Polimorfizam

Sadržaj

[Uvod 2](#_Toc21837210)

[Dinamičko Vezivanje metoda 4](#_Toc21837211)

[Konverzija objekata (Casting Objects) 8](#_Toc21837212)

[Operator instanceof 9](#_Toc21837213)

[Metoda equals klase Object 11](#_Toc21837214)

[ArrayList klasa 11](#_Toc21837215)

[Protected članovi 11](#_Toc21837216)

[Zabrana naslijeđivanja i override-anja 12](#_Toc21837217)

# Uvod

Tri stuba objektno orijentisanog programiranja su:

1. *Enkapsulacija*
2. *Naslijeđivanje*
3. *Polimorfizam*

*Polimorfizam* znači da varijabla supertipa može ukazivati na objekat subtipa.

**PRIMJER:**

Animal cuko = new Dog();

Klasa definiše tip.  
*Subtip* (podtip) je tip definisan od strane klase!  
*Supertip* je tip definisan od strane superklase.   
Možemo reći da je Dog subtip Animal-a, te da je Animal supertip Dog-a.

**PRIMJER:**

Animal cuko = new Dog();  
SUPERTIP SUBTIP

Naslijeđivanje omogućava subklasi da naslijedi članove (podatke i metode) od superklase, te da subklasa doda još novih članova. Zato se kaže da extenda (proširuje). Što znači da subklasa ima više članova nego superklasa. Svaka instanca subklase je također instanca svoje superklase, ali ne možemo reći i obrnuto. Na primjer: svaki je cuko životinja, ali nije svaka životinja cuko.

Stoga, uvijek možemo proslijediti instancu subklase parametru tipa superklase.

**PRIMJER:**

public class Demo {

public static void displayObject(GeometricObject object){

System.out.println(''Created on: '' + object.getDate());

}

public static void main (String[] args) {

displayObject( new Circle(1, ''red'' , false));

displayObject (new Rectangle(1,1, ''black'', true));

}

**OUTPUT**

Created on: Mon Mar 09 19:25:20 EDT 2011.   
Created on: Mon Mar 09 19:25:20 EDT 2011.

Metoda displayObject kao parametar prima GeometricObject tip.   
Možemo pozvati displayObject proslijeđujući bilo koju instancu od GeometricObject-a. Na primjer:

new Circle(1, ''red'' , false), new Rectangle(1,1, ''black'', true).

Objekat subklase se može koristiti gdje god se koristi objekat superklase. Ovo je poznato kao polimorfizam (grč. „više oblika“).

U pojednostavljenom smislu, polimorfizam znači da varijabla supertipa može ukazivati na objekat subtipa.

# Dinamičko Vezivanje metoda

Metoda se može implementirati (override-ati) u više klasa kroz lanac naslijeđivanja. JVM odlučuje koja će se metoda pozvati prilikom izvršavanja. Metoda može biti definisana u superklasi i override-ana u subklasi. Na primjer, metoda toString() je definisana u klasi Object, a mi je u svojim klasama override-amo. U sljedećem primjeru u klasi GeometricObject je override-ana toString() metoda:  **PRIMJER:**

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

Nameće se pitanje: koja je toString() metoda pozvana od objekta ''o''?  
Da bismo odgovorili na ovo pitanje, prvo ćemo uvesti nova dva pojma:  
- deklarisani tip  
- stvarni tip  
  
Varijabla mora biti deklarisana tipom (npr: int broj, Animal cuko.. isl).  
Tip koji deklariše varijablu se naziva deklarisani tip. U našem primjeru deklarisani tip objekta ''o'' je Object.   
  
Varijable referentnih tipova mogu čuvati NULL vrijednost ili referencu ( memorijsku adresu ) na instancu deklarisanog tipa. Instanca se može kreirati koristeći se konstruktorom od deklarisanog tipa ili subtipa!   
  
Stvarni tip varijable je stvarna klasa objekta na kojeg varijabla referencira! U našem primjeru, stvarna klasa objekta ''o'' je GeometricObject, zato što varijabla ''o'' referencira na objekat koji je kreiran korištenjem konstruktora GeometricObject (new GeometricObject()).   
 Object o = new GeometricObject();  
 DEKLARISANI TIP STVARNI TIP

Koja toString() metoda je pozvana od strane objekta ''o'' je određena stvarnim tipom objekta ''o''! Ovo je poznato kao Dinamičko vezivanje.

Dinamičko vezivanje radi po slijedećem principu:  
Pretpostavimo da je objekat ''o'' instanca klasa C1, C2, ... Cn-1, Cn.  
Gdje je C1 subklasa C2, a C2 subklasa C3 .... a Cn-1 subklasa Cn.  
Cn je generalna klasa (najsuper klasa, mama svima),   
a C1 je najproširenija klasa (najsubklasa, podojila ostale).

U javi, Cn je Object klasa.

Ako ''o'' pozove metodu ''nekaMetoda'', JVM traži implementaciju ''nekaMetoda'' metode u C1,C2,...Cn-1, Cn, po redu, sve dok je ne pronađe! Jednom kada pronađe implementaciju, obustavlja pretraživanje, te pozove prvu pronađenu implementaciju.

Cn

C2

C1

Cn-1

Ako je ''o'' instanca C1, ona je također instanca i C2,C3....Cn-1, Cn.

Metoda koja treba da se pozove je dinamički vezana pri izvršavanju.

public class Demo {

class Person extends Object {

@Override

public String toString() {

return ''Person'';

}

}

class Student extends Person {

@Override

public String toString() {

return ''Person'';

}

}

class GraduateStudent extends Student {

}  
  
 public static void print (Object x) {

System.out.println(x.toString());

}

public static void main (String [] args) {

print( new GraduateStudent());

print( new Student());

print( new Person());

print( new Object())  
 }

**OUTPUT**

1.Student

2.Student

3.Person

[4.java.lang.Object@130c19b](mailto:4.java.lang.Object@130c19b)

Metoda print prima parametar Object tipa. Možemo pozvati metodu print sa bilo kojim drugim objektom ( npr: new GraduateStudent, Student, Person...). Kada se metoda print izvrši, toString() metoda argumenta x je pozvana. X može biti instanca GraduateStudent, Student, Person ili Object. Svaka od navedenih klasa ima svoju implementaciju toString() metode. A koja će implementacija biti pozvana, zavisi od stvarnog tipa argumenta x pri izvršavanju. Pozivanje metode print( new GraduateStudent()) inicira pozivanje toString() metode definisane u klasi Student.

Podudaranje potpisa metode i vezivanje implementacije metode su dva različita problema! Deklarisani tip referentne varijable odlučuje koja se metoda podudara pri kompajliranju. Kompajler pronalazi podudarajuću (odgovarajuću) metodu na osnovu tipa, broja i redoslijeda parametara pri kompajliranju. Metoda može biti implementirana (override-ana) u više klasa kroz lanac naslijeđivanja. JVM dinamički vezuje implementacije metode pri izvršavanju, određenu od strane stvarnog tipa varijabli.

# Konverzija objekata (Casting Objects)

Konverzija objekata znači da referenca jednog objekta može biti pretvorena u referencu drugog objekta:

Object o = new Student(); //IMPLICITNA KONVERZIJA

U ovom slučaju, implicitna konverzija je ispravna, zato što je instanca Student instanca Object.

Pretpostavimo da želimo dodijeliti ''o'' objektnu referencu varijabli Student tipa, koristeći se sljedećim kodom:

Student b = o;

U tom slučaju bi se desila greška pri kompajliranju. Stoga se nameće pitanje zašto izjava Object o = new Student(); radi, ali Student b = o; ne radi?  
Razlog je taj što je objekat Student uvijek instanca Object-a, ali Object nije nužno instanca Student-a.   
Čak iako možemo vidjeti da je ''o'' zaista Student objekat, kompajler nije dovoljno pametan da zna to.   
  
Da bismo rekli kompajleru da je ''o'' Student objekat, moramo koristiti eksplicitnu konverziju. Sintaksa je slična kao i kod konverzije primitivnih tipova podataka:

Student b = (Student) o; //EKSPLICITNA KONVERZIJA

Uvijek je moguće pretvoriti instancu subklase u varijablu superklase (poznato kao upcasting), zato što je instanca subklase uvijek instanca superklase. Kada pretvaramo instancu superklase u varijablu subklase (poznato kao downcasting), moramo koristiti eksplicitnu konverziju, da bismo svoje namjere objasnili kompajleru.

# Operator instanceof

Da bi konverzija bila uspješna, moramo obezbijediti da je objekat kojeg ćemo pretvoriti instanca subklase. Ako objekat superklase nije instanca subklase, desit će se greška pri izvršavanju (ClassCastException). Na primjer: ako objekat nije instanca Student, ne može biti konvertovana u varijablu Student-a. Dobra je praksa da obezbijedimo da je objekat instanca drugog objekta, prije nego izvršimo konverziju. Da bismo to obezbijedili, koristimo se instanceof operatorom. Razmotrimo sljedeći kod:

Object myObject = new Circle();

if(myObject instanceof Circle){

System.out.println(''Dijametar kruga je: '' +

((Circle)myObject).getDiameter());

}

Kaže izvrši konverziju ako je myObject instanca Circle-a.

Zašto je konverzija važna?  
Varijabla myObject je deklarisana Object-om. Deklarisani tip odlučuje koja se metoda podudara (odgovara) pri kompajliranju. Koristeći myObject.getDiameter() bi izazvalo grešku pri kompajliranju, zato što Object klasa nema getDiameter() metodu, te je kompajler ne može naći. Zato je neophodno konvertovati myObject u Circle, kako bi to kompajler znao!

Nameće se pitanje zašto onda jednostavno ne definišemo myObject kao Circle tip, umjesto kao Object tip?   
Da bismo omogućili generičko programiranje, dobra je praksa definisati varijable supertipom, koje mogu prihvatiti objekte bilo kog subtipa.

Da bismo bolje shvatili konverziju, razmotrimo analogiju voća, jabuka i narandža. Voće klasa je superklasa Jabuka i Narandži. Jabuka je Voće, te uvijek sigurno možemo dodijeliti instancu Jabuka varijabli Voće. Međutim, Voće nije nužno Jabuka, te moramo koristiti eksplicitnu konverziju da dodijelimo instancu Voća varijabli Jabuka.

Kada konvertujemo primitivne vrijednosti, onda nam to vraća novu vrijednost:

int broj = 45;

byte noviBroj = (byte) broj;

//nova vrijednost je dodijeljena varijabli noviBroj

Međutim, kada konvertujemo složene vrijednosti, onda se ne kreira novi objekat:

Object o = new Circle();

Circle c = (Circle) o;

//nije se kreirao novi objekat!

Sada referentne varijable o i c ukazuju na isti objekat!

# Metoda equals klase Object

Kao što je toString() metoda korisna iz Object klase, tako je i equals metoda. Ova metoda testira da li su 2 objekta jednaka. Provjerava se da li referentne varijable ukazuju na istu memorijsku adresu, a ne jednakost sadržaja. Možemo override-ati metodu u našim klasama pa da se onda provjerava jednakost sadržaja dva objekta. String i Date klasa su override-ali metodu equals.

# ArrayList klasa

ArrayList objekat se može koristiti da pohrani listu objekata. ArrayLista nije fiksne dužine kao niz, u ArrayListu možemo pohraniti neograničen broj objekata.   
Neke od važnijih metoda ArrayListe:

add(objekat); //dodaje objekat  
get(index); //vraća objekat iz liste sa odgovarajućeg indexa.  
size(); //veličina liste  
remove(index ili objekat); //uklanja objekat iz liste   
indexOf(objekat); //vraća index odgovarajućeg objekta iz liste

ArrayList<String> imena = new ArrayList<>();

# Protected članovi

Protected članovima klase se može pristupiti iz subklase. Često je poželjno da omogućimo subklasama da pristupe poljima podataka ili metodama definisanih u superklasi, ali da onemogućimo nesubklasama pristup tip podacima i metodama. Da bi se ovo sprovelo, možemo koristiti ključnu riječ protected. Na ovaj način možemo pristupiti članovima superklase iz subklase.

Vidljivost članova se povećava prema sljedećem redoslijedu:

Private, Default, Protected, Public

Kod protected-a, nije važno da li su superklasa i subklasa u istom paketu, u svakom slučaju je omogućen pristup.   
  
Subklasa može override-ati protected metodu definisanu u Superklasi, te time promijeniti njenu vidljivost na public. Međutimm subklasa ne može oslabiti vidljivost metode definisane u superklasi. Na primjer: ako je metoda definisana kao public u superklasi, onda mora biti public i u subklasi!

# Zabrana naslijeđivanja i override-anja

Finalne klase i finalne metode se ne mogu naslijeđivati. Finalno polje podataka je konstanta. String i Math klasa su final klase.

public final class A{ //ne može bit naslijeđena

}

public final void print(){ //ne može bit override-ana

}