7 Polimorfizam

Polimorfizam je pojam vezan za nasleđivanje. Omogućava da se objekti izvedeni iz zajedničke superklase, bilo direktne ili indirektne, procesiraju kao da su u pitanju objekti superklase, što može značajno pojednostaviti razvoj aplikacije.

Posmatrajmo klasu `Oblik2D` sa slike 6.1 koja predstavlja superklasu klasa `Krug`, `Trougao`, `Pravougaonik`, kao i mnogih drugih 2D figura. Pretpostavimo da svaka od ovih klasa ima metodu za rotiranje koja se naziva `rotiraj`. Rotiranje svih objekata svih potklasa klase `Oblik2D` se elegantno može izvršiti koristeći polimorfizam na sledeći način. Kreiramo niz (ili kolekciju) referencijskih promenljivih superklase `Oblik2D` i u njega upišimo reference svih objekata izvedenih iz `Oblik2D`. Nakon toga, dovoljno je da prođemo kroz taj niz i za svaki objekat pozovemo metodu `rotiraj`. Svaka izvedena klasa ima svoj način rotiranja. Na primer, rotiranje kruga i kvadrata u 2D ravni nije isto. Dakle, pozivanjem metode `rotiraj`, mi svakom objektu *šaljemo istu poruku*, a u zavisnosti od predmetne klase, objekat će se rotirati na ovaj ili onaj način. Objekat *zna* kako da reaguje da poslatu poruku. Ovo je ključni koncept polimorfizma - ista poruka, poslata mnoštvu različitih objekata, ima više formi rezultata. Zbog ove višeobličnosti rezultata, sam koncept se naziva polimorfizam.

Polimorfizam: poruka poslata objektima različitih klasa ima *više formi* rezultata.

Polimorfizam značajno pojednostavljuje dodavanje novih klasa postojećoj aplikaciji, uz minimalnu modifikaciju kôda. Na primer, ukoliko želimo da uvedemo nove 2D oblike u našu aplikaciju, i da njih rotiramo, potrebno je izvesti odgovarajuću klasu i realizovati njenu metodu `rotiraj`. Nakon toga, novokreirana klasa može se dodati u aplikaciju bez značajne modifikacije postojećeg kôda, iako ta klasa nije bila planirana u početku. Modifikuje se samo kôd koji koristi novokreiranu klasu, tako što treba da uključi kreiranje instanci te klase i slanje odgovarajućih poruka. Na ovaj način, polimorfizam promoviše proširivost kôda.

Polimorfizam omogućava *uopšteno programiranje*, nasuprot *specifičnom programiranju*, koje je karakteristično za strukturne jezike. Koristeći polimorfizam, možemo programirati generalno, dok izvršnom okruženju (u slučaju Jave, to je JVM) prepuštamo da vodi računa o pojedinostima. Sve dok objekti pripadaju istoj hijerarhiji nasleđivanja, možemo pozivati metode instanci bez bojazni da će biti pozvana metoda neke od superklasa.

Tehnički, polimorfizam se implementira kreiranjem referencijskih promenljivih superklase i upisivanjem referenci potklasa u te promenljive. Za to je najjednostavnije koristiti niz

referencijskih promenljivih superklase. Prolaženjem kroz niz i pozivanjem metoda svakog objekta, neće se pozivati metoda superklase, već verzija metode koja odgovara tipu referenciranog objekta.

> Tip referenciranog objekta, a ne tip referencijske promenljive, određuje koja će se metoda pozvati.

Dodeljivanje vrednosti reference potklase promenljivoj tipa superklase je dozvoljeno zbog veze tipa jeste. Objekat potklase jeste objekat superklase. Obrnuto ne važi. Veza tipa jeste se, u hijerarhiji nasleđivanja, primenjuje samo od date klase naviše, prema njenim direktnim i indirektnim superklasama. Veza ne važi naniže.

Ukoliko referencijska promenljiva superklase sadrži referencu na objekat potklase, preko te promenljive možemo pristupiti samo delu potklase nasleđenom iz superklase, tj. ne može se pristupiti delu potklase koji je karakterističan za potklasu. Kad se preko referencijske promenljive superklase poziva nasleđena metoda, koja je redefinisana u potklasi, pozvaće se upravo ta redefinisana metoda, a ne istoimena metoda iz potklase.

Ako želimo da preko referencijske promenljive superklase pristupimo članovima karakterističnim za potklasu, referenca superklase se mora eksplicitno konvertovati u tip potklase. Ova konverzija se naziva *konverzija naniže* (eng. *downcasting*).

7.1 Primer polimorfizma: `Knjiga` i `KnjigaSaCenom`

Posmatrajmo klase `Knjiga` i `KnjigaSaCenom` iz poglavlja 6.3. Da bi demonstrirali polimorfizam, kreiraćemo dve referencijske promenljive, `knjiga1` i `knjiga2`, u koje ćemo upisati reference na objekte čiji tip odgovara tipu promenljive. Kreiraćemo referencijsku promenljivu `knjiga3`, tipa `Knjiga`, u koju ćemo upisati referencu na objekat tipa `KnjigaSaCenom`. Konačno, kreiraćemo referencijsku promenljivu `knjiga4`, tipa `KnjigaSaCenom`, u koju ćemo upisati konvertovanu vrednost promenljive `knjiga3`.

```
public class TestPolimorfizma {

  // Klasa koja testira polimorfizam na klasama Knjiga i KnjigaSaCenom

  public static void main(String[] args) {
    Knjiga knjiga1 = new Knjiga("Grof Monte Kristo", 234);
    KnjigaSaCenom knjiga2 = new KnjigaSaCenom("Mobi Dik", 477, 15.6);
    Knjiga knjiga3 = new KnjigaSaCenom("Životinjska farma", 111, 15.8);
    KnjigaSaCenom knjiga4 = (KnjigaSaCenom) knjiga3;
    // greška bi se desila ako umesto knjiga3 stavimo knjiga1

    System.out.println(knjiga1);
    System.out.println(knjiga2);
    System.out.println(knjiga3);
```

```
    System.out.println(knjiga4);
  }
}
```

```
Knjiga: Grof Monte Kristo
broj strana: 234

Knjiga: Mobi Dik
broj strana: 477
cena: 15.60 €

Knjiga: Životinjska farma
broj strana: 111
cena: 15.80 €

Knjiga: Životinjska farma
broj strana: 111
cena: 15.80 €
```

Što se tiče promenljivih `knjiga1` i `knjiga2`, situacija je jasna. Tipovi objekata i promenljivih su isti, pa ne dolazi do konverzije. Preko promenljive `knjiga3`, iako je u pitanju promenljiva superklase, poziva se metoda potklase. U Javi je dozvoljen ovaj *prelaz* sa superklasu na potklasu, jer objekat potklase jeste objekat superklase. Suprotno ne važi. Tokom izvršavanja, tip objekta na koji ukazuje promenljiva, a ne tip same promenljive, određuje koja verzija metode će se pozvati. Ovaj proces se naziva *dinamičko vezivanje* (eng. *dynamic binding*) ili *kasno vezivanje* (eng. *late binding*). Nasuprot dinamičkom, imamo statičko vezivanje, objašnjeno u poglavlju 7.4.

Vidimo da je dozvoljeno upisivanje vrednosti referencijske promenljive superklase `knjiga3` u promenljivu potklase `knjiga4`, pri čemu se mora izvršiti eksplicitna konverzija u tip potklase. Bez ove konverzije, došlo bi do greške.

Greška bi se desila i ako bi pokušali konvertovati promenljivu `knjiga3` u tip `KnjigaSaCenom`, a ona pritom ne sadrži referencu na objekat tipa `KnjigaSaCenom`. Drugim rečima, ako bismo u deklaraciji

```
KnjigaSaCenom knjiga4 = (KnjigaSaCenom) knjiga3;
```

umesto `knjiga3` stavili promenljivu `knjiga1`, koja je istog tipa kao `knjiga3`, ali sadrži referencu na objekat klase `Knjiga`, došlo bi do greške tokom izvršavanja, tj. bacio bi se izuzetak tipa `ClassCastException`.

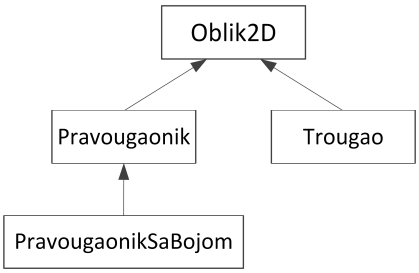
Konačno, greška bi se desila i ako pokušamo da pozovemo metodu `getCena` pomoću promenljive `knjiga3`. Iako `knjiga3`, koja je klase `Knjiga`, sadrži referencu na objekat tipa `KnjigaSaCenom`, koji ima metodu `getCena`, preko te promenljive možemo da pozovemo samo članove koje ima klasa `Knjiga`.

Konstruktori i statičke metode ne mogu biti apstraktni. Konstruktori se ne nasleđuju, pa se apstraktni konstruktor ne bi mogao implementirati. Statičke metode se mogu naslediti (ako su `public` ili `protected`), ali se ne mogu redefinisati.

Ne moraju sve superklase biti apstraktne. Ipak, definisanje apstraktnih superklasa može značajno pojednostaviti programiranje jer, uzimajući u obzir polimorfizam, na jednostavan način možemo manipulisati svim objektima potklasa preko promenljivih tipa apstraktne superklase. Uzmimo za primer program koji crta oblike izvedene iz apstraktne superklase `Oblik2D`, koja može imati veliki broj izvedenih potklasa. Štaviše, može se desiti da se nove izvedene klase mogu naknadno definisati i dodavati u program. Crtanje oblika vrši metoda `crtaj`, koja je implementirana za svaku konkretnu potklasu, i koja je navedena, a nije implementirana u klasi `Oblik2D`. Elegantan način da se izvede crtanje svih 2D oblika je da se definiše niz (ili kolekcija) promenljivih tipa `Oblik2D`, i da se u taj niz upišu reference na sve objekte potklasa. Nakon toga, obilazimo niz i pozivamo metodu `crtaj` svakog elementa niza, ne vodeći računa o tipu objekta. Dinamičko vezivanje nas oslobađa brige o tome. Dodavanje novih objekata ne menja postojeći programski kôd, bez samo u delu gde se kreiraju instance novokreirane klase i njihove reference upisuju u niz tipa `Oblik2D`.

7.3 Primer polimorfizma: `Oblik2D` i njene potklase

U nastavku dajemo primer polimorfnog procesiranja sa klasama čija je hijerarhija nasleđivanja data na slici 7.1.



Slika 7.1. Klasa Oblik2D i njene potklase.

Superklasa `Oblik2D` je apstraktna, njena metoda `povrsina` je takođe apstraktna. Klase koje direktno nasleđuju klasu `Oblik2D` su `Pravougaonik` i `Trougao`. Obe klase za svoje podatke imaju stranice, `Pravougaonik` dve i `Trougao` tri. Obe klase su konkretne, implementiraju metodu `povrsina`, svaka na način svojstven svom tipu. Površina trougla se računa prema Heronovom obrascu. Konačno, iz klase `Pravougaonik` izvodimo klasu `PravougaonikSaBojom`, koja sadrži dodatni podatak - `byte` niz od tri elementa (udeo crvene, zelene i plave boje), i metodu `getBoja()`, koja vraća `String` koji sadrži informaciju o boji pravougaonika. Udeo crvene, zelene i plave boje je ceo broj iz opsega [0, 255], pa ga je, u konstruktoru ove klase, potrebno svesti na opseg [-128, 127], što se vrši

```
    this.c = c;
  }

  public double povrsina() {
    // Površina trougla se računa po Heronovom obrascu
    double po = (getA() + getB() + getC()) / 2; // poluobim trougla
    return Math.sqrt(po * (po - getA()) * (po - getB()) * (po - getC()));
  }
}
-------------------------------------------------------------------------

public class TestPolimorfizma2 {

  // Klasa koja testira polimorfizam na klasi Oblik2D i izvedenim klasama

  public static void main(String[] args) {

    Oblik2D nizOblik2D[] = new Oblik2D[6];
    nizOblik2D[0] = new Pravougaonik(2.6, 3.4);
    nizOblik2D[1] = new Pravougaonik(3.3, 5.8);
    nizOblik2D[2] = new Trougao(3, 1.9, 2.6);
    nizOblik2D[3] = new Trougao(3, 4, 5);
    nizOblik2D[4] = new PravougaonikSaBojom(2.5, 2.3, 221, 99, 100);
    nizOblik2D[5] = new PravougaonikSaBojom(5.1, 6.7, 76, 145, 10);

    System.out.println("Štampanje svih objekata:");
    for(Oblik2D x: nizOblik2D)
      System.out.printf("Tip %s, površina %.2f\n",
                        x.getClass().getName(), x.povrsina());

    System.out.println("\nŠtampanje boja (PravougaonikSaBojom):");
    for(Oblik2D x: nizOblik2D)
      if(x instanceof PravougaonikSaBojom) {
        PravougaonikSaBojom p = (PravougaonikSaBojom) x;
        System.out.println(p.getBoja());
      }
  }
}
```

```
Štampanje svih objekata:
Tip Pravougaonik, površina 8.84
Tip Pravougaonik, površina 19.14
Tip Trougao, površina 2.45
Tip Trougao, površina 6.00
Tip PravougaonikSaBojom, površina 5.75
Tip PravougaonikSaBojom, površina 34.17

Štampanje boja (PravougaonikSaBojom):
(R,G,B) = (221,99,100)
(R,G,B) = (76,145,10)
```

Finalne metode se ne mogu redefinisati. Finalne klase se ne mogu naslediti. Finalni podaci se ne mogu menjati.

7.5 Interfejsi

Polimorfizam omogućava jednostavno procesiranje objekata koji imaju zajedničku superklasu. Zbog dinamičkog vezivanja, ne moramo voditi računa o tipu objekta sa kojim radimo. Sigurni smo da će se pozvati prava metoda koja odgovara tipu objektu. Međutim, ponekad treba da procesiramo objekte koji nemaju zajedničku superklasu, odnosno nemaju nikakve veze jedni sa drugim. Uzmimo, na primer, program za obračunavanje troškova vezanih za poslovanje firme. Firma ima zaposlene, koji primaju platu, a ima i račune za nabavljenu robu. Što se tiče obračunavanja troškova, poželjno je da se iznos plata i računa može dobiti uniformnim zapisom, a ne da vodimo računa o tipu svih objekata koji mogu biti uključeni u račun. U tom smislu, ne možemo koristiti polimorfizam kako smo ga do sad koristili, jer objekti tipa `Radnik` i `Racun` najverovatnije nemaju ništa zajedničko.

Interfejsi (eng. *interfaces*) omogućavaju da klase koje nisu povezane implementiraju skup zajedničkih metoda. Kada proglasimo da klase implementiraju određeni interfejs, objekti tih klasa mogu biti uniformno procesirani u smislu da se mogu pozivati metode interfejsa. Takođe, objekti mogu komunicirati preko interfejsa.

Interfejsi omogućavaju polimorfno procesiranje objekata koji nemaju zajedničku superklasu.

7.5.1 Deklarisanje interfejsa

Deklaracija interfejsa započinje ključnom rečju `interface`, i sadrži samo konstante i apstraktne metode. Za razliku od klasa, svi članovi interfejsa moraju biti `public` i interfejsi ne mogu sadržati nikakvu specifikaciju metoda. Sve metode deklarisane u interfejsu su implicitno `public abstract`, dok su podaci implicitno `public abstract final`. U skladu sa specifikacijom Jave, ne navode se ključne reči `public`, `static` i `final` pri deklaraciji metoda i podataka interfejsa. Opšti oblik deklaracije interfejsa je dat u nastavku.

```
public interface Interfejs {
  tip metoda1(lista parametara);
  tip metoda2(lista parametara);
  ...
  tip metodaN(lista parametara);

  tip prom1 = vrednost1;
  tip prom2 = vrednost2;
  ...
  tip promM = vrednostM;
```

7.5.5 Primer korišćenja interfejsa

Kao primer korišćenja interfejsa, uzmimo primer klasa `Oblik2D`, `Pravougaonik`, `Trougao` i `PravougaonikSaBojom` realizovanih u poglavlju 7.3. Pretpostavimo da, pored ovih klasa, imamo i klasu `Prozor`, koja će predstavljati grafički prozor gde, recimo, možemo da crtamo objekte `Pravougaonik`, `Trougao` i `PravougaonikSaBojom`. Klasa `Prozor` nije srodna klasi `Oblik2D`, ali ima jednu zajedničku karakteristiku, a to je površina. I 2D oblici i grafički prozori imaju površinu. Zajednička akcija koja se može izvesti za klase izvedene iz `Oblik2D` i klasu `Prozor` je, na primer, pomeranje. Oblici se mogu pomerati u okviru grafičkog prozora, a grafički prozor se može pomerati po ekranu monitora.

Posmatrajmo samo računanje površine oblika i prozora. Da bismo ovu zajedničku akciju obavili uniformno, definisaćemo interfejs `Povrsina` koji sadrži samo jednu apstraktnu metodu `getPovrsina()`.

```
public interface Povrsina {
  double getPovrsina();  // izračunavanje površine, bez implementacije
}
```

Ovaj interfejs se mora naći u okviru fajla `Povrsina.java`.

Dalje, dajemo realizaciju klase `Prozor`, koja ima dva podatka, širinu i visinu, odgovarajuće metode `set` i `get`, kao i redefinisanu metodu `getPovrsina()`. Ova klasa implementira interfejs `Povrsina`, što je navedeno u zaglavlju klase.

```
public class Prozor implements Povrsina {

  // Klasa koja predstavlja grafički prozor

  double sirina, visina;

  public Prozor() {
    this(0, 0);
  }

  public Prozor(double x, double y) {
    setSirina(x);
    setVisina(y);
  }

  public double getSirina() {
    return sirina;
  }

  public void setSirina(double x) {
    sirina = x;
  }

  public double getVisina() {
```

```
    return visina;
  }

  public void setVisina(double y) {
    visina = y;
  }

  // Implementacija metode getPovrsina() iz interfejsa Povrsina
  @Override
  public double getPovrsina() {
    return getSirina() * getVisina();
  }

  public String toString() {
    return String.format("Prozor širine %.2f i visine %.2f",
                         getSirina(), getVisina());
  }
}
```

Da bi klasa `Oblik2D` i sve njene izvedene klase implementirale interfejs `Povrsina`, potrebno je navesti da klasa `Oblik2D` implementira taj interfejs na sledeći način:

```
public abstract class Oblik2D implements Povrsina
```

Realizacija ove klase se ne menja. Pošto ova klasa neće implementirati metodu `getPovrsina()`, nećemo navoditi tu metodu u okviru klase, iako se to može uraditi na sledeći način:

```
abstract public double getPovrsina();
```

Da bi ispunile ugovor koji je superklasa `Oblik2D` potpisala sa kompajlerom, njene potklase `Pravougaonik` i `Trougao` moraju implementirati metodu `getPovrsina()`, što se može uraditi dodavanjem sledećeg kôda unutar njihove realizacije (nećemo ponavljati čitavu realizaciju):

```
@Override
public double getPovrsina() {
  return povrsina();
}
```

Nema potrebe navoditi u zaglavlju klasa `Pravougaonik` i `Trougao` da one implementiraju interfejs `Povrsina`, pošto je njihova superklasa to navela.

Konačno, ništa se neće promeniti u klasi `PravougaonikSaBojom`, jer ona nasleđuje implementaciju metode `getPovrsina()` iz direktne superklase `Pravougaonik`, koja se za svrhu računanja površine ne mora redefinisati.

Klasa `InterfejsTest`, koja testira rad sa interfejsom, data je u nastavku.

```
public class InterfejsTest {

  // Klasa koja testira interfejs Povrsina
```