
Crystal
Talent Solutions

Leksion 7

Trashëgimia dhe polimorfizmi

1. Trashëgimia

Një klasë përfaqëson një grup objektesh që ndajnë të njëjtën strukturë dhe sjellje. Klasa përcakton strukturën e objekteve duke specifikuar variablat që përmbahen në çdo instancë të klasës, gjithashtu klasa përcakton sjelljen duke siguruar metodat e instancës që shprehin sjelljen e objekteve. Programimi i orientuar në objekte lejon që të trashëgoni (përftoni) klasa të reja nga klasat ekzistuese. Kjo quhet *trashëgimi*.

1.1 Superklasat dhe nënklasat

Trashëgimia ju mundëson të përcaktoni një klasë të përgjithshme dhe më pas ta trashëgoni atë në klasa më të specializuara. Klasat e specializuara trashëgojnë karakteristika dhe metoda nga klasa e përgjithshme. *Në Java një klasë C2 që trashëgon nga një klasë C1 quhet një nënklasë (subclass), kurse klasa C2 quhet një superklasë.* Një superklasë njihet si nj ë *klasë prind* ose një *klasë bazë* dhe një nënklasë njihet si nj ë *klasë fëmijë*, një *klasë e trashëguar ose një klasë e përftuar*. Një nënklasë është ë një klasë ekzistuese që mund të përshtatet me disa ndryshime ose shtime. Më konkretisht, një nënklasë trashëgon fusha të dhënash dhe metoda të akses ueshme nga superklasa e saj por gjithashtu mund të shtojë fusha të dhënash dhe metoda të reja. Sintaksa për krijimin e një nënklase nga një superklasë është:

public class <emri_nënklasës> extends <emri-superklasës> { . . . //ndryshimet dhe

shtimet }

Fjala extends i tregon kompilatorit që nënklasa po zgjeron superklasën, kështu ajo trashëgon metodat dhe fushat e të dhënave të aksesueshme nga superklasa. Me trashëgiminë, programuesit kursejnë kohë gjatë zhvillimit të programit duke ripërdorur software. Çdo nënklasë në Java mund të jetë një superklasë për klasa të tjera. Nga një nënklasë mund të përftohen më shumë se një nënklasë. Java, ndryshe nga C++ nuk e suporton trashëgiminë e shumëfishtë, ku një nënklasë mund të trashëgojë nga disa superklasa.

Shembull: Supozojmë një klasë ObjekteGjeometrike që mund të përdoret për të modeluar gjithë objektet gjeometrike. Kjo klasë përmban karakteristikat ngjyra dhe i_ngjyrosur dhe metodat e tyre përkatëse *get* dhe *set*. Supozojmë se kjo klasë ka dhe një karakteristikë dataKrijimit dhe metodat getDataKrijimit dhe toString(). Metoda toString kthen një prezantim në formë stringu për objektin. Java ofron klasën Date të disponueshme në paketën java.util, kjo klasë tregon datën dhe orën aktuale.

```
import java.util.Date;

public class ObjekteGjeometrike {
  private String ngjyra = "e_zeze";
  private boolean i_ngjyrosur;
  private Date dataKrijimit;

  public ObjekteGjeometrike(){
          dataKrijimit = new Date();
  }
  public ObjekteGjeometrike(String ngjyra, boolean i_ngjyrosur){ dataKrijimit = new Date();
            this.ngjyra = ngjyra;
```

```
            this.i_ngjyrosur = i_ngjyrosur;
  }
  //kthen ngjyren
  public String getNgjyra(){
        return ngjyra;
  }
  //cakto nje ngjyre te re
  public void setNgjyra(String ngjyra){
        this.ngjyra = ngjyra;
  }
  public boolean eshteNgjyrosur(){
        return i_ngjyrosur; //mqs eshte boolean metoda nga get quhet eshte
  }
  public void setNgjyrosur(boolean i_ngjyrosur){ this.i_ngjyrosur =
        i_ngjyrosur;
  }
  public Date getDataKrijimit(){
        return dataKrijimit;
  }
  public String toString(){
        return "ngjyra " + ngjyra + "; A eshte ngjyrosur? " + i_ngjyrosur + ".\nData kur u krijua: " +
                    dataKrijimit;
  }
}
```

Meqenëse një rreth është një tip i veçantë objektesh gjeometrike, ai ndan karakteristika dhe metoda të njëjta me objektet e tjera gjeometrike. Kështu ka kuptim të përcaktojmë klasën Rreth që trashëgon nga klasa ObjekteGjeometrike. Klasa Rreth trashëgon të gjitha fushat e të dhënave dhe metodat e aksesueshme nga klasa ObjekteGjeometrike. Për më tepër ajo ka një fushë të re të dhëne rrezja, dhe metodat *get* dhe *set* q ë lidhen me të. Klasa Rreth përmban dhe metodat getSiperfaqja(), getDiametri() dhe getPerimeter()që kthejnë sipërfaqen, diametrin dhe perimetrin e rrethit.

```
public class Rreth extends ObjekteGjeometrike{ private double
        rrezja;
        public Rreth(){
        }
        public Rreth(double rrezja){
              this.rrezja = rrezja;
        }
        public Rreth(double rrezja, String ngjyra, boolean i_ngjyrosur){ this.rrezja = rrezja;
              setNgjyra(ngjyra);
              setNgjyrosur(i_ngjyrosur);
        }
        public double getRrezja(){
              return rrezja;
        }
        public void setRrezja(double rrezja){
              this.rrezja = rrezja;
        }
        public double getSiperfaqja(){
              return rrezja * rrezja * Math.PI;
        }
        public double getDiametri(){
              return 2*rrezja;
        }
        public double getPerimeter(){
              return 2 * Math.PI * rrezja;
        }
```

```
	public void printoRreth(){
		System.out.println("Rrethi eshte krijuar ne " + getDataKrijimit() + "dhe rrezja e tij eshte " +
				rrezja);
	}
}
```

Klasa Rreth trashëgon nga klasa ObjekteGjeometrike metodat getNgjyra , setNgjyra, eshteNgjyrosur, setNgjyrosur dhe toString. Nëse do përpiqeshit që në konstruktor ti përdornit direkt fushat e të dhënave ngjyra dhe i_ngjyrosur si më poshtë:

```
public Rreth(double rrezja, String ngjyra, boolean i_ngjyrosur){ this.rrezja = rrezja;
		this.ngjyra = ngjyra; //nuk lejohet
		this.i_ngjyrosur = i_ngjyrosur; //nuk lejohet
	}
```

do të ishte gabim pasi fushat e të dhënave ngjyra dhe i_ngjyrosur janë përcaktuar si private në klasën ObjekteGjeometrike dhe nuk mund të aksesohen jashtë asaj klase. E vetmja mënyrë për të lexuar dhe modifikuar ngjyra dhe i_ngjyrosur është nëpërmjet metodave të tyre *get* dhe *set*.

Në m ënyrë të ngjashme klasa Drejtkendesh deklarohet si nj ë nënklasë e klasës ObjekteGjeometrike. Klasa Drejtkendesh trashëgon të gjitha fushat e të dhënave dhe metodat e aksesueshme nga klasa ObjekteGjeometrike. Përveç tyre, ajo ka fushat e të dhënave gjeresia dhe gjatesia dhe metodat *get* dhe *set* që lidhen me to. Klasa Drejtkendesh përmban gjithashtu metodën getSiperfaqja() dhe getPerimeter() që kthejnë sipërfaqen dhe perimetrin e drejtkëndëshit.

```
public class Drejtkendesh extends ObjekteGjeometrike{ private double gjeresi;
	private double gjatesi;

	public Drejtkendesh() {
	}
	public Drejtkendesh(double gjeresi, double gjatesi) { this.gjeresi = gjeresi;
		this.gjatesi = gjatesi;
	}
	public Drejtkendesh(double gjeresi, double gjatesi, String ngjyra, boolean i_ngjyrosur) {
		this.gjeresi = gjeresi;
		this.gjatesi = gjatesi;
		setNgjyra(ngjyra);
		setNgjyrosur(i_ngjyrosur);
	}
	public double getGjeresi(){
		return gjeresi;
	}
	public void setGjeresi(double gjeresi){
		this.gjeresi = gjeresi;
	}
	public double getGjatesi(){
		return gjatesi;
	}
	public void setGjatesi(double gjatesi){
		this.gjatesi = gjatesi;
	}
	public double getSiperfaqja(){
		return gjeresi * gjatesi;
	}
	public double getPerimeter(){
		return 2 * (gjeresi + gjatesi);
	}
```

```
}
```

Klasa Drejtkendesh trashëgon nga klasa ObjekteGjeometrike metodat getNgjyra, setNgjyra, eshteNgjyrosur, setNgjyrosur dhe toString. Klasa TestRreth krijon dy objekte rreth dhe thërret metoda mbi këto objekte.

```
public class TestRreth {

	public static void main(String[] args) {
		Rreth rr1 = new Rreth(1);
		System.out.println("Ngjyra e rrethit eshte " + rr1.toString());
		System.out.println("Rrezja eshte " + rr1.getRrezja());
		System.out.println("Siperfaqja eshte " + rr1.getSiperfaqja());
		System.out.println("Diametri eshte " + rr1.getDiametri());

		Rreth rr2 = new Rreth(2, "blu", true);
		System.out.println("\nNgjyra e rrethit eshte " + rr2.toString());
		System.out.println("Rrezja eshte " + rr2.getRrezja());
		System.out.println("Siperfaqja eshte " + rr2.getSiperfaqja());
		System.out.println("Diametri eshte " + rr2.getDiametri());
	}
}
```

Kjo klasë afishon:

```
Ngjyra e rrethit eshte ngjyra e_zeze; A eshte ngjyrosur? false.
Data kur u krijua: Wed Nov 25 16:57:02 CET 2015
Rrezja eshte 1.0
Siperfaqja eshte 3.141592653589793
Diametri eshte 2.0

Ngjyra e rrethit eshte ngjyra blu; A eshte ngjyrosur? true.
Data kur u krijua: Wed Nov 25 16:57:02 CET 2015
Rrezja eshte 2.0
Siperfaqja eshte 12.566370614359172
Diametri eshte 4.0
```

Klasa TestDrejtkendesh krijon një objekt drejtkëndësh dhe thërret metoda mbi këtë objekt.

```
public class TestDrejtkendesh {

	public static void main(String[] args) {
		Drejtkendesh dr1 = new Drejtkendesh (2,4);
		System.out.println("Ngjyra e drejtkendeshit eshte " +
		dr1.toString());
		System.out.println("Siperfaqja eshte " + dr1.getSiperfaqja()); System.out.println("Perimetri eshte
		" + dr1.getPerimeter());
	}
}
```

Kjo klasë afishon:

```
Ngjyra e drejtkendeshit eshte ngjyra e_zeze; A eshte ngjyrosur? false. Data kur u krijua: Wed Nov 25
17:09:18 CET 2015
Siperfaqja eshte 8.0
Perimetri eshte 12.0
```

Ju mund të përcaktoni një klasë të vetme TestRrethDrejtkendesh ku krijohen objekte rreth dhe drejtkëndësh

në vend që të përcaktoni dy klasat TestRreth dhe TestDrejtkëndësh më vete.

```
public class TestRrethDrejtkendesh {

	public static void main(String[] args) {
		Rreth rr1 = new Rreth(1);
		System.out.println("Ngjyra e rrethit eshte " + rr1.toString());
		System.out.println("Rrezja eshte " + rr1.getRrezja());
		System.out.println("Siperfaqja eshte " + rr1.getSiperfaqja());
		System.out.println("Diametri eshte " + rr1.getDiametri());

		Rreth rr2 = new Rreth(2, "blu", true);
		System.out.println("\nNgjyra e rrethit eshte " + rr2.toString());
		System.out.println("Rrezja eshte " + rr2.getRrezja());
		System.out.println("Siperfaqja eshte " + rr2.getSiperfaqja());
		System.out.println("Diametri eshte " + rr2.getDiametri());

		Drejtkendesh dr1 = new Drejtkendesh (2,4);
		System.out.println("\nNgjyra e drejtkendeshit eshte " + dr1.toString());
		 System.out.println("Siperfaqja eshte " + dr1.getSiperfaqja()); System.out.println("Perimetri eshte
		" + dr1.getPerimeter());
	}
}
```

Ky program afishon:

```
Ngjyra e rrethit eshte ngjyra e_zeze; A eshte ngjyrosur? false.
Data kur u krijua: Wed Nov 25 17:16:06 CET 2015
Rrezja eshte 1.0
Siperfaqja eshte 3.141592653589793
Diametri eshte 2.0

Ngjyra e rrethit eshte ngjyra blu; A eshte ngjyrosur? true.
Data kur u krijua: Wed Nov 25 17:16:06 CET 2015
Rrezja eshte 2.0
Siperfaqja eshte 12.566370614359172
Diametri eshte 4.0

Ngjyra e drejtkendeshit eshte ngjyra e_zeze; A eshte ngjyrosur? false.
Data kur u krijua: Wed Nov 25 17:16:06 CET 2015
Siperfaqja eshte 8.0
Perimetri eshte 12.0
```

1.2 Fjala çelës *super*

Fjala çelës super i referohet superklasës të klasës në të cilën shfaqet fjala super. Ajo mund të përdoret:

- për të thirrur një konstruktor të një superklase.
- për të thirrur një metodë të një superklase.

1.2.1 Thirrja e një konstruktori të superklasës

Një konstruktor përdoret për të ndërtuar një instancë të një klase. Ndryshe nga fushat e të dhënave dhe metodat, konstruktorët e një superklase nuk trashëgohen në nënklasë. Ata mund të thirren nga konstruktori i nënklasës duke përdorur fjalën çelës super.

Sintaksa për të thirrur konstruktorin e një superklase është:

```
super();
```

ose

super(parametrat);

Këto statement-e duhet të shfaqen në rreshtin e parë të konstruktorit të nënklasës, për të thirrur një kontruktor të superklasës. Për shembull, konstruktori në klasën Rreth:

```
public Rreth(double rrezja, String ngjyra, boolean i_ngjyrosur){ this.rrezja = rrezja;
                    setNgjyra(ngjyra);
                    setNgjyrosur(i_ngjyrosur);
}
```

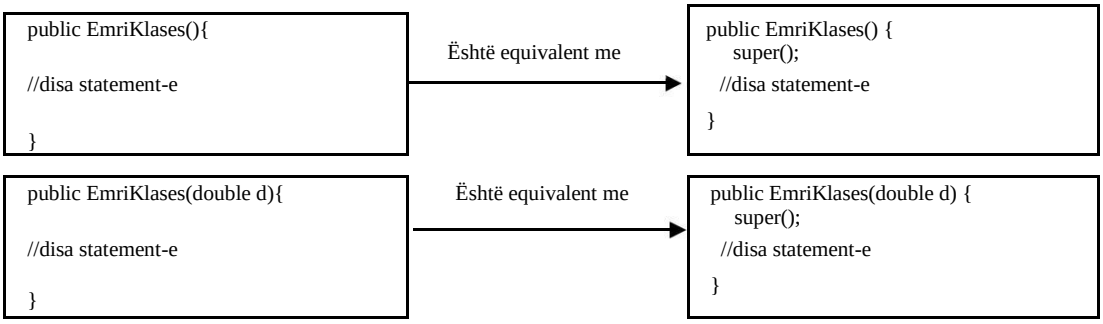
Mund të zëvendësohet nga kodi i mëposhtëm:

```
public Rreth(double rrezja, String ngjyra, boolean i_ngjyrosur){ super(ngjyra, i_ngjyrosur);
                    this.rrezja = rrezja;
}
```

Ku super(ngjyra, i_ngjyrosur); i referohet konstruktorit të klasës ObjekteGjeometrike.

1.2.2 Vargëzimi i konstruktorëve

Një konstruktor mund të thërrasë një konstruktor të mbingarkuar ose konstruktorin e superklasës të tij. Konstruktorët e mbingarkuar janë konstruktorë që kanë të njëjtin emër por listë të ndryshme parametrash. Nëse asnjë konstruktor nuk është thirrur në mënyrë eksplicite, kompilatori automatikisht vendos super() si statement-in e parë në konstruktor. Për shembull:



Në çdo rast, ndërtimi i një instance nga një klasë thërret konstruktorët e gjithë superklasave përgjatë vargut të hierarkisë. Kur ndërtohet një objekt i një nënklase, konstruktori i nënklasës fillimisht thërret konstruktorin e superklasës të tij përpara se të kryejë detyrat e tij. Nëse superklasa derivohet nga një klasë tjetër, konstruktori i superklasës thërret konstruktorin e klasës prind të saj përpara se të kryejë detyrat e tij. Ky proces vazhdon derisa të thirret konstruktori i fundit përgjatë hierarkisë të trashëgimisë. Ky është *vargëzimi i konstruktorëve*.

Shembull:

```
public class Pedagog extends Punonjesi { public static void
    main(String[] args) {
        new Pedagog();
    }
    public Pedagog() {
        System.out.println("(4)Kryen detyrat e Pedagog ");
    }
}
class Punonjesi extends Personi {
    public Punonjesi () {
```

```
        this("(2)therret konstruktorin e mbingarkuar te Punonjesi"); System.out.println("(3)Kryen
        detyrat e Punonjesi");
    }
    public Punonjesi (String s){
        System.out.println(s);
    }
}

class Personi {
    public Personi() {
        System.out.println("(1) Kryen detyrat e Personi");
    }
}
```

Afishon:

(1) Kryen detyrat e Personi
(2) therret konstruktorin e mbingarkuar te Punonjesi
(3) Kryen detyrat e Punonjesi
(4) Kryen detyrat e Pedagog

new Pedagog() thërret konstruktorin pa parametra të Pedagog. Meqenëse Pedagog është ë një nënklasë e Punonjesi, konstruktori pa parametra i Punonjesi është thirrur para se të ekzekutohen statement-et në konstruktorin e Pedagog. Konstruktori pa parametra i Punonjesi thërret konstruktorin e dytë të Punonjesi. Meqenëse Punonjesi është një nënklasë e Personi, kontruktori pa parametra i Personi është thirrur përpara se të ekzekutohen statement-et në konstruktorin e dytë të Punonjesi.

1.2.3 Thirrja e metodave të superklasës

Fjala çelës super mundet gjithashtu t ë përdoret për tiu referuar një metode dhe jo konstruktorit në superklasë. Sintaksa e referimit të një metode të superklasës është:

super.metoda(parametrat);

Ju mund ta rishkruani metodën printoRreth() në klasën Rreth() si më poshtë:

```
public void printoRreth(){
        System.out.println("Rrethi eshte krijuar ne " + super.getDataKrijimit() + "dhe rrezja e tij eshte " + rrezja);
}
```

Nuk është e nevojshme të vendoset super përpara getDataKrijimit() në këtë rast, sepse getDataKrijimit është një metodë në klasën ObjekteGjeometrike dhe është trashëguar nga klasa Rreth. Ka raste siç do shohim në vazhdimësi që fjala super është e nevojshme të vendoset.

1.3 Mbishkrimi i metodave

Një nënklasë trashëgon metoda nga një superklasë. Ndonjëherë është e nevojshme për nënklasën që të modifikojë implementimin e një metode të përcaktuar në superklasë. Kjo njihet si *mbishkrimi i metodave*. Metoda toString në klasën ObjekteGjeometrike kthen prezantimin në formë stringu për një objekt gjeometrik. Kjo metodë mund të mbishkruhet për të kthyer prezantimin në formë stringu për një rreth. Për ta mbishkruar atë, shtoni metodën e mëposhtme në klasën Rreth që kemi parë më sipër:

```
public class Rreth extends ObjekteGjeometrike { //metodat e tjera

jane hequr
```

```
//Mbishkruhet metoda toString e percaktuar ne klasen ObjekteGjeometrike public String toString {
      return super.toString()+ "\nrrezja eshte " + rrezja;
  }
}
```

Metoda toString është përcaktuar në klasën ObjekteGjeometrike dhe është modifikuar në klasën Rreth. Të dyja metodat mund të përdoren në klasën Rreth. Për të thirrur metodën toString të përcaktuar n ë klasën ObjekteGjeometrike nga klasa Rreth, përdoret super.toString(). A mundet një nënklasë e Rreth të aksesojë metodën toString të përcaktuar në klasën ObjekteGjeometrike duke përdorur sintaksën super.super.toString()? Jo. Ky është një gabim sintaksor.

Një metodë instance mund të mbishkruhet vetëm nëse ajo është e aksesueshme. Kështu një metod ë private nuk mund të mbishkruhet, sepse ajo nuk është e aksesueshme jashtë klasës të saj. Nëse një metodë e përcaktuar në një nënklasë është private në superklasën e saj, të dyja metodat nuk kanë lidhje me njëra-tjetrën.

Ashtu si një metodë instance, një metodë statike mund të trashëgohet. Gjithsesi, *një metodë statike nuk mund të mbishkruhet. Nëse një metodë statike e përcaktuar në superklasë është ripërcaktuar në një nënklasë, metoda e përcaktuar në superklasë është e fshehur (hidden).* Metodat e fshehura statike mund të thirren duke përdorur sintaksën EmriSuperKlasës.emriMetodësStatike.

1.4 Mbishkrimi kundrejt mbingarkimit

Mbingarkim do të thotë të përcaktosh disa metoda me emra të njëjtë por me nënshkrime të ndryshme. Nënshkrimi i një metode përfshin emrin e metodës, numrin, tipin dhe rendin e parametrave të saj. *Mbishkrim* do të thotë të ofrosh një implementim të ri për një metodë në nënklasë. Metoda është e përcaktuar tashmë në superklasë. Për të mbishkruar një metodë, vetë metoda duhet të përcaktohet në nënklasë duke përdorur të njëjtin nënshkrim dhe të njëjtin tip kthimi. Le të shohim një shembull për të kuptuar më mirë diferencën midis mbishkrimit dhe mbingarkimit.

```
public class Test {
    public static void main(String[] args) { A a = new A();
        a.p(10);
        a.p(10.0);
    }
}

class B {
    public static void p(double i) {
        System.out.println(i * 2);
    }
}

class A extends B {
    //  kjo metode mbishkruan metoden ne B public void
    p(double i) {
        System.out.println(i);
    }
}
```

(a)

```
public class Test {
    public static void main(String[] args) { A a = new A();
        a.p(10);
        a.p(10.0);
    }
}

class B {
    public void p(double i) {
        System.out.println(i * 2);
    }
}

class A extends B {
    //  kjo metode mbingarkon metoden e B public
    void p(int i) {
        System.out.println(i);
    }
}
```

(b)

Në (a), metoda p(double të i) në klasën A mbishkruan të njëjtën metodë të përcaktuar në klasën B. Kur ju ekzekutoni klasën Test në (a), të dyja a.p(10) dhe a.p(10.0) thërrasin metodën p(double i) të të shfaqur përcaktuar në klasën A për 10.0. Në (b), klasa A ka dy metoda të mbingarkuara, p(double i)

dhe p(int i). Metoda p(double i) është trashë guar nga klasa B. Kur ekzekutohet klasa Test në (b), a.p(10) thërret metodën p(int i) të përcaktuar në klasën A për të shfaqur 10 dhe a.p(10.0) thërret metodën p(double i) të përcaktuar në klasën B për të shfaqur 20.0.

1.5 Klasa Object dhe metoda toString() e saj

Çdo klasë në Java trashëgohet nga klasa java.lang.Object. Nëse kur përcaktohet një klasë nuk specifikohet trashëgimia, superklasa e klasës është klasa Object, by default. Për shembull, përcaktimet e mëposhtme të klasave janë të njëjta:

```
public class KlasaA {
...
}
```

Ekuivalente me

```
public class KlasaA extends Object {
...
}
```

Klasat String dhe ObjekteGjeometrike janë nënklasa të Object, siç janë të gjitha klasat e tjera që kemi parë deri tani.

Nënshkrimi i metodës toString() është:

```
public String toString()
```

Thirrja e metodës toString() mbi një objekt kthen një string që përshkruan objektin. By default, ajo kthen një string që përbëhet nga emri i klasës që ka si instancë objektin, një shenjë @ dhe adresa e objektit në kujtesë në hekzadecimal. Për shembull, shikoni kodin për një klasë KlasaA:

```
KlasaA objekt = new KlasaA();
System.out.println(objekt.toString());
```

Kodi shfaq diçka si A@14057e9. Ky mesazh nuk ju ndihmon apo informon shumë. Zakonisht ju duhet të mbishkruani metodë n toString në mënyrë që ajo të kthejë një string përshkrues për objektin. Metoda toString në klasën Object është mbishkruar në klasën ObjekteGjeometrike që kemi parë më sipër:

```
public String toString(){
        return "ngjyra " + ngjyra + "; A eshte ngjyrosur? " + i_ngjyrosur + ".\nData kur u krijua: " +
                        dataKrijimit;
  }
```

2. Polimorfizmi

Një klasë përcakton një tip. Një tip i përcaktuar nga një nënklasë quhet një *nëntip* dhe një tip i përcaktuar nga superklasa e tij quhet një *supertip*. Kështu, ju mund të thoni se Rreth është një nëntip i ObjekteGjeometrike dhe klasa ObjekteGjeometrike është një supertip për Rreth. Trashëgimia bën të mundur që një nënklasë të trashëgojë tipare nga superklasa e saj dhe të shtojë tipare të reja. Çdo instancë e një nënklase është gjithashtu instancë e superklasës të saj, por jo e kundërta. Për shembull, çdo rreth është një objekt gjeometrik, por jo çdo objekt gjeometrik është një rreth. Prandaj, ju mundeni gjithmonë të kaloni një instancë të një nënklase tek një parametër e tipit të superklasës të tij. Shikoni kodin
e mëposhtëm:

```
public class Polimorfizmi {
        public static void main(String[] args) { shfaqObjekt(new Rreth(1, "e_kuqe", false));
                shfaqObjekt(new Drejtkendesh(1, 1, "e_zeze", true));
        }
    public static void shfaqObjekt(ObjekteGjeometrike objekt){ System.out.println("Krijuar me " +
        objekt.getDataKrijimit()
                        + ". Ngjyra eshte " + objekt.getNgjyra());
    }
}
```

Kjo klasë afishon:

```
Krijuar me Tue Dec 08 21:25:51 CET 2015. Ngjyra eshte e_kuqe
Krijuar me Tue Dec 08 21:25:52 CET 2015. Ngjyra eshte e_zeze
```

Metoda shfaqObjekt merr një parametër të tipit ObjekteGjeometrike. Ju mund të thërrisni metodën shfaqObjekt duke i kaluar si argument ndonjë instancë të ObjekteGjeometrike (p.sh new Rreth(1, "e_kuqe", false) dhe new Drejtkendesh(1, 1, "e_zeze", true)). Një objekt i një nënklase mund të përdoret kudo që është përdorur objekti i superklasës të tij. Kjo njihet si *polimorfizëm* (vjen nga Greqishtja dhe do të thotë "shumë forma"). Me terma të thjeshtë, polimorfizëm do të thotë që një variabël
i një supertipi mund ti referohet objektit të një nëntipi.

3. Lidhja dinamike

Një metodë mund të përcaktohet në një superklasë dhe të mbishkruhet në nënklasën e saj. Për shembull, metoda toString() është përcaktuar në klasën Object dhe është mbishkruar në klasën ObjekteGjeometrike. Shikoni kodin e mëposhtëm:

```
Object o = new ObjekteGjeometrike();
System.out.println(o.toString());
```

Cila metodë toString() është thirrur nga o? Tipi i një variabli quhet *tip i deklaruar*. Këtu tipi i deklaruar i o është Object. Një variabël i një tipi referencial mund të mbajë një vlerë null ose një referencë të një instance të tipit të deklaruar. Instanca mund të krijohet duke përdorur konstruktorin e tipit të deklaruar ose të nëntipit të tij. *Tipi aktual* i variablit është klasa aktuale për objektin e referuar nga variabli. Këtu tipi aktual i o është ObjekteGjeometrike, meqenëse o i referohet një objekti të krijuar duke përdorur new ObjekteGjeometrike(). Se cila metodë toString() është thirrur nga o është vendosur nga tipi aktual i o. Kjo njihet si *lidhje dinamike*.

Lidhja dinamike funskionon si m ë poshtë: supozoni një objekt o që është një instancë e klasës C_1, C_2,…, C_{n-1}, dhe C_n, ku C_1 është një nënklasë e C_2, C_2 është një nënklasë e C_3, . . . , C_{n-1} është një nënklasë e C_n. Kështu, C_n është klasa më e përgjithshme dhe C_1 është klasa më specifike. Në Java, C_n është klasa Object. Nëse o thërret një metodë p, JVM kërkon implementimin e metodës p në C_1, C_2, . . . , C_{n-1} dhe C_n, në këtë renditje derisa ai të gjendet. Sapo gjendet një implementim, kërkimi ndalon dhe thirret implementimi i parë.

Kodi i mëposhtëm demostron lidhjen dinamike:

```
public class LidhjeDinamike {

        public static void main(String[] args){
            m(new StudentDiplomuar());
            m(new Student());
            m(new Person());
            m(new Object());
        }
        public static void m(Object x) {
            System.out.println(x.toString());
        }
}

class StudentDiplomuar extends Student {
}

class Student extends Person {
public String toString() {
            return "Student";
    }
```

```
}

class Person extends Object {
	public String toString() {
		return "Person";
	}
}
```

Afishon:

```
Student
Student
Person
java.lang.Object@2a139a55
```

Metoda m merr një parametër të tipit Object. Ju mund ta thërrisni m me çdo objekt (p.sh., new StudentDiplomuar(), new Student(), new Person() dhe new Object(). Kur ekzekutohet metoda m(Object x) thirret argumenti x i metodës toString(). x mund të jetë një instancë e StudentDiplomuar, Student, Person ose Object. Klasat StudentDiplomuar, Student, Person ose Object kanë implementimet e tyre të metodës toString. Thirrja e m(new StudentDiplomuar()) shkakton thirrjen e metodës toString të përcaktuar në klasën Student. Thirrja e m(new Student()) shkakton thirrjen e metodës toString të përcaktuar në klasën Student. Thirrja e m(new Person()) shkakton thirrjen e metodës toString të përcaktuar në klasën Person. Thirrja e m(new Object()) shkakton thirrjen e metodës toString të përcaktuar në klasën Object.

4. Konvertimi i tipit të objekteve dhe Operatori instanceOf

4.1 Casting i objekteve (ndërrimi i tipit të objekteve)

Ju tashmë e keni përdorur operatorin e ndryshimit të tipit për të konvertuar variabla të një tipi primitiv në një tjetër tip primitiv. *Casting* gjithashtu mund të përdoret për të konvertuar një objekt të një tipi klasë një një tip tjetër klase brenda hierarkisë së trashëgimisë. Në seksionin e mëparshëm, statementi

```
m(new Student());
```

e cakton objektin new Student() tek një parameter të tipit Object. Ky statement është ekuivalent me:

```
Object o = new Student(); // konvertimi implicit i tipit(implicit casting)
m(o);
```

Statement-i Object o = new Student(), i njohur si konvertim implicit i tipit, është i lejuar sepse një instancë e Student është automatikisht një instancë e Object.

Supozoni se ju doni të caktoni referencën e objektit o tek një variabël i tipit Student duke përdorur statement-in e mëposhtëm:

```
Student b = o;
```

Në këtë rast do të ndodhte një gabim kompilimi. Pse statement-i Object o = new Student() funksionon dhe Student b = o nuk funksionon? Arsyeja është sepse një objekt Student është gjithmonë një instancë e Object, por një objekt Object nuk është detyrimisht një instancë e Student. Edhe pse ju mund ta shihni që o është me të vërtetë një objekt Student, kompilatori nuk është mjaftueshëm I zgjuar sa ta kuptojë atë. Për ti treguar kompilatorit që o është një objekt Student, përdoret *konvertimi eksplicit i tipit* (explicit casting). Sintaksa e ndërrimit ekplicit të tipit të objekteve është e ngjashme me atë të ndërrimit implicit vetëm se në këtë rast tipi objekt i synuar futet midis një çifti kllapash dhe vendoset para objektit të cilit do i konvertohet tipi, si më poshtë:

```
Student b = (Student) o; //konvertimi eksplicit i tipit
```

Është gjithmonë e mundur të konvertohet një instancë e një nënklase në një variabël të një superklase (e njohur si *konvertimi i sipërm*: upcasting), sepse një instancë e një nënklase është gjithmonë një instancë e

një superklase. Kur bëhet konvertimi nga një instancë e një superklase në një variabël të nënklasës të saj (i njohur si *konvertimi i poshtëm*: downcasting), konvertimi eksplicit duhet të përdoret për të konfirmuar qëllimin tuaj tek kompilatori me shënimin e ndërrimit (EmriNënKlasës). Që konvertimi të jetë i suksesshëm, ju duhet të siguroheni që objekti të cilit do ti ndërrohet tipi është një instancë e nënklasës. Nëse objekti i superklasës nuk është një instancë e nënklasës do të ndodhë një *ClassCastException*. Për shembull, nëse një objekt nuk është një instancë e Student, ai nuk mund të konvertohet në një variabël të Student.

4.2 Operatori *instanceOf*

Operatori instanceOf përdoret për të kontrolluar nëse një objekt është nj ë instancë e një klase. Përdorimi i tij është i rëndësishëm përpara se të bëhet konvertimi eksplicit i tipit. Shikoni kodin e mëposhtëm:

```
Object njëObjekt = new Rrethi();
... // disa rreshta kod
/** realizon konvertimin nëse njëObjekt është një instancë e Rrethi */ if (njëObjekt instanceof Rrethi) {
    System.out.println("Diametri i Rrethit është " +
        ((Rrethi)njëObjekt).getDiametër());
    ...
}
```

Variabli njëObjekt është deklaruar i tipit Object. Përdorimi i njëObjekt.getDiametër() do të shkaktonte një gabim kompilimi, sepse klasa Object nuk e ka metodën getDiametër. Është i nevojshëm konvertimi i njëObjekt në tipin Rrethi për ti treguar kompilatorit që njëObjekt është gjithashtu një instancë e Rrethi.

Për të ndihmuar në të kuptuarit më mirë të konvertimit, ju mund të merrni në konsideratë analogjinë e frutit, mollës dhe portokallit me klasën Fruti si superklasë për nënklasat Mollë dhe Portokall. Një mollë është një frut, prandaj mund të caktohet një instancë e Mollë tek një variabël i tipit Fruti.
Megjithatë një frut nuk është domosdoshmërisht një mollë, prandaj përdoret konvertimi eksplicit për të caktuar një instancë të Fruti tek një variabël i tipit Mollë.

5. Metoda equals()

Metoda equals është një metodë e përcaktuar në klasën Object. Nënshkrimi i saj është:

```
public boolean equals(Object o)
```

Kjo metodë teston nëse dy objekte janë të njëjta. Sintaksa për thirrjen e saj janë:

```
objekt1.equals(objekt2);
```

Implementimi default i metodës equals në klasën Object është:

```
public boolean equals(Object obj){
    return(this == obj);
}
```

Ky implementim kontrollon nëse dy variabla referencial shënjojnë tek i njëjti objekt duke përdorur operatorin ==. Ju duhet të mbishkruani metodën equals në klasën tuaj për të testuar nëse dy objekte të ndryshme kanë t ë njëjtën përmbajtje. Metoda equals përdoret dhe në rastin kur duam të krahasojmë dy stringje. Metoda equals në klasën String është trashëguar nga klasa Object dhe është mbishkruar në klasën String për të testuar nëse dy stringje janë identike në përmbajtje. Ju mund të mbishkruani metodën equals në klasën Rrethi për të krahasuar nëse dy rrathë janë të njëjtë bazuar në rrezet e tyre, si më poshtë:

```
public boolean equals(Object o) {
if (o instanceof Rrethi) {
return rrezja == ((Rrethi)o).rrezja;
}
else
return false;
}
```

6. Fushat e të dhënave dhe metodat protected dhe modifikuesi final

6.1 Fjala çelës protected

Deri tani kemi përdorur fjalët çelës private dhe public për të specifikuar nëse fushat e të dhënave dhe metodat mund të aksesohen nga jashtë klasës. Anëtarët privatë mund të aksesohen vetëm brenda klasës dhe anëtar ët publikë mund të aksesohen nga çdo klasë tjetër. Fjala çelës protected përdoret për të bërë të mundur që nënklasat t ë aksesojnë fushat e të dhënave dhe metodat e përcaktuara në superklasën e tyre. Pra një fushë të dhënash ose metodë në një superklasë mund të aksesohet në nënklasat e saj.

Modifikuesit private, protected dhe public njihen si *modifikuesit e dukshmërisë* (vizibilitetit) ose *modifikuesit e aksesueshmërisë* sepse ata specifikojnë sesi aksesohen klasa dhe anëtarët e klasës. Dukshmëria e këtyre modifikuesve rritet në rendin

Dukshmëria rritet

private, default (nëse nuk është përdorur modifikues), protected, public

6.2 Modifikuesi final

Modifikuesi final përdoret për të ndaluar trashëgiminë e klasave. Pra, kur një klasë është final, ajo nuk mund të jetë një klasë prind. Klasa Math dhe String janë klasa finale. Për shembull, klasa e mëposhtme është finale dhe nuk mund të zgjerohet (derivohet):

```
public final class C {
...
}
```

Një metodë finale nuk mund të mbishkruhet nga nënklasat e klasës ku ajo është përcaktuar. Për shembull, metoda e mëposhtme është finale dhe nuk mund të mbishkruhet:

```
public class Test {
...//fushat e të dhënave, konstruktorë, metoda mund të shkruhen këtu public final void metoda1() {
//     Bëj diçka
}
}
```

Modifikuesit mund të përdoren mbi klasat dhe anëtarët e klasave (fushat e të dhënave dhe metodat). Modifikuesi final mund të përdoret dhe mbi variablat lokalë në një metodë. Një variabël lokal final është një konstante brenda një metode.

7. Ushtrime

Ushtrimi 1

Le të shohim një program në Java që ilustron trashëgiminë duke përdorur klasën Person. Çfarë printon programi:

```
class Person{
        String emri;
```

```
        String mbiemri;

        Person(String em, String mb){
                        emri = em;
                        mbiemri = mb;
        }

        void shfaq(){
                    System.out.println("Emri : " + emri);
                    System.out.println("Mbiemri : " + mbiemri);
        }
}

class Nxenes extends Person{
        int id;
        String klasa;

        Nxenes(String em, String mb, int nId, String kl){ super(em,mb);
                id = nId;
                klasa = kl;
        }
        void shfaq(){
                    super.shfaq();

                    System.out.println("ID : " + id);
                    System.out.println("Klasa : " + klasa);
        }
}

class Mesues extends Person{
          String lendaKryesore;
          int paga;
          String tipi; // mesuese fillore ose nentevjeçareje

        Mesues(String em, String mb, String lenda, int pg, String tp)
        {
                super(em,mb);
                lendaKryesore = lenda;
                paga = pg;
                tipi = tp;
          }
        void shfaq(){
                    super.shfaq();

                    System.out.println("Lenda kryesore : " + lendaKryesore);
                    System.out.println("Paga : " + paga);
                    System.out.println("Tipi : " + tipi);
        }
}

class TrashegimiaDemo{
            public    static    void    main(String    args[]){    Person    p    =    new
                        Person("Reni","Bushi");
                        Nxenes n = new Nxenes("Jani","Smith",1,"1 - B");
                        Mesues m = new Mesues("Edi","Papa","Anglisht",45000,"Mesues
                        filloreje");
                        System.out.println("Person");
                        p.shfaq();
                        System.out.println("");
                        System.out.println("Nxenes");
                        n.shfaq();
```

```
                        System.out.println("");
                        System.out.println("Mesues");
                        m.shfaq();
                }
}
```

Pas ekzekutimit të klasës TrashegimiaDemo do printohet:

```
Person
Emri : Reni
Mbiemri : Bushi

Nxenes
Emri : Jani
Mbiemri : Smith
ID : 1
Klasa : 1 - B

Mesues
Emri : Edi
Mbiemri : Papa
Lenda kryesore : Anglisht
Paga : 45000
Tipi : Mesues filloreje
```