cilësitë nga një klasë tjetër quhet klasë e derivuar, kurse klasa nga e cila merren cilësitë quhet klasa bazë.

### Fjala çelës extends

Fjala extends përdoret në deklarimin e klasës së derivuar.

### Shembull

Në këtë shembull përfshihet kodi i dy klasave, Calculator dhe MyCalculator. Duke përdorë fjalën çelës extends, MyCalculator trashëgon metodat mbledh dhe zbrit të klasës Calculator.

```
class Calculator {
   int z;
   public void mbledh(int x, int y) {
      z = x + y;
      System.out.println("Shuma e "+x+" dhe "+y+" eshte "+z);
   }
   public void zbrit(int x, int y) {
```

```
        z = x - y;
        System.out.println("Diferenca mes "+x+" dhe "+y+" eshte "+z);
    }
}
public class MyCalculator extends Calculator {
    public void shumezim(int x, int y)
    {
        z = x * y;
        System.out.println("Prodhimi i "+x+" dhe "+y+" eshte "+z);
    }
    public static void main(String args[]) {
            int a = 20, b = 10;
            MyCalculator demo = new MyCalculator();
            demo.mbledh(a, b);
            demo.zbrit(a, b);
            demo.shumezim(a, b);
        }
}
```

Në console:

```
Shuma e 20 dhe 10 eshte 30
Diferenca mes 20 dhe 10 eshte 10
Prodhimi i 20 dhe 10 eshte 200
```

Në programin e dhënë, kur krijohet nje objekt MyCalculator është përfshirë një kopje e përmbajtjes së klasës bazë. Kur përdoret një objekt i klasës së derivuar, mund të aksesojnë pjesën e klasës bazë.

Vini re që një klasë e derivuar trashëgon elementet e klasës bazë si variablat e instancës, metodat dhe klasat e ndërmjetme. Konstruktorët nuk janë elemente të klasës, prandaj nuk trashëgohen tek klasa e derivuar.

### Fjala çelës super

Fjala çelës super është e ngjashme me fjalën çelës this. Më poshtë jepen disa skenarë ku përdoret fjala super:

- Në diferencimin e elementëve të klasës bazë nga elementët e klasës së derivuar, nëse këto elementë kanë të njëjtin emër.
- Në thirrjen e konstruktorit të klasës bazë nga klasa e derivuar.

### Diferencimi i elementëve të klasës bazë nga klasa e derivuar

Nëse një klasë trashëgon cilësitë e një klase tjetër, dhe nëse një element i klasës bazë ka emër të njëjtë me një element të klasës së derivuar, atëherë fjala super bën dallimin duke thirrë elementin e klasës bazë.

### Shembull

```
class KlaseBaze {
     int numer = 20;

     // metoda afisho e klases baze
     public void afisho() {
           System.out.println("Ky afishim behet nga klasa baze");
     }
}
public class KlaseDerivuar extends KlaseBaze {
     int numer = 10;
        // metoda afisho e klases se derivuar
     public void afisho() {
       System.out.println("Ky afishim behet nga klasa e derivuar");
     }
     public void metode_shtese() {
           // Inicializimi i klasave
        KlaseDerivuar kldr = new KlaseDerivuar();
           // thirrja e metodes afisho te klases derivuar
```

```
        kldr.afisho();
           // thirrja e metodes afisho e klases baze
        super.afisho();
           // afishimi i variablit te instances numer
        System.out.println("Vlera e variablit te klases derivuar:"+
kldr.numer);
        // afishimi i variablit te instances numer te klases baze
        System.out.println("Vlera e variablit te klases baze:"+
super.numer);
        }
     public static void main(String args[]) {
        KlaseDerivuar obj = new KlaseDerivuar();
        obj.metode_shtese();
     }
}
```

Në console :

```
Ky afishim behet nga klasa e derivuar
Ky afishim behet nga klasa baze
Vlera e variablit te klases derivuar:10
Vlera e variablit te klases baze:20
```

### Thirrja e konstruktorit të klasës bazë

Nqs një klasë trashëgon cilësitë e një klase tjetër, klasa e derivuar automatikisht përdor konstruktorin e gatshëm të klasës bazë. Por nqs klasa bazë ka konstruktor me parametra, atëherë duhet përdorë fjala çelës this për të thirrë konstruktorin e klasës bazë nga klasa e derivuar.

### Relacioni is-a

Is-a është një mënyrë të thënuri që ky objekt është tip me atë objekt. Le të shohim një shembull se si fjala çelës extends realizon trashëgiminë e klasave.

### Relacioni has-a

Ky relacion përcakton nëse një klasë përmban si cilësi një objekt të një klase tjetër. Ky relacion ndihmon në zvogëlimin e dublikimeve në kod.

### Shembull

```
public class Timon {
}
public class Motorr {
}
public class Automjet {
      private Timon tm;
      private Motorr m;
}
```

Relacioni has-a në këtë rast përcakton se :

- Automjet ka një Timon (has-a)
- Automjet ka një Motorr (has-a)

Klasa Automjet ka të përcaktuar një timon, si pjesë përbërëse, e cila duke qenë klasë më vete, atëherë një objekt i saj do jetë pjesë e automjetit. E njëjta gjë ndodh dhe me motorin e klasës Automjet. Në programimin e orientuar nga objektet, përdoruesi nuk është në dijeni se cili objekt kryen funksionalitetin. Për këtë, klasa automjet fsheh implementimin e detajeve. Përdoruesi kërkon të kryejë veprime me klasën automjet, dhe kjo klasë do kryejë këtë veprim vetëm ose me ndihmën e ndonjë klase tjetër.

## Leksioni 11

## Polimorfizmi

Polimorfizmi është teknika që mundëson një objekt të marrë disa forma. Rasti më i zakonshëm është kur një referencë i klasës bazë përdoret për të referuar një objekt i klasës së derivuar. Në Java çdo objekt mund të quhet edhe polimorfik, meqë çdo objekt është një

objekt i derivuar i klasës Object. E rëndësishme është të dihet që mënyra e vetme për të aksesuar një objekt është nëpërmjet referencës për të. Kjo referencë i përket vetëm një tipi, dhe pasi deklarohet ky tip nuk mund të ndërrohet më. Kjo referencë mund ti caktohet objekteve tjera. Tipi i referencës përcakton metodat që ajo mund të thërrasë me objektin. Një referencë mund të deklarohet e tipit të një klase ose të një ndërfaqeje.

**Shembull**

Më lart kemi diskutuar rreth efektit overriding te metodave, kur një objekt i një klase të derivuar mbivendos një metodë të klasës bazë. Versioni i metodës së klasës bazë është e fshehur për klasën e derivuar dhe nuk mund të thirret përveç se me përdorimin e fjalës çelës super, brenda versionit të klasës së derivuar.

```
public class Nenpunes {
        private String emri;
        private String adresa;
        public Nenpunes(String e, String a) {
           System.out.println("Konstruktori i klases nenpunes");
           this.emri = e;
           this.adresa = a;
        }
        public void kontrolloMail() {
           System.out.println("Kontrollim i mailit " + this.emri +
" " + this.adresa);
        }
        public String toString() {
           return emri + " " + adresa ;
        }
        public String getEmri() {
           return emri;
        }
        public String getAdresa() {
           return adresa;
```

```
        }
        public void setAdresa(String a) {
           adresa = a;
        }
     }
public class Administrator extends Nenpunes {
   private int oreNeJave;
   public Administrator(String e, String a, int o) {
      super(e, a);
      setOreShtese(o);
   }
   public void kontrolloMail() {
      System.out.println("Kontroll email ne klases administrator ");
      System.out.println("Kontrollim email " + getEmri());
   }
   public double getOreShtese() {
      return oreNeJave;
   }
   public void setOreShtese(int ore) {
      if(ore >= 0.0) {
         oreNeJave = ore;
      }
   }
   public double llogaritPagen() {
      System.out.println("Paga per " + getEmri());
      return oreNeJave*15000;
   }
}
public class VirtualDemo {
```

```
        public static void main(String [] args) {
           Administrator ad = new Administrator("Mohd Mohtashim",
"Ambehta, UP", 50);
           Nenpunes n = new Administrator("John Adams", "Boston,
MA",40);
           System.out.println("Email check nga admin --");
           ad.kontrolloMail();
           System.out.println("\n Email check nga nenpunes");
           n.kontrolloMail();
        }
     }
```

Në console:

```
Konstruktori i klases nenpunes
Konstruktori i klases nenpunes
Email check nga admin --
Kontroll email ne klases administrator
Kontrollim email Mohd Mohtashim

 Email check nga nenpunes
Kontroll email ne klases administrator
Kontrollim email John Adams
```

Në këtë shembull janë inicializuar dy objekte administrator, njëri duke përdore një referencë administrator dhe tjetri duke përdorë një referencë nenpunes. Kur thirret metoda kontrolloMail( ) kompilatori shikon metodën kontrolloMail tek klasa Administrator dhe JVM thërret metodën kontrolloMail të kësaj klase.

Kontrollo mail e objektit n është krejt ndryshe sepse n është referencë e klasës Nenpunes. Kur kompilatori shikon kontrolloMail me këtë referencë ai thërret metodën e klasës Nenpunes. Gjatë kohës së kompilimit, kompilatori përdor kontrolloMail të klasës Nenpunës për të verifikuar këtë instruksion. Por gjatë kohës së ekzekutimit JVM thërret kontrolloMail të

klasës Administrator. Një sjellje e tillë quhet thirrje e metodave virtuale. Një metodë e mbivendosur thirret gjatë kohës së ekzekutimit cilado qoftë tipi i referencës së saj.

## Leksioni 12

## Abstraksioni në Java

Në gjuhët e programimit të orientuar nga objektet, abstraksioni është procesi i fshehjes së implementimit për përdoruesin, i cili shikon vetëm funksionalitetin. E thënë ndryshe, përdoruesi ka informacion se çfarë bën objekti, dhe jo si e realizon. Abstraksioni në Java realizohet nëpërmjet klasave abstrakte dhe ndërfaqeve.

### Klasa abstrakte

Një klasë abstrakte është e tillë me deklarimin me fjalën çelës abstract. Këto klasa mund të kenë ose jo metoda abstrakte të cilat nuk kanë instruksione brenda tyre. Por nqs një klasë ka një metodë të tillë abstrakte, atëherë klasa duhet të jetë e deklaruar si klasë abstrakte. Një klasë abstraktë nuk lejon inicializimin e objekteve të saj. Për të përdorë një klasë abstrakte, ajo duhet të caktohet si klasë bazë e një klase tjetër, dhe në këtë mënyrë mundësohet implementimi i metodave abstrakte që ajo mbart.

### Shembull

Më poshtë jepet një shembull i klasës abstrakte.

```
public abstract class Nenpunes {
        private String emri;
        private String adresa;
        public Nenpunes(String e, String a) {
           System.out.println("Konstruktori i klases nenpunes");
           this.emri = e;
           this.adresa = a;
        }
        public void kontrolloMail() {
```

```
            System.out.println("Kontrollim i mailit " + this.emri +
" " + this.adresa);
        }
        public String toString() {
           return emri + " " + adresa ;
        }
        public String getEmri() {
           return emri;
        }
        public String getAdresa() {
           return adresa;
        }
        public void setAdresa(String a) {
           adresa = a;
        }
}
public class AbstractDemo {
        public static void main(String [] args) {
         /* Kodi meposhte con ne gabim */
         //  Nenpunes e = new Nenpunes("George W.", "Houston, TX");
         System.out.println("\n Thirrje metodes kontroll email--");
        //   e.kontrolloMail();
        }
}
```

### Trashëgimi i klasës abstrakte

Elementet e klasës abstrakte Nenpunes mund të trashëgohen si zakonisht, si çdo klasë tjetër.

### Shembull

```
public class Administrator extends Nenpunes {
   private int oreNeJave;
   public Administrator(String e, String a, int o) {
      super(e, a);
      setOreShtese(o);
   }
   public void kontrolloMail() {
      System.out.println("Kontroll email ne klases administrator ");
      System.out.println("Kontrollim email " + getEmri());
   }
   public double getOreShtese() {
      return oreNeJave;
   }
   public void setOreShtese(int ore) {
      if(ore >= 0.0) {
         oreNeJave = ore;
      }
   }
   public double llogaritPagen() {
      System.out.println("Paga per " + getEmri());
      return oreNeJave*15000;
   }
}
public class AbstractDemo {
     public static void main(String [] args) {
          Administrator ad = new Administrator("Mohd Mohtashim",
"Ambehta, UP", 50);
          Nenpunes n = new Administrator("John Adams", "Boston,
MA",40);
```

```
            System.out.println("Email check nga admin --");
            ad.kontrolloMail();
            System.out.println("\n Email check nga nenpunes");
            n.kontrolloMail();
        }
}
```

Në console:

```
Konstruktori i klases nenpunes
Konstruktori i klases nenpunes
Email check nga admin --
Kontroll email ne klases administrator
Kontrollim email Mohd Mohtashim

 Email check nga nenpunes
Kontroll email ne klases administrator
Kontrollim email John Adams
```

### Metoda abstrakte

Nëse një klasë bazë përmban një metodë e cila nuk specifikohet aty por vetem tek klasat e derivuara, atëherë kjo metodë deklarohet si abstrakte tek klasa bazë. Fjala çelës e përdorë për këtë teknikë është fjala çelës abstract, dhe deklaron metodën si metodë abstrakte. Një metodë e tillë ka vetëm deklarimin, dhe jo implementimin. Në vend që të vendosen kllapat gjarpërushe, vendoset menjëherë pikëpresje pas deklarimit.

### Shembull

```
public abstract class Nenpunes {
        private String emri;
        private String adresa;
        public abstract void kontrolloMail();
}
```

Deklarimi i një metode si metodë abstrakte ka këto dy rrjedhoja:

- Klasa që përmban këtë metodë duhet gjithashtu të deklarohet abstrakte
- Çdo klasë që trashëgon këtë klasë duhet ose të mbishkruajë këtë metodë abstrakte (override) ose të deklarohet vetë si abstrakte.

### Enkapsulimi

Enkapsulimi është një nga katër konceptet më të rëndësishme në gjuhët e programimit të orientuara nga objektet. Tre tjerat janë trashëgimia, polimorfizmi dhe abstraksioni, të tria të trajtuara më lart. Enkapsulimi në Java është mekanizmi i mbledhjes së të dhënave (variablave) dhe veprimeve (metodave) në një njësi të vetme. Në enkapsulim, variablat e një klase do fshihen nga klasat e tjera, dhe mund të aksesohen vetëm nëpërmjet metodave të klasës. Një teknikë e tillë quhet data hiding (fshehje e të dhënave)

Enkapsulimi në Java realizohet me:

- Deklarimin e variablave të klasës si variabla privatë.
- Implementimin e metodave set dhe get për çdo variabël të klasës.

### Shembull

Më poshtë jepet një shembull si realizohet enkapsulimi në Java

```
public class Person {
        private String emri;
        private String id;
        private int mosha;
        public String getEmri() {
           return emri;
        }
        public int getMosha() {
           return mosha;
        }
        public String getId() {
           return id;
```

```
        }
        public void setEmri(String e) {
           emri = e;
        }
        public void setMosha( int m) {
           mosha = m;
        }
        public void setId( String i) {
           id= id;
        }
}
```

Metodat publike set dhe get janë mënyra e vetme aksesi për variablat e instancës së klasës Person. Zakonisht këtyre metodave iu referohemi si setter dhe getter. Variablat e instancës së klasës Person, do aksesohen në këtë formë :

```
public class EnkapsulationDemo {
     public static void main(String args[]) {
           Person p1 = new Person();
           p1.setEmri("James");
           p1.setMosha(20);
           p1.setId("12343ms");
           System.out.print("Emri : " + p1.getEmri() + " Mosha : "
+p1.getMosha());
     }
}
```

Në console :

```
Emri : James Mosha : 20
```

Avantazhet e enkapsulimit janë si më poshtë:

- Variablat e instancës së një klase mund të bëhen vetëm të lexueshme ose vetëm të shkrueshme. Vetëm të lexueshme kur mundësohet vetëm getter të variablit, ndërsa vetëm të shkrueshme kur mundësohet setter i variablit.
- Klasa ka kontroll total se cilat të dhëna ruhen në variablat e instancës të saj.
- Përdoruesi i një klase nuk është në dijeni se si i ruan të dhënat klasa. Klasa mund të ndryshohet duke ndërruar një tip të dhëne, dhe asgjë tjetër në kodin e përdoruesit nuk ka nevojë për modifikim.

## Leksioni 13

## Rast Studimi : Sistemi i pagesës, implementim i polimorfizëm dhe klasë abstrakte

Në këtë rast studimi do të implementohet sistemi i llogaritjes së pages duke përdore metoda abstrakte dhe polimorfizëm, që bazohen në trashëgiminë e klasave të nënpunësve. Kërkesat për këtë software janë të tilla:

Një kompani paguan nënpunësit me një pagë javore. Nënpunësit janë në katër lloje të ndryshëm: nënpunës full time me page fikse javore të pavarur nga numri i orëve, nënpunes part time i cili paguhet në bazë të numrit të orevë të punës, nënpunës me komision i cili paguhet në bazë të përqindjeve të shitjeve si dhe nënpunës baze me komision i cili paguhet me një page bazë plus me një komision shtesë për shitjet. Aktualisht kompania ka vendosë një bonus shtesë për nënpunësit bazë me komision duke i shtuar 10% të rrogës bazë. Kompania kërkon një aplikacion që llogarit sistemin e pagesën në mënyrë polimorfike.

Në këtë rast studimi do përdoret një klasë abstrakte Nënpunës e cila përfaqëson konceptin e përgjithshëm të një nënpunësi. Klasa do derivohet në klasat NenpunesFullTime, NenpunesPartTime, NenpunesMeKomision si dhe NenpunesMeKomisionBaze e cila është derivat i klasës NenpunesMeKomision.

Klasa Nenpunes mundëson metodat set dhe get per variablat e instancës si dhe metodat fitimet ( ) dhe toString( ). Metoda fitimet ( ) e klasës Nenpunes aplikohet në çdo nënpunes

por çdo llogaritje për pagën bëhet në varësi të llojit të nënpunesit. Prandaj, metoda fitimet deklarohet si abstrakte tek klasa bazë Nenpunes sepse implementimi i gatshëm nuk ka kuptim në këtë metodë në këtë klasë. Klasa Nenpunes nuk ka informacion të saktë se si llogaritet fitimi dhe paga e nënpunësit. Çdo klasë do e mbivendosë këtë metodë ( override ) me një implementim konkret. Për të llogaritë fitimet, applikacioni i cakton një referencë Nenpunes për çdo objekt dhe thërret metodën fitimet ( ) me këtë variabël. Në klasën Demo do krijohet një tabelë me variabla Nenpunes, ku secili element është një referencë Nenepunes. Për klasën abstrakte Nenpunes nuk do jetë e mundur krijimi i objekteve, por për arsye se kemi trashëgimi, nënklasat e saj mund të trajtohen si objekte Nenpunes. Aplikacioni do trajtojë çdo element të tabelës dhe do thërrasë metodën fitimet ( ) për çdo objekt Nenpunes. Java i proceson këto thirrje në mënyrë polimorfike. Deklarimi i fitimet ( ) si një metodë abstrakte në klasën Nenpunes bën të mundur që të thirret në çdo referencë Nenpunes dhe detyron çdo klasë të derivuar të klasës Nenpunes që të plotësoje implementimin e kësaj metode.

Metoda toString ( ) në klasën Nenpunes kthen një String e cila përmban emrin, mbiemrin dhe numrin e sigurimeve shoqërore të nenpunesit. Do shohim më poshtë se çdo nënklasë e Nenpunes mbishkruan këtë metodë për të personalizuar paraqitjen e objektit të asaj klase, duke shtuar informacion të posatshëm të klasës.

### Klasa Nenpunes

```
public abstract class Nenpunes
{
	private String emri;
	private String mbiemri;
	private String nrSigShoqerore;


	// konstruktori me tre argumente
	public Nenpunes( String e, String m, String s )
	{
		emri = e;
		mbiemri = m;
		nrSigShoqerore = s;
	}

	public void setEmri( String e )
	{
```

```
            emri = e;
      }


      public String getEmri()
      {
            return emri;
      }

      public void setMbiemri( String m )
      {
            mbiemri = m;
      }

      public String getMbiemri()
      {
            return mbiemri;
      }

      public void setNrSigShoqerore( String s )
      {
            nrSigShoqerore = s;
      }

      public String getNrSigShoqerore()
      {
            return nrSigShoqerore;
      }

      // kthen prezantimin e objektit si string
      @Override
      public String toString()
      {
            return String.format( "%s %s\n numri sigurimeve
shoqerore: %s", getEmri(), getMbiemri(), getNrSigShoqerore() );
      }
      //metoda abstrakte
      public abstract double fitimet(); // nuk ka implementim ketu
}
```

### Klasa NenpunesFullTime

```
public class NenpunesFullTime extends Nenpunes
{
      private double pagaJavore;
```

```
	// konstruktor me kater argumenta
	public NenpunesFullTime( String e, String m, String s, double
paga )
	{
		super( e, m, s ); // kalon tek konstruktori i Nenpunes
		setPagaJavore( paga );
	}

	public void setPagaJavore( double paga )
	{
		if ( paga >= 0.0 )
			pagaJavore = paga;
		else
			throw new IllegalArgumentException("paga javore
duhet te jete >= 0.0" );
	}
	public double getPagaJavore()
	{
		return pagaJavore;
	}

	//llogarit fitimet, kjo metode i mbivendoset metodes baze
	@Override
	public double fitimet()
	{
		return getPagaJavore();
	}
	//kthen paraqitje te String per objektin nenpunes
	@Override
	public String toString()
	{
		return String.format( "nenpunes full time: %s\n%s:
$%,.2f",super.toString(), " paga javore", getPagaJavore() );
	}
}
```

### Klasa NenpunesPartTime

```
public class NenpunesPartTime extends Nenpunes
{
```

```
	private double pagesa; // pagesa per ore
	private double ore; //oret e punes ne jave


	public NenpunesPartTime( String e, String m, String s, double
pagesa, double ore )
	{
		super( e, m, s );
		setPagesa( pagesa );
		setOre( ore );
	}

	public void setPagesa( double p )
	{
		if ( p >= 0.0 )
			pagesa = p;
		else
			throw new IllegalArgumentException("Pagesa per ore
duhet te jete >= 0.0" );
	}


	public double getPagesa()
	{
		return pagesa;
	}

	public void setOre( double h )
	{
		if ( ( h >= 0.0 ) && ( h <= 168.0 ) )
			ore = h;
		else
			throw new IllegalArgumentException("Oret duhet te
jene mes >= 0.0 dhe <= 168.0" );
	}


	public double getOre()
	{
		return ore;
	}




	//llogarit fitimet, metode e mbivendosur nga klasa baze
```

```
	@Override
	public double fitimet()
	{
		if ( getOre() <= 40 )
			return getPagesa() * getOre();
		else
			return 40 * getPagesa() + ( getOre() - 40 ) *
getPagesa() * 1.5;
	}
	//return String representation of HourlyEmployee object
	@Override
	public String toString()
	{
		return String.format( "nenpunes part time: %s\n%s:
$%,.2f; %s: %,.2f", super.toString(), "pagesa per ore ",
getPagesa(), "Ore pune ", getOre() );
	}
}
```

### Kryerja e veprimeve specifike për nënpunësit me komision

Objektet e klasës NenpunesMeKomision dhe NenpunesMeKomisionBaze kryejnë procesim specifik të të dhënave, sepse pagesa e tyre do jete në bazë komisioni, si dhe për komision dhe bazë do ketë një rritje të bazës prej 10%. Kur objektet procesohen në mënyrë polimorfike, nuk shqetësohemi për specifikat por për të rregulluar pagën baze do na duhet të specifikojmë tipin e objektit Nenpunes gjatë kohës së ekzekutimit. Në metodën main të klasës testuese, këtë dallim do e bëjmë duke përdore operatorin instanceOf, për të dalluar nëse objekti është një NenpunesMeKomisionBaze.

### Klasa NenpunesMeKomision

```
//behet fjale per pagese shtese komision per shitje produkti
public class NenpunesMeKomision extends Nenpunes
{
	private double shitjetTotale;
	private double perqindjeKomision;

	public NenpunesMeKomision( String e, String m, String s,
double shitje, double perqindje )
	{
		super( e, m, s );
		setShitjetTotale( shitje );
```

```
            setPerqindjeKomision( perqindje );
      }
      public void setPerqindjeKomision( double perqindje )
      {
            if ( perqindje > 0.0 && perqindje < 1.0 )
                  perqindjeKomision = perqindje;
            else
                  throw new IllegalArgumentException( "Komisioni duhet
te jete mes > 0.0 dhe < 1.0" );
      }


      public double getPerqindjeKomision()
      {
            return perqindjeKomision;
      }

      public void setShitjetTotale( double shitjet )
      {
            if ( shitjet >= 0.0 )
                  shitjetTotale = shitjet;
            else
                  throw new IllegalArgumentException("Shitjet duhet te
jene >= 0.0" );
      }


      public double getShitjetTotale()
      {
            return shitjetTotale;
      }

      //llogarit fitimet; mbishkruan metoden abstrakte te nenpunes
      @Override
      public double fitimet()
      {
            return getPerqindjeKomision() * getShitjetTotale();
      }



      //kthen String info per objektin e kesaj klase
      @Override
      public String toString()
      {
            return String.format( "%s: %s\n%s: $%,.2f; %s: %.2f",
```

```
                "nenpunes me komision ", super.toString(),
                "shitjet totale ", getShitjetTotale(),
                "perqindje komisionit ", getPerqindjeKomision() );
    }
}
```

### Klasa NenpunesMeKomisionBaze

```
public class NenpunesMeKomisionBaze extends NenpunesMeKomision
{
    private double pagaBaze; // pagaBaze per jave


    public NenpunesMeKomisionBaze( String e, String m, String s,
double shitjet, double perqindje, double paga )
    {
        super( e, m, s, shitjet, perqindje );
        setPagaBaze( paga );
    }


    public void setPagaBaze( double paga )
    {
        if ( paga >= 0.0 )
            pagaBaze = paga;
        else
            throw new IllegalArgumentException("Paga baze duhet
te jete >= 0.0" );
    }


    public double getPagaBaze()
    {
        return pagaBaze;
    }




    //llogarit fitimet, metode e mbivendosur
    @Override
    public double fitimet()
    {
        return getPagaBaze() + super.fitimet();
```

```
	}

	//kthen paraqitjen ne string per objektin
	@Override
	public String toString()
	{
		return String.format( "%s %s; %s: $%,.2f",
			"Paga si nenpunes me komision ", super.toString(),
			"paga baze", getPagaBaze() );
	}
}
```

Në klasën PagesaDemo deklarohet dhe inicializohet tabela me nenpunes dhe thërrasin metodën toString duke e thirrë me variablin np. Këtij variabli i caktohet një referencë të ndryshme për çdo nënpunës të tabelës. Output në console tregon se është thirrë metoda e përshtatshme e secilës klasë. Të gjitha thirrjet e metodës toString dhe fitimet janë caktuar gjatë kohës së ekzekutimit, duke u nisë nga tipi i objektit aktual ku po referon variabli np. Ky proces ndrysh quhet dhe lidhje dinamike, ose late binding. Si rezultat i lidhjes dinamike, Java vendos se cilës metodë toString do të thërrasë në kohën e ekzekutimit dhe jo në kohën e kompilimit. Vetëm metodat e klasës Nenpunes mund të thirren nëpërmjet variablit Nenpunes. Një referencë e klasës bazë mund të përdoret për të thirrë metodat e klasës bazë, ndërsa metodat e klasës së derivuar thirren nëpërmjet polimorfizmit.

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


		NenpunesFullTime FTnenpunes =new NenpunesFullTime(
"John", "Smith", "111-11-1111", 800.00 );
		NenpunesPartTime PTnenpunes =new NenpunesPartTime(
"Karen", "Price", "222-22-2222", 16.75, 40 );
		NenpunesMeKomision KSnenpunes =new
NenpunesMeKomision("Sue", "Jones", "333-33-3333", 10000, .06 );
		NenpunesMeKomisionBaze KSBnenpunes =new
NenpunesMeKomisionBaze("Bob", "Lewis","444-44-4444",5000,.04,300 );
		System.out.println( "LLogaritja e pageses per
nenpunesit:\n" );
		System.out.printf( "%s\n%s: $%,.2f\n\n",
		FTnenpunes, " ka pagesen ", FTnenpunes.fitimet() );
		System.out.printf( "%s\n%s: $%,.2f\n\n",
```

```
            PTnenpunes, " ka pagesen ", PTnenpunes.fitimet() );
            System.out.printf( "%s\n%s: $%,.2f\n\n",
            KSnenpunes, " ka pagesen ", KSnenpunes.fitimet() );
            System.out.printf( "%s\n%s: $%,.2f\n\n",
            KSBnenpunes," ka pagesen ", KSBnenpunes.fitimet() );
      // krijimi i nje array me nenpunes, secili nje lloj me vehte
            Nenpunes[] nenpunes = new Nenpunes[ 4 ];
             nenpunes[ 0 ] = PTnenpunes;
            nenpunes[ 1 ] = FTnenpunes;
            nenpunes[ 2 ] = KSnenpunes;
            nenpunes[ 3 ] = KSBnenpunes;
            System.out.println( "pagat e nenpunesit do te procesohen
ne menyre polimorfike :\n" );


            for ( Nenpunes nn: nenpunes )
            {
                  System.out.println(nn); // thirret toString
            // percakton nese eshte nenpunes me komision baze
                  if (nn instanceof NenpunesMeKomisionBaze)
                  {
            // zbritje nga nenpunes ne nenpunes me komision baze

                        NenpunesMeKomisionBaze np =(
NenpunesMeKomisionBaze ) nn;
                        np.setPagaBaze( 1.10 * np.getPagaBaze() );
                        System.out.printf("baga baze me shitje 10%%
rritet me : $%,.2f\n",np.getPagaBaze() );
                  }
             System.out.printf(" ka pagese $%,.2f\n\n",nn.fitimet());
            }
            for ( int j = 0; j < nenpunes.length; j++ )
                  System.out.printf( "Nenpunes %d eshte nje %s\n", j,
            nenpunes[ j ].getClass().getName() );
      }
}
```

Në console :

```
LLogaritja e pageses per nenpunesit:

nenpunes full time: John Smith
 numri sigurimeve shoqerore: 111-11-1111
 paga javore: $800.00
 ka pagesen : $800.00

nenpunes part time: Karen Price
```

```
 numri sigurimeve shoqerore: 222-22-2222
pagesa per ore : $16.75; Ore pune : 40.00
 ka pagesen : $670.00

nenpunes me komision : Sue Jones
 numri sigurimeve shoqerore: 333-33-3333
shitjet totale : $10,000.00; perqindje komisionit : 0.06
 ka pagesen : $600.00

Paga si nenpunes me komision  nenpunes me komision : Bob Lewis
 numri sigurimeve shoqerore: 444-44-4444
shitjet totale : $5,000.00; perqindje komisionit : 0.04; paga baze:
$300.00
 ka pagesen : $500.00

pagat e nenpunesit do te procesohen ne menyre polimorfike :

nenpunes part time: Karen Price
 numri sigurimeve shoqerore: 222-22-2222
pagesa per ore : $16.75; Ore pune : 40.00
 ka pagese $670.00

nenpunes full time: John Smith
 numri sigurimeve shoqerore: 111-11-1111
 paga javore: $800.00
 ka pagese $800.00

nenpunes me komision : Sue Jones
 numri sigurimeve shoqerore: 333-33-3333
shitjet totale : $10,000.00; perqindje komisionit : 0.06
 ka pagese $600.00

Paga si nenpunes me komision  nenpunes me komision : Bob Lewis
 numri sigurimeve shoqerore: 444-44-4444
shitjet totale : $5,000.00; perqindje komisionit : 0.04; paga baze:
$300.00
baga baze me shitje 10% rritet me : $330.00
 ka pagese $530.00

Nenpunes 0 eshte nje polymorphsm.NenpunesPartTime
Nenpunes 1 eshte nje polymorphsm.NenpunesFullTime
Nenpunes 2 eshte nje polymorphsm.NenpunesMeKomision
Nenpunes 3 eshte nje polymorphsm.NenpunesMeKomisionBaze
```

### Përmbledhje e rastit të studimit

Pas studimit të një aplikacioni konkret dhe të plotë kur procesohen objekte të klasave të ndryshme në mënyrë polimorfike, mund të konkludohet se çfarë mund të bëhet dhe çfarë nuk mund të bëhet nëpërmjet klasave baze dhe klasave të derivuara. Megjithëse një objekt i klasës së derivuar është gjithashtu një objekt i klasës bazë, këto dy objekte janë të ndryshëm. Objektet e klasës së derivuar mund të trajtohen si objekte të klasës bazë por meqë klasa e derivuar mund të ketë elemente shtesë të saj, caktimi i një reference të klasës bazë tek një referencë e klasës së derivuar nuk lejohet pa një thirrje eksplicite. Një vlerëdhënie e tillë do linte elementin e klasës së derivuar të papërcaktuar tek objekti i klasës bazë.

Katër mënyrat e caktimit të referencave të klasave bazë dhe të derivuar përmblidhen në :

1. Caktimi i referencës të klasës bazë në një variabël të klasës bazë realizohet natyrshëm.
2. Caktimi i referencës të klasës së derivuar një variabël të klasës së derivuar caktohet natyrshëm.
3. Caktimi i referencës së klasës së derivuar për një variabël të klasës bazë është pa probleme, sepse objekti i klasës së derivuar është nje objekt edhe i klasës bazë. Variabli i klasës bazë mund të përdoret për të referuar vetëm elemente të klasës baze (atribute dhe metoda). Nëse ky kod i referohet vetëm elementëve të klasës së derivuar, kompilatori raporton për gabim.
4. Caktimi i referencës së klasës bazë për variablin e klasës së derivuar shkakton gabim gjatë kohës së kompilimit. Për të shmangë këtë gabim, referenca e klasës bazë duhet të kalojë në mënyrë eksplicite tek tipi i klasës së derivuar. Duhet të përdorni operatorin instanceOf për të bërë këtë kalim vetëm kur objekti është i një klase të derivuar të tipit aktual.

## Leksioni 14

## Ndërfaqja

Një ndërfaqe është nje tip reference në Java. Ajo është e ngjashme me një klasë, por përmban vetëm metoda abstrakte. Një klasë implementon një ndërfaqe, pra trashëgon metodat

abstrakte të ndërfaqes. Ndërfaqja mund të përmbajë konstante, metoda default apo metoda statike. Metodat default apo metodat statike përmbajnë edhe implementimin e tyre.

Shkrimi i një ndërfaqeje është i ngjashëm me shkrimin e një klase. Ndërfaqja dhe klasa ngjajnë në këto këndvështrime:

- Një ndërfaqe përmban një numër çfarëdo metodash.
- Një ndërfaqe shkruhet në skedar .java dhe emri i skedarit është identik me emrin e ndërfaqes.
- Kodi i kompiluar i një ndërfaqes është i ruajtur në një skedar .class
- Ndërfaqet organizohen në paketa.

Dallimi mes ndërfaqes dhe klasës përfshin :

- Nuk mund të inicializohet një ndërfaqe (nuk mund të krijohen objekte të saj)
- Ndërfaqja nuk përmban konstruktor
- Gjitha metodat e ndërfaqes janë abstrakte
- Një ndërfaqe nuk përmban variabla të instancës. Fushat e ndërfaqes deklarohen si statike dhe finale.
- Ndërfaqja nuk trashëgohet nga klasa. Ajo implementohet nga klasa.

### Deklarimi i ndërfaqes

Për deklarimin e ndërfaqes përdoret fjala celes interface. Ndërfaqja gëzon këto cilësi :

- Ndërfaqja është abstrakte, edhe pse nuk cilësohet si e tillë me fjalë çelës.
- Çdo metodë në ndërfaqe është gjithashtu abstrakte, dhe nuk është e nevojshme cilësimi si e tillë me fjalën çelës abstract.
- Metodat e ndërfaqes janë publike.

### Implementimi i ndërfaqes

Kur një klasë implementon një ndërfaqe në Java, ngjason si me nënshkrimin e një kontrate, ku klasa deklaron se është dakord të ketë sjelljen e përcaktuar nga ndërfaqja. Nëse një klasë nuk implementon çdo metodë të ndërfaqes, klasa duhet të deklarohet si klasë abstrakte.

Fjala çelës implements realizon implementimin e ndërfaqes. Kjo fjalë shënohet tek klasa, pikërisht në deklarimin e saj.

### Trashëgimia e ndërfaqeve

Duhen theksuar mundësitë që një klasë mund të implementojë më shumë se një ndërfaqe, si dhe një ndërfaqe mund të trashëgojë një ndërfaqe tjetër, në mënyrë të ngjashme me trashëgiminë e klasave.

Një ndërfaqe trashëgon një tjetër ndërfaqe duke përdorë fjalën çelës extends. Në këtë mënyrë ndërfaqja e derivuar trashëgon gjitha metodat e ndërfaqes bazë.

**Shembull**

```
public interface Sporte {
   public void setSkuadra1(String emri);
   public void setSkuadra2(String emri);
}
public interface Futboll extends Sporte {
   public void NrGolPerSkuadren1(int g);
   public void NrGolPerSkuadren2(int g);
   public void minuta(int min);
}
public interface Hokej extends Sporte {
   public void shenonSkuadra1();
   public void shenonSkuadra2();
   public void minuta(int min);
   public void koheShtese(int min);
}
```

Në këtë shembull, ndërfaqja Hokej ka katër metoda, por dy të tjera i trashëgon nga ndërfaqja Sporte, prandaj një klasë që implementon ndërfaqen Hokej do duhet të implementojë të gjashtë metodat e saj. Në mënyrë analoge, çdo klasë që implementon ndërfaqen Futboll do duhet të implementojë tre metodat e Futboll si dhe dy metodat e po kësaj ndërfaqeje, por që i ka trashëgimi nga ndërfaqja Sporte.

### Trashëgimia e shumëfishtë e ndërfaqeve

Në Java, një klasë mund të trashëgojë vetëm nga një klasë bazë. Trashëgimia e shumëfishtë nuk lejohet për klasat. Ndërfaqet nuk janë klasa. Ato mund të trashëgojë cilësitë e disa ndërfaqeve bazë. Fjala çelës extends përdoret vetëm njëherë por ndërfaqet bazë ndahen me presje. Për shembull, ndërfaqja Hokej mund të jetë trashëgimi e ndërfaqes Sporte dhe ndërfaqes Evente, njëkohësisht. Ky fakt deklarohet si më poshtë :

```
public interface Hockey extends Sports, Event
```

## Leksioni 15

## Rast Studimi : Sistemi i pagesës, implementim i ndërfaqeve

Supozoni se kompania e përfshirë në rastin e studimit më lart kërkon të kryejë disa veprime financiare në një aplikacion të vetëm për llogaritë e pagueshme. Kompania do përfshijë rrogat e nënpunësve gjithashtu do paguajë edhe faturat për mallrat e blerë. Megjithëse këto dy gjera janë pa lidhje me njëra tjetrën (nënpunesi dhe fatura) të dy veprimet kanë të përbashkët një lloj pagese nga ana e kompanisë. Për një nënpunës, pagesa merr formën e rrogës së nënpunësit, kurse për faturën pagesa i referohet shumës totale të mallrave të blerë. A mund ti ndërthurim këto dy gjëra të ndryshme në një aplikacion polimorfik? A ofron Java mundësinë që dy klasa që nuk kanë lidhje trashëgimie, të kenë të përbashkët një grup metodash? Ndërfaqet në Java bëjnë të mundur këtë gjë.

Objektet komunikojnë nëpërmjet ndërfaqes. Një ndërfaqe në Java është cilësuar si një bashkësi metodash që mund të thirren më një objekt, duke i treguar të kryejë disa veprime apo të kthejë një të dhënë. Shembulli i këtij rasti studimi na prezanton më ndërfaqen EPagueshme e cila përshkruan funksionalitetin e çdo objekti që mund të paguhet, prandaj ofron një metodë që përcakton një sasi parash që paguhen. Deklarimi i ndërfaqes fillon me fjalën çelës interface, dhe përmban vetëm konstante dhe metoda abstrakte. Ndryshe nga klasat, çdo ndërfaqe ka elementë publikë dhe nuk duhet të përmbajë ndonjë detaj të implementimit, si për shembull variabla të instancës. Të gjitha metodat e deklaruara në një ndërfaqe janë metoda publike dhe abstrakte, dhe gjithashtu fushat përcaktohen si publike, statike dhe final.

Klasa konkrete që implementon një ndërfaqe, duhet të deklarojë çdo metodë të ndërfaqes njëlloj siç është përcaktuar në ndërfaqe. Një klasë që nuk implementon të gjitha metodat e ndërfaqes do deklarohet si një klasë abstrakte.

### Zhvillimi i strukturës hierarkike dhe ndërfaqes EPagueshme

Për të ndërtuar një aplikacion që përcakton pagën e nënpunësit si dhe pagesën e faturave, së pari na duhet të krijojmë një ndërfaqe EPagueshme, e cila përmban metodën getPerTuPaguar që kthen një vlerë double për sasinë që duhet paguar për objektin e klasës ku implementohet ndërfaqja. Metoda getPerTuPaguar është një version i përgjithshëm e cila mund të përdoret për një rang të gjerë objektesh që mund të kenë relacion mes tyre ose jo. Pas deklarimit të ndërfaqes EPagueshme, implementohet klasa Fatura, e cila implementon ndërfaqen EPagueshme. Më pas modifikohet klasa Nenpunes në mënyrë të tillë që të implementojë ndërfaqen EPagueshme. Në fund, modifikohet klasa NenpunesFullTime për tu përshtatur me hierarkinë EPagueshme duke riemëruar metodën fitimet në getPerTuPaguar. Klasat Fatura dhe Nenpunes përfaqësojnë gjëra që kompania duhet të jetë e aftë të llogarisë në sasi pagese. Të dyja klasat implementojnë një ndërfaqe të përbashkët prandaj programi mund të thërrasë metodën getPerTuPaguar edhe me objekt fature, edhe me objekt nenpunes. Kjo bën të mundur një procesim polimorfik të Fatures dhe Nenpunes të aplikacionit të kërkuar

### Ndërfaqja EPagueshme

Kjo ndërfaqe në këtë rast ka vetëm një metodë: getPerTuPaguar, e cila nuk ka parametra. Ndërfaqet mund të kenë fusha të cilat janë statike dhe finale, por në këtë rast ndërfaqja nuk ka fushë të tillë.

```
public interface EPagueshme{
    double getPerTuPaguar();
}
```

### Klasa Fatura

Në vijim do krijojmë klasën Fatura, e cila përfaqëson një shembull të thjeshtë informacioni per tu printuar në faturë. Klasa përmban variabla të instancës për id, përshkrimin, sasine dhe cmimPerArtikull. Klasa Fature përmban konstruktorin e klasës, metodat set dhe get për variablat e instancës si dhe metodën toString që kthen një përshkrim String për objektin e klasës Fatura.

```
public class Fatura implements EPagueshme{
    private String id;
    private String pershkrim;
    private int sasia;
    private double cmimPerArtikull;

    // konstruktori
    public Fatura(String i, String p, int s, double c){
        setId(i);
        setPershkrim(p);
        setSasia(s);
        setCmimPerArtikull(c);
    }
    // SETTERS
    public void setId(String i){
        id = i;
    }
    public void setPershkrim(String p){
       pershkrim = p;
    }
    public void setSasia(int s){
        if(s >= 0)
            sasia = s;
        else
            throw new IllegalArgumentException("Sasia duhet te jete
>= 0");
    }
    public void setCmimPerArtikull(double c){
        if(c >= 0.0)
            cmimPerArtikull = c;
        else
            throw new IllegalArgumentException("Cmimi duhet te jete
>= 0");
    }
    // GETTERS
```

```
    public String getId(){
        return id;
    }
    public String getPershkrim(){
        return pershkrim;
    }
    public int getSasia(){
        return sasia;
    }
    public double getCmimPerArtikull(){
        return cmimPerArtikull;
    }
    // kthen paraqitjen ne String te fatures
    @Override
    public String toString(){
        return String.format("%s: \n%s: %s (%s) \n%s: %d \n%s:
$%,.2f",
                "Fatura", " nr ", getId(), getPershkrim(),
                "sasia", getSasia(), "cmim per artikull",
getCmimPerArtikull());
    }
    // metoda e implementuar
    @Override
    public double getPerTuPaguar(){
        return getSasia() * getCmimPerArtikull();
    }
}
```

Në vijim do modifikojmë klasën Nenpunes në mënyrë të tillë që të implementojë ndërfaqen EPagueshme. Gjithashtu duhet riemërtuar metoda fitimet në getPerTuPaguar. Kjo metodë është pa implementim konkret tek klasa Nenpunes. Për këtë arsye, klasa Nenpunes do deklarohet si abstrakte. Çdo klasë që trashëgohet nga klasa Nenpunes do përmbajë implementim të metodës getPerTuPaguar.

```
public abstract class Nenpunes implements EPagueshme{
    private String emri;
    private String mbiemri;
    private String nrSigShoqerore;



 // konstruktori me tre argumente
     public Nenpunes( String e, String m, String s )
     {
          emri = e;
```

```
            mbiemri = m;
            nrSigShoqerore = s;
      }
      public void setEmri( String e )
      {
            emri = e;
      }

      public String getEmri()
      {
            return emri;
      }

      public void setMbiemri( String m )
      {
            mbiemri = m;
      }

      public String getMbiemri()
      {
            return mbiemri;
      }

      public void setNrSigShoqerore( String s )
      {
            nrSigShoqerore = s;
      }

      public String getNrSigShoqerore()
      {
            return nrSigShoqerore;
      }









      // kthen prezantimin e objektit si string
      @Override
      public String toString()
      {
```

```
            return String.format( "%s %s\n numri sigurimeve
shoqerore: %s", getEmri(), getMbiemri(), getNrSigShoqerore() );
      }
     // Vini re: Nuk do implementojme nderfaqen e pagueshme ketu
       // klasa deklarohet si abstrakte
}
```

Klasa NenpunesFullTime është një version i modifikuar si një trashëgimi e klasës Nenpunes, dhe që implementon metodën getPerTuPaguar të ndërfaqes EPagueshme. Ky version është i ngjashëm me klasën me të njëjtin emër të trajtuar më lart, por metoda fitimet është tashmë një implementim në getPerTuPaguar.

Kur një klasë implementon një ndërfaqe, aplikohet relacioni is-a. Ky fakt interpretohet : klasa Nenpunes implementon EPagueshme, pra mund të themi se Nenpunes është një element i pagueshëm. Gjithashtu objektet që trashëgohen nga Nenpunes, janë gjithashtu elemente të pagueshëm. Të tillë janë dhe objektet e klasës NenpuneFullTime. Objektet e klasës së derivuar, gjithashtu trajtohen si objekte të tipit të ndërfaqes. Prandaj mund të caktojmë një referencë të objektit të NenpunesFullTime për te variabli Nenpunes. Gjithashtu mund të caktojmë një referencë të objektit NenpunesFullTime për tek një variabël EPagueshme. Në mënyrë analoge, ndërfaqja implementohet nga klasa Fatura, kështu që një objekt Fature është gjithashtu një objekt EPagueshme, dhe mund të caktojmë një referencë të objektit Fatura për tek variabli EPagueshme.

```
public class NenpunesFullTime extends Nenpunes{

      private double pagaJavore;

      // konstruktor me kater argumenta
      public NenpunesFullTime( String e, String m, String s, double
paga )
      {
            super( e, m, s ); // kalon tek konstruktori i Nenpunes
            setPagaJavore( paga );
      }

      public void setPagaJavore( double paga )
      {
            if ( paga >= 0.0 )
```

```
                pagaJavore = paga;
            else
                throw new IllegalArgumentException("paga javore
duhet te jete >= 0.0" );
        }
        public double getPagaJavore()
        {
            return pagaJavore;
        }

        //kthen paraqitje te String per objektin nenpunes
            @Override
            public String toString()
            {
                return String.format( "nenpunes full time: %s\n%s:
$%,.2f",    super.toString(), " paga javore", getPagaJavore() );
            }

    // llogarit fitimet; implementon nderfaqen qe ishte abstrakte
tek klasa baze nenpunes

    @Override
    public double getPerTuPaguar(){
        return getPagaJavore();
    }
}
```

### Klasa PagesaDemo

Kjo klasë përmban funksionin main i cili në implementim të tij tregon se si ndërfaqja EPagueshme mund të procesohet në mënyrë polimorfike për dy lloj objektesh, fatura dhe nenpunes, në një aplikacion të vetëm.

```
public class PagesaDemo{
    public static void main(String[] args){
        // krijon tabele me elemente per tu paguar
        EPagueshme[] objTePagueshme = new EPagueshme[4];

        // plotesohet tabela me elemente fature ose nenpunes per tu
paguar
        objTePagueshme[0] = new Fatura("01234", "seat", 2, 375.00f);
        objTePagueshme[1] = new Fatura("56789", "tire", 4, 79.95f);
        objTePagueshme[2] = new NenpunesFullTime("John", "Smith",
"111-11-1111", 800.00f);
```

```
        objTePagueshme[3] = new NenpunesFullTime("Lisa", "Barnes",
"888-88-8888", 1200.00f);

        System.out.println("Fatura dhe nenpunes procesohen ne menyre
polimorf:\n");

        for(EPagueshme pg : objTePagueshme){

            System.out.printf("%s \n%s: $%,.2f\n\n",
pg.toString(),"per tu paguar", pg.getPerTuPaguar());
        }
    }
}
```

Në console:

```
Fatura dhe nenpunes procesohen ne menyre polimorf:


Fatura:
 nr : 01234 (seat)
sasia: 2
cmim per artikull: $375.00
per tu paguar: $750.00


Fatura:
 nr : 56789 (tire)
sasia: 4
cmim per artikull: $79.95
per tu paguar: $319.80



nenpunes full time: John Smith
 numri sigurimeve shoqerore: 111-11-1111
 paga javore: $800.00
per tu paguar: $800.00
```

```
nenpunes full time: Lisa Barnes
 numri sigurimeve shoqerore: 888-88-8888
 paga javore: $1,200.00
per tu paguar: $1,200.00
```

# Programim në Java në rrjet

Termi "programim në rrjet" i referohet shkrimit të programeve që ekzekutohen në disa paisje që janë të lidhur në rrjet. Paketa java.net e J2SE API përmban grumbull klasash dhe ndërfaqesh që lejojnë të shkruhen programe që fokusohen në komunikimin mes këtyre paisjeve.
Paketa java.net suporton këto dy protokolle kryesore:

1. TCP është shkurtim i Transmission Control Protocol, e cila lejon komunikim të besueshëm mes dy aplikacioneve në rrjet. TCP përdoret në Internet Protocol, dhe i referohemi si TCP/IP.
2. UDP është shkurtim i User Datagram Protocol, i cili është një protokoll connection-less, dhe lejon transmetimin e paketave të të dhënave ndërmjet aplikacioneve.

Në këtë kapitull do njihemi më gjërë me Socket Programming, që është një koncept shumë i përdorur në rrjete. Socket mundësojnë mekanizëm komunikimi mes dy kompjuterave, nëpërmjet protokollit TCP. Një program klient krijon një socket në skajin e komunikimit, dhe pret që ky socket të lidhet me serverin. Kur kryhet kjo lidhje, serveri krijon një objekt socket në skajin e tij të komunikimit. Në këtë moment, klienti dhe serveri mund te shkruajnë dhe të lexojnë në socket.
Klasa java.net.Socket përfaqson një socket, dhe klasa java.net.ServerSocket ofron një mekanizëm që programi në server të shohë klientët dhe të krijojë lidhjet me to.
Më poshtë jepen hapat që duhen kryer për të krijuar një lidhjet TCP mes dy kompjuterave, duke përdore sockets :

- Serveri inicializon një objekt ServerSocket, duke përcaktuar se në cilën portë do komunikojë.
- Serveri thërret metodën accept( ) të klasës ServerSocket. Kjo metodë pret derisa një klient lidhet me serverin në portën e caktuar.
- Ndërkohë që serveri pret, klienti inicializon një objekt Socket, duke specifikuar emrin e serverit dhe numrin e portës ku do bëhet komunikimi.
- Konstruktori i klases Socket përpiqet të lidhet me klientin në serverin dhe numrin e portës të përcaktuar. Në qoftë se reazohet komunikimi, klienti ka një objekt Socket me të cilin mund të komunikojë me serverin.
- Nga ana tjetër e komunikimit, metoda accept( ) kthen një referencë për socket në server që është lidhur me socket të klientit.

Pasi është kryer lidhja, komunikimi kryhet duke përdorë rrjedhat e të dhënave, input dhe output. Secili socket ka një OutputStream dhe një InputStream. OutputStream i klientit është i lidhur me InputStream i serverit, dhe InputStream i klientit është i lidhur me OutputStream i serverit.
TCP është protokoll komunikimi i dy anëshëm, prandaj të dhënat mund të dërgohen nëpërmejt të dy rrjedhave njëkohësisht. Më poshtë janë klasat e nevojshme që ofrojnë komplet setin e metodave që implementojnë sockets.

## Klasa ServerSocket

Klasa java.net.ServerSocket përdoret nga aplikacionet e server për të dëgjuar portën dhe për kërkesat e klientëve.
Kjo klasë ka katër konstruktorë, të cilët përshkruhen në tabelën më poshtë.

| Përshkrimi |
| --- |
| `public ServerSocket(int port) throws IOException`<br>Ky konstruktor krijon një server socket me një portë specifike. Një përjashtim ndodh nëse porta është e zënë nga një aplikacion tjetër. |
| `public ServerSocket(int port, int backlog) throws IOException`<br>Ky konstruktor krijon një socket me një portë specifike, ndërsa parametri backlog specifikon se sa klienta do ruhen në pritje në një rradhë. |
| `public ServerSocket(int port, int backlog, InetAddress address) throws IOException`<br>Ky konstruktor krijon një socket duke përcaktuar porten, backlog, por edhe një parameter shtesë. Ky parametër address specifikon adresën lokale IP ku do lidhet socket. Kjo adresë përdoret për servera që mund të kenë disa adresa IP, duke lejuar në këtë mënyrë që të specifikohen se cila adresë IP të pranojë kërkesën e klientit. |
| `public ServerSocket() throws IOException`<br>Ky konstruktor krijon një server socket të palidhur më ndonjë portë. Kur thirret ky konstruktor, përdoret metoda `bind()` për të lidhë server socket. |

Në qoftë se konstruktori i ServerSocket nuk lëshon ndonjë përjashtim, kjo do të thotë se aplikacioni ka lidhur me sukses socket dhe portën e kërkuar, dhe është gati për kërkesat e klientave.
Më poshtë janë disa nga metodat e klasës ServerSocket.

| Përshkrimi i metodave |
| --- |
| `public int getLocalPort()`<br>Kjo metodë kthen si përgjigje portën me të cilën është lidhur socket server. Ajo përdoret nqs kalojmë vlerën 0 si numër porte në një konstruktor, dhe të lëmë serverin të gjejë një portë lirshëm. |
| `public Socket accept() throws IOException`<br>Kjo metodë pret për një klient që të kërkojë të lidhet me serverin, në portën e përcaktuar, ose socket ti mbarojë koha në dispozicion, duke u nisur nga koha e përcaktuar nga metoda setSoTimeout. Në rast të kohë të palimituar, kjo metodë bllokohet pa përcaktim. |
| `public void setSoTimeout(int timeout)`<br>Kjo metodë përcakton vlerën se sa do presë socket për klientin, gjatë metodës accept |
| `public void bind(SocketAddress host, int backlog)`<br>Kjo metodë lidh socket me një server specifik, dhe një portë të objektit SocketAddress. Kjo metodë përdoret nqs keni inicializuar ServerSocket duke përdorë konstruktorin pa argumenta. |

Kur ServerSocket thërret metodën accept, kjo metodë nuk kthen asgjë derisa të lidhet një klient. Pasi klienti është lidhur, ServerSocket krijon një Socket në një portë të papërcaktuar dhe kthen referencën për këtë Socket. Një lidhje TCP ekziston mes klientit dhe serverit, dhe mund të nisë komunikimi.

## Klasa Socket

Klasa java.net.Socket përfaqson një socket që mund të përdoret edhe nga klienti edhe nga serveri, për të komunikuar me njëri tjetrin. Klienti merr një objekt socket duke e inicializuar, ndërsa serveri e merr një objekt Socket si vlerë kthyese e metodën accept.
Klasa Socket ka pesë konstruktorë që përdor klienti për tu lidhur me një server.

| Përshkrimi |
| --- |
| `public Socket(String host, int port) throws UnknownHostException, IOException`<br>Ky konstruktor lidh serverin e përcaktuar në një portë të përcaktuar. Në qoftë se ky konstruktor nuk lëshon një përjashtim, lidhja është kryer me sukses dhe klienti është lidhë me serverin. |
| `public Socket(InetAddress host, int port) throws IOException`<br>Ky konstruktor është identik me konstruktorin më lart, vecse hosti është përcaktuar nga InetAddress object. |
| `public Socket(String host, int port, InetAddress localAddress, int localPort) throws IOException`<br>Ky konstruktor lidhet me një host dhe një portë specifike, duke krijuar një socket në local host në adresën specifike dhe në portën specifike. |
| `public Socket(InetAddress host, int port, InetAddress localAddress, int localPort) throws IOException`<br>Ky konstruktor është identik me konstruktorin më lart, vetëm se hosti është shënuar nga një objekt InetAddress, në vend të një stringe. |
| `public Socket()`<br>Ky konstruktor krijon një socket të palidhur. Duhet thirrë metoda connect për të lidhë socket me serverin. |

Kur konstruktori Socket kthen si vlerë një objekt të ri socket, ai përpiqet ta lidhë këtë objekt me serverin dhe portën specifike.
Më poshtë janë listuar disa metoda të klasës Socket. Vini re se edhe klienti edhe serveri kanë objektin socket, prandaj këto metoda mund të thirren edhe nga klienti edhe nga serveri.

| Përshkrimi |
| --- |
| `public void connect(SocketAddress host, int timeout) throws IOException`<br>Kjo metodë lidh socket me hostin e përcaktuar. Ajo thirret vetëm kur socket është krijuar nga konstruktori pa argumenta, dhe duhet bërë lidhja. |
| `public InetAddress getInetAddress()`<br>Kjo metodë kthen adresën e kompjuterit me të cilin është lidhë ky socket. |
| `public int getPort()`<br>Kjo metodë kthen portën ku është lidhë socket. |
| `public int getLocalPort()`<br>Kjo metodë kthen portën e kompjuterit lokal ku është lidhë socket. |
| `public SocketAddress getRemoteSocketAddress()`<br>Kjo metodë kthen adresën e remote socket |
| `public InputStream getInputStream() throws IOException`<br>Kjo metodë kthen një input stream të socket. Kjo rrjedhë është e lidhur me një output stream të remote socket |
| `public OutputStream getOutputStream() throws IOException`<br>Kjo metodë kthen output stream të socket. Kjo rrjedhë është e lidhur me një input stream të remote socket. |
| `public void close() throws IOException`<br>Kjo metodë mbyll socket dhe bën që ky objekt të mos mundet të lidhet më me serverin. |

### Klasa InetAddress

Kjo klasë përfaqson një adresë të Internet Protocol (IP address). Më poshtë jepen metodat e përdorura gjatë programimit me socket.

| Përshkrimi |
|---|
| `static InetAddress getByAddress(byte[] addr)`<br>Kjo metodë kthen një objekt InetAddress në formën e IP address |
| `static InetAddress getByAddress(String host, byte[] addr)`<br>Kjo metodë krijon një InetAddress duke u bazuar nga argumentat e dhënë host dhe adresë. |
| `static InetAddress getByName(String host)`<br>Kjo metodë percakton adresën IP të një hosti, i cili jepet si argument |
| `String getHostAddress()`<br>Kjo metodë kthen një stringë IP adresë |
| `String getHostName()`<br>Kjo metodë kthen emrin e hostit për këtë IP adrese |
| `static InetAddress InetAddress getLocalHost()`<br>Kjo metodë kthen adresën e hostit lokal |
| `String toString()`<br>Kjo metodë konverton IP adresë në një String |

### Shembull : Socket Client dhe Server

Në këtë shembull është ndërtuar një aplikacion client, i cili lidhet me një server duke përdorë socket dhe duke dërguar mesazh përshëndetës, dhe pret për një përgjigje.

```
import java.net.*;
import java.io.*;

public class GreetingClient {

   public static void main(String [] args) {
        String serverName="GreetingServer";
        int port=6066;
       try {
          System.out.println("Duke u lidhur me serverin " +
serverName + " ne porten " + port);
          Socket client = new Socket(serverName, port);

          System.out.println("Sapo u lidh me " +
client.getRemoteSocketAddress());
          OutputStream outToServer = client.getOutputStream();
          DataOutputStream out = new DataOutputStream(outToServer);

          out.writeUTF("Pershendetje nga " +
client.getLocalSocketAddress());
          InputStream inFromServer = client.getInputStream();
          DataInputStream in = new DataInputStream(inFromServer);

          System.out.println("Serveri thote " + in.readUTF());
          client.close();
```

```
      }catch(IOException e) {
         e.printStackTrace();
      }
   }
}

import java.net.*;
import java.io.*;

public class GreetingServer extends Thread {
private ServerSocket serverSocket;

public GreetingServer(int port) throws IOException {
   serverSocket = new ServerSocket(port);
   serverSocket.setSoTimeout(3000000);
}

public void run() {
   while(true) {
      try {
         System.out.println("Duke pritur per klientin ne porten " +
            serverSocket.getLocalPort() + "...");
         Socket server = serverSocket.accept();

         System.out.println("Sapo u lidh me  " +
server.getRemoteSocketAddress());
         DataInputStream in = new
DataInputStream(server.getInputStream());

         System.out.println(in.readUTF());
         DataOutputStream out = new
DataOutputStream(server.getOutputStream());
         out.writeUTF("Lidhja me serverin  " +
server.getLocalSocketAddress()
        + "u krye me sukses \nDalje...");
         server.close();

      }catch(SocketTimeoutException s) {
         System.out.println("Socket timed out!");
         break;
      }catch(IOException e) {
         e.printStackTrace();
         break;
      }
   }
}
```

```
public static void main(String [] args) {
   int port = 6066;
   try {
      Thread t = new GreetingServer(port);
      t.start();
   }catch(IOException e) {
      e.printStackTrace();
   }
}
```

Në console të klasës Server :

```
Duke pritur per klientin ne porten 6066...
```

Për klasën Klient

```
Duke u lidhur me serverin ne porten 6066
Sapo u lidh me localhost/127.0.0.1:6066
Serveri thote Lidhja me serverin u krye me sukses
Dalje...
```

# Multithreading

Java është një gjuhë programimi që ofron mundësinë e krijimit të një aplikacioni me shumë thread-e. Një program i tillë përmban dy ose më shumë pjesë që ekzekutohen në mënyrë konkurente dhe secila pjesë kryen një detyrë të ndryshme, duke përdorë resurset në mënyrë eficente.
Me përkufizim, multitasking ndodh kur disa procese ndajnë resurset e përbashkëta, si psh CPU. Multithreading zgjeron idenë e multitasking në aplikacionet ku mund të ndahen disa veprime në threade individuale. Secili prej këtyre threadeve mund të ekzekutohet në paralel. Sistemi i shfrytëzimit ndan kohën ndërmjet proceseve, por edhe kohën ndërmjet thread të aplikacionit. Multithreading ju jep mundësinë të zhvilloni aplikacione në mënyrë të tillë që disa aktivitete procesohen paralelisht brenda të njëjtit program.

## Cikli i jetës i thread

Një thread kalon disa etapa gjatë ciklit të jetës së tij.

- Thread i ri: një thread nis ciklin e jetës në këtë gjendje, dhe mbetet këtu derisa programi nis ekzekutimin e këtij thread.
- Të ekzekutueshme: Pasi thread ka nisë ekzekutimin, ai kalon në gjendje të ekzekutueshm. Një thread në këtë status është duke kryer detyrat që i janë caktuar.
- Në pritje: Ndonjëherë, një thread kalon në gjendjen duke pritur, pasi ai pret një tjetër thread për të kryer një veprim. Një thread kalon në përsëri në gjendjen duke u ekzekutuar, kur threadi tjetër i dërgon një sinjal për të vazhduar ekzekutimin.
- Në pritje me afat kohe: Një thread në ekzekutim kalon në një gjendje pritjeje me afat kohor, në një interval të caktuar kohe. Një thread në këtë gjendje kalon në gjendje të ekzekutueshme përsëri, kur intervali i kohës ka mbaruar, ose kur eventi që po priste ka ndodhur.
- Të përfunduar: Një thread kalon nga gjendja e ekzekutueshme në gjendje të përfunduar, kur thread ka kryer detyrën e tij.

## Prioriteti i thread

Cdo thread në Java ka një prioritet, që ndihmon sistemin e shfrytëzimit të caktojërendin në të cilin skedulohet thread.
Prioritetet e Java janë mes prioritetit MIN_PRIORITY (me vlerë numerike 1) dhe MAX_PRIORITY (me vlerë 10). NJë threadi të ri i jepet automatikisht prioriteti NORM_PRIORITY (me vlerë 5).
Thread me prioritet të lartë janë më të rëndësishme dhe duhet tiu caktohet procesori më parë se threaded me prioritet të ulët.

## Krijimi i thread duke implementuar ndërfaqen Runnable

Në qoftë se kërkoni që klasa në Java të ekzekutohet si thread, mund të implementoni ndërfaqen Runnable. Për këtë ndiqen këto hapa:

### Hapi 1:

Duhet të implementoni një metodë `run()` e cila mundësohet nga ndërfaqja Runnable. Kjo metodë është një pikë hyrëse për thread dhe komplet logjika e detyrës do vendoset në këtë metodë. Sintaksa e run është si mëposhtë:

```
public void run( )
```

### Hapi 2:

Në këtë hap do inicializohet një objekt Thread duke përdorë konstruktorin :

```
Thread(Runnable threadObj, String threadName);
```

Objekti threadObj është një instancë e klasës që implementon Runnable dhe threadName është emri i thread të ri.

Hapi 3:

Pasi është krijuar thread, mund të thirret metoda `start()`, e cila kryen një thirrje të metodës `run()`. Sintaksa e mëposhtme thërret këtë metodë:

```
void start();
```

**Shembull**

```
class RunnableDemo implements Runnable {
         private Thread t;
         private String emriThrd;
         RunnableDemo( String e) {
            emriThrd = e;
            System.out.println("Duke u krijuar " +  emriThrd );
         }

         public void run() {
            System.out.println("Duke u ekzekutuar " +  emriThrd );
            try {
             for(int i = 4; i > 0; i--) {
               System.out.println("Thread: " + emriThrd + ", " + i);
               Thread.sleep(50);
             }
            }catch (InterruptedException e) {
               System.out.println("Thread " +  emriThrd + " u
nderpre.");
            }
            System.out.println("Thread " +  emriThrd+ " duke
perfunduar.");
         }

         public void start () {
            System.out.println("Filloi " +  emriThrd );
            if (t == null) {
               t = new Thread (this, emriThrd);
               t.start ();
            }
         }
      }

      public class TestThread {
         public static void main(String args[]) {
            RunnableDemo R1 = new RunnableDemo( "Thread-1");
            R1.start();
            RunnableDemo R2 = new RunnableDemo( "Thread-2");
            R2.start();
         }
      }
```

Në console:

```
Duke u krijuar Thread-1
Filloi Thread-1
Duke u krijuar Thread-2
Filloi Thread-2
Duke u ekzekutuar Thread-1
Thread: Thread-1, 4
Duke u ekzekutuar Thread-2
Thread: Thread-2, 4
Thread: Thread-1, 3
Thread: Thread-2, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-1 duke perfunduar.
Thread Thread-2 duke perfunduar.
```

### Krijimi i thread si trashëgimi i klasës Thread

Mënyra e dytë për të krijuar një thread është krijimi i një klase të re të trashëguar nga klasa Thread duke përdorë këto dy hapa të thjeshtë. Kjo mënyrë ofron një fleksibilitet në menaxhimin e disa threadeve duke përdorë metodat e klasës Thread.

Hapi 1:

Duhet bërë override metoda run() e cila është e trashëguar nga klasa Thread. Kjo metodë mundëson një pikë starti për dhe do përmbajë logjikën funksionale të thread. Sintaksa e metodës run është :

```
public void run( )
```

Hapi 2:

Pasi është krijuar objekti Thread, mund të thirret metoda start, e cila thërret metodën run. Sintaksa e metodës start është :

```
void start( );
```

**Shembull :**

```
class MyThread extends Thread {
         private Thread t;
         private String thrd;

         MyThread( String emri) {
            thrd = emri;
            System.out.println("Duke inicializuar " +  thrd );
         }

         public void run() {
            System.out.println("Duke u ekzekutuar " +  thrd );
            try {
               for(int i = 4; i > 0; i--) {
                  System.out.println("Thread: " + thrd + ", " + i);
                  Thread.sleep(50);
```

```
            }
         }catch (InterruptedException e) {
            System.out.println("Thread " +  thrd + " u
nderpre.");
         }
         System.out.println("Thread " +  thrd + " perfundoi.");
      }

      public void start () {
         System.out.println("Duke filluar " +  thrd );
         if (t == null) {
            t = new Thread (this, thrd);
            t.start ();
         }
      }
   }

   public class ThreadDemo {

      public static void main(String args[]) {
        MyThread T1 = new MyThread( "Thread-1");
         T1.start();

         MyThread T2 = new MyThread( "Thread-2");
         T2.start();
      }
   }
```

Në console :

```
Duke inicializuar Thread-1
Duke filluar Thread-1
Duke inicializuar Thread-2
Duke filluar Thread-2
Duke u ekzekutuar Thread-1
Thread: Thread-1, 4
Duke u ekzekutuar Thread-2
Thread: Thread-2, 4
Thread: Thread-1, 3
Thread: Thread-2, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-1 perfundoi.
Thread Thread-2 perfundoi.
```

## Metodat e klasës Thread

Më poshtë jepen metodat më kryesore të klasës Thread

| Përshkrimi |
|---|
| `public void start()`<br>Kjo metodë nis një thread në një ekzekutim të vecantë dhe më pas thërret metodën run () për këtë thread. |
| `public void run()`<br>Në qoftë se ky objekt është inicializuar nga një target i Runnable, metoda run() e thërrasin për atë objekt Runnable. |
| `public final void setName(String name)`<br>Kjo metodë ndryshon emrin e objektit Thread. Gjithashtu ka një metodë getName() për të marrë emrin e objektit Thread. |
| `public final void setPriority(int priority)`<br>Kjo metodë cakton një prioritet për objektin Thread. Vlerat e mundshme janë nga 1 deri në 10. |
| `public final void setDaemon(boolean on)`<br>Kjo metodë i jep thread një parametër boolean, ku true shënon që thread është një deamon thread. |
| `public final void join(long millisec)`<br>Kjo metodë thirret nga një thread për një thread tjetër, dhe shkakton bllokim të thread aktual derisa thread i dytë të përfundojë, ose të kalojnë një kohë me një numër i përcaktuar milisekondash |
| `public void interrupt()`<br>Kjo metodë ndërpret threadin, duke shkaktuar ekzekutimin e një tjetër thread. |
| `public final boolean isAlive()`<br>Kjo metodën kthen true nqs threadi është ende pa përfunduar detyrën e tij. |

Metodat më lart thirren për një objekt Thread të vecantë. Kurse metodat e mëposhtme janë statike. Thirrja e tyre shkakton veprim mbi thread që po ekzekutohet.

| Përshkrimi |
|---|
| `public static void yield()`<br>Kjo metodë bën që thread që po ekzekutohet të lëshojë kontrollin e CPU tek një tjetër thread me të njëjtin prioritet. |
| `public static void sleep(long millisec)`<br>Kjo metodë bën që threadi që po ekzekutohet të bllokohet në një numër të dhënë milisec. |
| `public static boolean holdsLock(Object x)`<br>Kjo metodë kthen vlerën true nqs thread që po ekzekutohet mban kycin e Objektit x të dhënë. |
| `public static Thread currentThread()`<br>Kjo metodë kthen një referencë të threadit që po ekzekutohet, i cili në fakt është thread që thirri këtë metodë. |
| `public static void dumpStack()`<br>Kjo metodë printon stack trace të threadit që po ekzekutohet, i cili duhet kur do bëhet debug në një aplikacion multithread. |

**Shembull :**
Ky shembull krijon thread si implementim i Runnable.

```
public class DisplayMessage implements Runnable {
         private String message;

         public DisplayMessage(String message) {
            this.message = message;
         }

         public void run() {
            while(true) {
               System.out.println(message);
            }
         }
}
```

Klasa mëposhtë krijon thread si trashëgimi i e klasës Thread

```
public class GjejNr extends Thread {
         private int nr;
         public GjejNr(int nr) {
            this.nr = nr;
         }

         public void run() {
            int numrues = 0;
            int gjej = 0;
            do {
               gjej = (int) (Math.random() * 100 + 1);
               System.out.println(this.getName() + " here " + gjej);
               numrues++;
            } while(gjej != nr);
            System.out.println("** Sakte!" + this.getName() + "in" +
numrues + "here.**");
         }
      }
```

Më poshtë është klasa Demo e cila përdor të dyja këto klasa Thread.

```
public class ThreadClassDemo {

         public static void main(String [] args) {
            Runnable sms1 = new DisplayMessage("Pershendetje");
            Thread thread1 = new Thread(sms1);
            thread1.setDaemon(true);
            thread1.setName("pershendetje");
            System.out.println("Nisje e thread pershendetje...");
            thread1.start();
```

```
            Runnable sms2 = new DisplayMessage("Pashim");
            Thread thread2 = new Thread(sms2);
            thread2.setPriority(Thread.MIN_PRIORITY);
            thread2.setDaemon(true);
            System.out.println("Nisje e thread pashim...");
            thread2.start();

            System.out.println("Nisje e thread 3...");
            Thread thread3 = new GjejNr(27);
            thread3.start();
            try {
               thread3.join();
            }catch(InterruptedException e) {
               System.out.println("Thread u nderpre.");
            }
            System.out.println("Nisje e thread4...");
            Thread thread4 = new GjejNr(4);

            thread4.start();
            System.out.println("main() po perfundon...");
         }
      }
```

Në console:

```
Nisje e thread pershendetje...
Nisje e thread pashim...
Pershendetje
Pershendetje
Pershendetje
Pershendetje
Pershendetje
Pershendetje
Pashim
Pashim
Pashim
Pashim
Pashim
.......
```