6 Nasleđivanje

6.1 Uvod u nasleđivanje

Kreiranje novog softvera retko kad počinje od nule, već se zasniva na postojećem softveru. Najbolji i najbrži način kreiranja je imitirajući postojeći softver, poboljšavajući ga i kombinujući sa drugim softverom. Tradicionalni softverski dizajn je dobrim delom zanemarivao ovaj aspekt razvoja softvera, dok je u OO dizajnu on od suštinskog značaja.

Nasleđivanje predstavlja oblik ponovne upotrebe softvera kojim se, iz postojeće klase, kreira nova klasa. Nova klasa nasleđuje članove postojeće klase, uz mogućnost da doda nove članove ili modifikuje postojeće. Na ovaj način se smanjuje vreme razvoja aplikacije korišćenjem postojećeg proverenog programskog kôda. Postojeća klasa naziva se *superklasa* ili *osnovna klasa*, a klasa dobijena nasleđivanjem naziva se *potklasa* ili *izvedena klasa*. Proces nasleđivanja se može nastaviti u smislu da nova klasa može predstavljati superklasu drugih klasa.

Pored postojećih podataka i metoda, potklasa može definisati svoje podatke i metode, na ovaj način postajući specifičnija od svoje superklase. Potklasa može modifikovati nasleđene podatke i metode, prilagođavajući ih svom tipu. Zbog ovoga se nasleđivanje još naziva i *specijalizacija*. Superklase su opštije, a potklase specifičnije. Za potklasu se još kaže da *proširuje* (eng. *extends*) superklasu, jer može dodati nove članove u odnosu na superklasu.

> Nasleđivanje je proces kojim jedan objekat dobija svojstva drugoga, uz mogućnost da ta svojstva unapredi. Nasleđivanje podržava ponovnu upotrebu kôda čime omogućava da složenost OO programa raste linearno, a ne geometrijski.

U hijerarhiji nasleđivanja, potklasa može imati *direktnu superklasu* i *indirektne superklase*. Indirektna superklasa je bila koja klasa u hijerarhiji nasleđivanja iznad direktne superklase. Najviša klasa u Javinoj klasnoj hijerarhiji je `Object` (deklarisana u paketu `java.lang`) i sve Javine klase direktno ili indirektno nasleđuju ovu klasu.

Nasleđivanje odražava *vezu tipa jeste* (eng. *is-a relationship*), nasuprot kompozicije koja odražava vezu tipa ima. U vezi tipa jeste, objekat potklase se može tretirati i kao objekat superklase. Pošto objekat potklase *jeste* objekat superklase, i jedna superklasa može imati više potklasa, skup objekata superklase je veći od skupa objekata bilo koje potklase.

potklase može direktno pristupiti tom podatku i menjati ga. Drugi problem koji se može desiti je da metode potklasa zavise od implementacije superklase, što je loša programerska praksa. Potklase treba da zavise samo od javnih servisa superklase (`public` ili `protected` metoda), a ne od implementacije podataka superklase. Ako bi metode potklasa zavisile od implementacije podataka superklase, svaka promena implementacije podataka superklase bi implicirala promenu svih potklasa koje zavise od te implementacije. Na ovaj način bi mala promena u superklasi uzrokovala velike promene u potklasama.

	Public	Protected	Bez modifikatora	Private
Ista klasa	Da	Da	Da	Da
Potklasa iz istog paketa	Da	Da	Da	Ne
Klasa iz istog paketa koja nije potklasa	Da	Da	Da	Ne
Potklasa iz drugog paketa	Da	Da	Ne	Ne
Klasa iz drugog paketa koja nije potklasa	Da	Ne	Ne	Ne

Tabela 6.1. Prava pristupa u Javi.

Jedan od ključnih faktora kvaliteta softvera je proširivost ili skalabilnost, i predstavlja mogućnost prilagođavanja softvera promenama u njegovoj specifikaciji. Kao što ime kaže, softver treba da bude mek (eng. *soft*). U direktnoj vezi sa proširivošću softvera je kriterijum *modularnog kontinuiteta* (eng. *modular continuity*) po kome bi mala promena u specifikaciji problema kojim se bavi određeni modul trebala da se odrazi samo na taj modul ili na mali broj modula. Ako bismo matematički predstavili funkciju promene softvera (y osa) u zavisnosti od promene specifikacije (x osa), ta funkcija ne bi smela da ima naglih skokova, tj. njen prvi izvod bi trebao biti mali u čitavom domenu funkcije. Idealan slučaj bi bio da se softver ne menja u zavisnosti od promene specifikacije.

Korišćenje `protected` podataka superklase je u suprotnosti sa proširivošću softvera i modularnim kontinuitetom, dok im upotreba metoda `set` i `get` za manipulaciju podacima superklase ide u prilog.

Prava pristupa predstavljaju mehanizam prevencije grešaka usled pogrešne upotrebe članova klase. Pravilo je da se koristi najrestriktivnije pravo pristupa koje ima smisla za određeni član klase, tj. da se ne koristi `private` samo kad postoji valjan razlog za to.

6.3 Primer nasleđivanja: `KnjigaSaCenom`

Pređimo, konačno, na primer nasleđivanja. Kao superklasu, koristićemo klasu `Knjiga` iz poglavlja 3.1. Ovde ćemo ponoviti realizaciju klase `Knjiga` uz sledeće modifikacije: dodati su konstruktori, dodate su metode `postaviKnjigu` i `toString`, uklonjena je metoda `prikaziKnjigu`, i unapređena je metoda `setBrojStrana` proverom validnosti parametra metode i bacanjem izuzetka po potrebi.

```
public class Knjiga {

  private String ime;
  private int brojStrana;

  public Knjiga() {
    this(null, 0);
  }

  public Knjiga(String imeKnjige) {
    this(imeKnjige, 0);
  }

  public Knjiga(int brStr) {
    this(null, brStr);
  }

  public Knjiga(String imeKnjige, int brStr) {
    postaviKnjigu(imeKnjige, brStr);
  }

  public void postaviKnjigu(String imeKnjige, int brStr) {
    setIme(imeKnjige);
    setBrojStrana(brStr);
  }

  public String getIme() {
    return ime;
  }

  public void setIme(String imeKnjige) {
    ime = imeKnjige;
  }

  public int getBrojStrana() {
    return brojStrana;
  }

  public void setBrojStrana(int brStr) {
    if(brStr > 0)
      brojStrana = brStr;
    else
      throw new IllegalArgumentException("Broj strana nije pozitivan");
  }

  public String toString() {
    return String.format("Knjiga: %s\nbroj strana: %d\n",
                         getIme(), getBrojStrana());
  }
}
```

```
  public double getCena() {
    return cena;
  }

  public void setCena(double cenaKnjige) {
    if(cenaKnjige >= 0)
      cena = cenaKnjige;
    else
      throw new IllegalArgumentException("Cena je negativna");
  }

  @Override
  public String toString() {
    return String.format("%scena: %.2f \u20AC\n", super.toString(),
                         getCena());
  }
}
-------------------------------------------------------------------------

// Klasa koja testira klasu KnjigaSaCenom

public class KnjigaSaCenomTest {

  public static void main(String[] args) {
    KnjigaSaCenom knjiga = new KnjigaSaCenom("Mobi Dik", 477, 15.6);
    System.out.print(knjiga);
  }
}
```

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

Uočite način zadavanja Unicode karaktera € kao `\u20AC` u metodi `format` klase `String`.

6.4 Realizacija nasleđivanja

6.4.1 Ključna reč `extends`

Ključna reč `extends` označava nasleđivanje. Klasa `KnjigaSaCenom` nasleđuje podatke i metode klase `Knjiga`, ali su samo `public` i `protected` članovi superklase dostupni potklasi, `private` nisu. Članovima superklase bez navedenog modifikatora pristupa (pravo pristupa paketa) mogu pristupiti samo potklase iz istog paketa (videti tabelu 6.1). Konstruktor superklase se ne nasleđuje.

Sve Javine klase direktno ili indirektno nasleđuju klasu `Object`. U tom smislu, u deklaraciji klase `Knjiga` mogli smo navesti da ona nasleđuje klasu `Object` na sledeći način:

6.4.3 Redefinisanje metoda i anotacija @Override

U slučaju kad klasa treba da redefiniše neku metodu superklase, u smislu da je prilagodi svome tipu, možemo koristiti anotaciju `@Override`. Metoda `toString()` klase `Knjiga` mora se prilagoditi klasi `KnjigaSaCenom` tako da se pri ispisu uključi i cena knjige. Kad kompajler naiđe na metodu deklarisanu sa `@Override` proverava da li potpis metode potklase odgovara potpisu metode superklase. Ukoliko oni nisu potpuno isti (isto ime metode i redosled i tip parametara metode), dolazi do greške kompajliranja. Upotreba anotacije `@Override` nije obavezna.

Potpis metode potklase deklarisane sa `@Override` mora biti identičan potpisu metode superklase koja se redefiniše. U suprotnom, dolazi do greške kompajliranja.

Greška je pokušati promeniti modifikator pristupa redefinisane metode u modifikator sa manje prava pristupa. Drugim rečima, `public` metoda ne može postati `protected` ili `private` u potklasi, dok `protected` metoda ne može postati `private` u potklasi. Ukoliko bi to bilo dozvoljeno, narušila bi se veza tipa jeste: `KnjigaSaCenom` jeste `Knjiga`, i moramo biti u mogućnosti da sve javne servise koje obezbeđuje klasa `Knjiga` koristimo i kod klase `KnjigaSaCenom`.

`public` metoda superklase ostaje `public` u svim njenim potklasama.

Napomenimo da su redefinisane metode u Javi slične virtuelnim funkcijama u C++ i C#.

6.4.4 Ponovna upotreba kôda u metodama potklase

Zapisom `super.imeMetode`, iz potklase može se pozvati bilo koja javna zasenjena metoda superklase. Ovo se ponekad može efikasno inkorporirati u metode potklase, dajući elegantno rešenje za dati problem. Konkretno, u redefinisanoj metodi `toString`, nismo iznova kreirali `String` reprezentaciju objekta, već smo pozvali istoimenu metodu superklase koja je kreirala deo stringa koji se odnosi na zajednički deo podataka (ime knjige i broj strana), a zatim nadovezali deo stringa karakterističan za potklasu. U nastavku dajemo metodu `toString` klase `KnjigaSaCenom` sa i bez korišćenja istoimene metode superklase.

```
public String toString() {
  return String.format("%scena: %.2f \u20AC\n", super.toString(),
                       getCena());
}

public String toString() {
  return String.format("Knjiga: %s\nbroj strana: %d\ncena: %.2f \u20AC\n",
                       getIme(), getBrojStrana(), getCena());
}
```