Generičke klase

Autori:
Milan Segedinac
Goran Savić

1. Generička lista

Metode su nam omogućile da namenu algoritma apstrahujemo od njegove implementacije i da dobijenu metodu dalje koristimo kao gradivni blok u drugim metodama. Ovo građenje složenih operacija od primitivnih i već razvijenih operacija je *proceduralna apstrakcija*. Kada smo uveli pojam *objekta*, uveli smo novi moćan mehanizam apstrakcije: pravljenje složenih objekata od primitivnih vrednosti i već napravljenih objekata. Ovaj mehanizam apstrakcije zove se *apstrakcija podataka* i u programskom jeziku Java realizuje se pisanjem *klasa*. Postojanje klasa u programskom jeziku nije preduslov za objektno orjentisano programiranje – kada bude bilo reči o programskom jeziku JavaScript videćemo kako možemo da kreiramo objekte i ostvarimo apstrakciju podataka bez oslanjanja na klase.

Kod metode nam je bilo dovoljno da znamo naziv, koje parametre prima i koji je tip vrednosti koju vraća da bismo je koristili. (Nije nam bilo potrebno da znamo *kako* je implementirana). Za klasu nam treba da znamo kako se zove, koje parametre prima njen konstruktor i koje metode ima da bismo koristili njene instance (objekte). Ne moramo da znamo kako je zaista implementirana. U ovoj lekciji ćemo videti kako klasa `ArrayList` apstrahuje niz. Klasu `ArrayList` ćemo iskoristiti za rukovanje podacima u aplikaciji za evidentiranje studenata. Da bismo mogli da objasnimo kako je ta klasa implementirana prvo moramo da pokažemo kako u programskom jeziku Java možemo da radimo sa strukturama podataka apstrahujući se od njihovog tipa, pa ćemo prvo uvesti pojmove hijerarhije tipova i generičkih tipova.

Hijerarhija tipova

Uočili smo da programski jezik Java ima tip za reprezentovanje celih brojeva (`Integer`) i tip za reprezentovanje razlomljenih brojeva (`Float`). I jednu i drugu klasu možemo posmatrati kao specijalni slučaj *broja*, koji je u programskom jeziku Java predstavljen klasom `Number`. Kažemo da i `Integer` i `Float` *nasleđuju* `Number`, odnosno klasa `Number` okuplja vrednosti i operacije koje su zajedničke za sve brojeve. Činjenica da `Float` nasleđuje `Number` omogućuje nam da svaki objekat klase `Float` možemo da posmatramo i kao `Number`. Stoga ćemo varijabli tipa `Number` moći da dodelimo vrednost koja je tipa `Float`. Međutim, ako bismo neki `Number` objekat hteli da posmatramo kao `Float`, morali bismo ga eksplicitno prevesti (konvertovati) u `Float`, kao što je prikazano lisingom ispod.

```
Number x;
Float f = new Float(5);
x = f; // implicitna konverzija
f = (Float) x; // eksplicitna konverzija
```

Listing – Eksplicitna konverzija

Generičke klase

Videli smo da se skup tipova programskog jezika Java može proširiti uvođenjem klasa. Klasa je recept po kom se instanciranjem kreiraju objekti i specificira koje atribute (podatke) i metode (funkcije) će objekat imati. Prilikom nasleđivanja, klasa preuzima sve atribute i metode klase koju nasleđuje. Nasleđivanje se u programskom jeziku Java realizuje pomoću ključne reči **extends**. Jedan primer nasleđivanja dat je listingom ispod.

```
public class Osoba {
	String ime;
	String prezime;
	public String predstaviSe(){
		return "zovem se "+ime+" "+prezime;
	}
}

public class Student extends Osoba{
	String brojIndeksa;
	public String prikaziPodatke(){
		return "ime: "+ime+"; prezime: "+prezime+"; broj indeksa:
"+brojIndeksa;
	}
}

…
public static void main(String[] args) {
		Student s = new Student();
		s.ime = "Pera";
		s.prezime = "Peric";
		s.brojIndeksa = "E10750";
		System.out.println(s.predstaviSe());
		System.out.println(s.prikaziPodatke());
	}
```

Listing – implementacija nasleđivanja

Klasa `Osoba` ima atribute `ime` i `prezime` i metodu `predstaviSe`. Klasa `Student` ima atribut `brojIndeksa` i metodu `prikaziPodatke`. Pored toga, klasa `Student` nasleđuje klasu `Osoba`, time što smo u definiciji klase `Student` naveli `extends Osoba`. Time smo postigli da će klasa `Student` imati i sve atribute i metode klase `Osoba`, pa ćemo za objekat `s` (instancom klase `Student`) moći da postavimo i `ime` i `prezime` i moćićemo da pozovemo i metodu `predstaviSe`.

Nasleđivanje je tema od ključne važnosti u objektno orjentisanom programiranju i ovom temom ćemo se kasnije baviti vrlo detaljno. Za sada nam je dovoljno da znamo da postoji klasa `Object` koju nasleđuju sve ostale klase i da svaki objekt možemo posmatrati kao da je instanca ove klase.

Generički tipovi

Videli smo da je niz kolekcija podataka u kojoj se za svaki element tačno zna na kojoj poziciji se nalazi. Međutim, operacije kao što su dodavanje elementa na zadatu poziciju u niz ili brisanje elementa sa zadate pozicije bile su komplikovane. `java.util.ArrayList` (u daljem tekstu `ArrayList`) je klasa programskog jezika Java koja sadrži metode koje omogućuju operacije koje se tipično obavljaju nad nizom (dodavanje, brisanje, pronalaženje), a pri tome skriva detalje implementacije od korisnika.

Prilikom kreiranja niza, specificirali smo kog tipa će biti vrednosti tog niza (`int[] niz`) i u niz koji je, recimo, tipa `int`, nismo mogli da dodajemo vrednosti koje su tipa `String`. Za takve kolekcije kažemo da su *homogene*. Poželjno je da možemo da napravimo homogenu listu, odnosno da prilikom instaciranja objekta tipa *ArrayList* možemo da navedemo kog tipa će biti objekti smešteni u nju. Ovo možemo ostvariti pomoću posebnog konstrukta programskog jezika Java koji se zove *generički tip*.

Generički tip je klasa *parametrizovana tipovima*. Na primer, ako bismo hteli da napravimo klasu koja skladišti neke vrednosti to bismo pomoću generičkog tipa mogli da realizujemo kao što je prikazano na listingu ispod.

```
public class Kutija<T> {

	private T sadrzaj;

	Kutija(T sadrzaj){
		this.sadrzaj = sadrzaj;
	}
	public T getSadrzaj(){
		return sadrzaj;
	}
}
```

Listing – Generička klasa

Kao što vidimo na listingu iznad, nakon naziva generičke klase sledi spisak parametara tipova u špicastim zagradama (između znakova < i >). Tako deklaracija klase Kutija izgleda kao `public class Kutija<T>`.

Parametre tipova koje smo ovako zadali koristimo u programskom kodu na svakom mestu na kom bismo koristili konkretan tip. Kada deklarišemo polje `sadrzaj` parametrizovanog tipa `T`, to radimo sa `private T sadrzaj`. Kada naznačavamo da funkcija vraća vrednost koja je parametrizovanog tipa `T`, to radimo sa `public T getSadrzaj()`.

Kada kreiramo objekte klase `Kutija` navodimo koju konkretnu vrednost će imati parametar tipa `T`, kao što je prikazano listingom ispod.

```
int x = 1;
String s = "Zdravo!";
Kutija<Integer> kutijaInt = new Kutija<>(x);
Kutija<String> kutijaStr = new Kutija<>(s);
```

Listing – Kreiranje objekata generičkog tipa

Kada deklarišemo generičku klasu, zadajemo koji konkretan tip će biti korišćen. Tako smo, kada smo hteli da radimo sa String vrednošću deklarisali `Kutija<String> kutijaStr = new Kutija<>(s);`

Obratite pažnju da smo deklarisali `Kutija<Integer>` iako je vrednost `x` tipa `int`. To je zbog toga što generički tipovi u Javi mogu biti parametrizovani samo neprimitivnim tipovima. U prethodnoj lekciji bilo je reči o boksingu i videli smo da u programskom jeziku Java vrednost primitivnog tipa `int` može da se dodeli varijabli koja je klase `Integer`.

ArrayList – generička lista

ArrayList je Java klasa koja pojednostavljuje manipulaciju uređenim kolekcijama. Listing ispod prikazuje kako se ArrayList instancira. Vidimo da smo listu instancirali navodeći argument tipa `String`. To znači da ćemo imati listu stringova.

```
ArrayList<String> lista = new ArrayList<String>();
```

Listing – instanciranje ArrayList

ArrayList omogućuje jednostavno dodavanje element na kraj i dodavanje elementa na proizvoljnu poziciju, što možemo da vidimo na listingu ispod.

```
//dodavanje na kraj liste
lista.add("Lorem ipsum");
lista.add("dolor sit amet");
//dodavanje na poziciju sa indeksom 1
lista.add(1, "consectetur adipiscing elit");
```

Listing – dodavanje elemenata u listu

Prva linija koda dodala je string `"Lorem ipsum"` na kraj prazne liste, tako da je tada lista bila `["Lorem ipsum"]`. Sledeće linija dodala je novi string (`"dolor sit amet"`) na kraj lista, pa je nakon te linije koda lista bila `["Lorem ipsum","dolor sit amet"]`. Poslednja linija koda dodala je novi string **na poziciju sa indeksom 1**, pa je lista nakon ove linije koda bila `["Lorem ipsum","consectetur adipiscing elit","dolor sit amet"]`.

ArrayList omogućuje preuzimanje elementa iz liste sa zadate pozicije. Za to se koristi metoda `get` koja kao parametar prima indeks pozicije sa koje se preuzima element. Poziv `lista.get(1);` vratio bi element na indksu 1, odnosno string `"consectetur adipiscing elit"` za listu iz prethodnog primera.

ArrayLista takođe pojednostavljuje i brisanje elementa iz liste. To se ostvaruje pozivom metode `remove` kojoj se prosleđuje indeks elementa koji treba obrisati. Poziv `lista.remove(1);` obrisao bi element na indeksu 1, pa bi lista iz prethodnog listinga nakon poziva ove metode bila `["Lorem ipsum", "dolor sit amet"]`. Pored ovih metoda, ArrayList ima i metode za preuzimanje dužine liste, proveru da i je element sadržan u listi, proveru da li je lista prazna, itd. Pregled najvažnijih metoda ArrayList dat je tabelom ispod.

Metoda	Objašnjenje
`void add(int index, Object element)`	Dodavanje elementa na poziciju zadatu indeksom
`boolean add(Object o)`	Dodavanje elementa na kraj liste
`void clear()`	Uklanjanje svih elemenata liste
`boolean contains(Object o)`	Provera da li lista sadrži dati objekat
`Object get(int index)`	Preuzimanje elementa sa zadate pozicije
`int indexOf(Object o)`	Preuzimanje indeksa zadatog elementa
`Object remove(int index)`	Uklanjanje elementa sa zadate pozicije
`Object set(int index, Object element)`	Zamena objekta u listi novim na zadatoj poziciji
`int size()`	Broj elemenata u listi
`Boolean isEmpty()`	Provera da li je lista prazna

Tabela – Metode klase ArrayList