Generičko programiranje u Javi

Sadržaj

Potreba za uvođenjem generičkog programiranja

Korištenje generičkih parametara

Prevođenje generičkih metoda

Generičke klase

Nedefinirani tipovi

Zamjenski simboli

Potreba za uvođenjem generičkog programiranja

- Ako je potrebno napisati metodu koja će ispisivati cjelobrojne članove jednodimenzionalnog polja, to je moguće napraviti na sljedeći način:

```
Integer[] integerArray = {1, 2, 3, 4, 5, 6};

System.out.printf("Array integerArray sadrži: ");
printArray(integerArray);

public static void printArray(Integer[] inputArray) {
  for (Integer element : inputArray) {
         System.out.printf("%s ", element);
  }
  System.out.println();
}
```

Potreba za uvođenjem generičkog programiranja

- U slučaju da je potrebno ispisati elemente polja koje sadrži npr. „Double” ili „Character” tipove podataka, potrebno je napisati još jednu *overloadanu* metodu za ispis koja će se razlikovati samo u tipu elemenata koje ispisuje:

```
public static void printArray(Double[] inputArray) {
    for (Double element : inputArray) {
          System.out.printf("%s ", element);
    }
       System.out.println();
}

public static void printArray(Character[] inputArray) {
    for (Character element : inputArray) {
          System.out.printf("%s ", element);
    }
    System.out.println();
}
```

Potreba za uvođenjem generičkog programiranja

- Kad kompajler poziva metodu za ispis polja, pokušava locirati deklaraciju metode koja ima zadano ime i prima parametre koji su navedeni kod poziva
- Sve metode za ispis polja su vrlo slične i razlikuju se samo po tipu podataka u polju – ta činjenica može se iskoristiti za pisanje samo jedne metode koja koristi generički parametar tipa **T** (koji predstavlja sve referentne tipove u Javi)
- Ta metoda može izgledati ovako:

```
public static <T> void printArray(T[] inputArray)
{
   for (T element : inputArray) {
         System.out.printf("%s ", element);
   }
   System.out.println();
}
```

Korištenje generičkih parametara

- Svaka generička metoda ima definiciju parametra **<T>** koji označava da će se unutar same metode koristiti generički tip s oznakom „**T**”
- Pozivi generičkih metoda su identični pozivima metoda bez generičkih parametara
- Taj parametar može se koristiti za definiranje povratnog tipa varijable ili tipova ulaznih argumenata metode
- Svaki parametar se kod deklariranja naziva i parametara metode može pojavljivati više puta kad je potrebno koristiti određeni tip podatka
- Na primjer, moguće je napisati metodu koja prima dva ulazna parametra istog tipa i vraća taj isti tip podatka:

```
public static <T> T maximum(T vrijednost1, T vrijednost2)
```

Prevođenje generičkih metoda

- U slučaju kad kompajler prevodi generičku metodu u *bytecode*, sva pojavljivanja parametra **T** zamjenjuje sa stvarnim tipom podatka
- Po *defaultu* se T mijenja **Object** tipom
- Osim konkretnih tipova koji se koriste u generičkim metodama moguće je postaviti i ograničenja koja moraju ispunjavati parametri neke metode
- Na primjer, ako je potrebno ograničiti da metoda prima samo objekt koji nasljeđuje klasu „Student”, onda je to moguće napisati na sljedeći način:

```
public static <T extends Student> void usporedi(T vrijednost1, T vrijednost2)
```

Generičke klase

- Osim generičkih metoda, postoje i generičke klase kod kojih je moguće koristiti generičke tipove (npr. „ArrayList”)
- Takve klase često se nazivaju i parametriziranim klasama ili tipovima
- Omogućavaju definiranja tipova objekata prilikom instanciranja
- Primjer definiranja generičke klase koja predstavlja memorijsku strukturu stoga:

```
public class Stack<T>
{
    private List<T> elements;
    …
```

Nedefinirani tipovi

- engl. *Raw Types*
- U slučaju kad se koristi generička klasa i kod instanciranja se ne navede tip podatka koji će generička klasa koristiti, implicitno se za taj tip koristi „Object”, na primjer:

```
Stack doubleStack = new Stack(5);
```

- Ta mogućnost je u Javi ostavljena zbog unazadne kompatibilnosti (engl. *backward compatibility*) s prošlim verzijama Jave (prije verzije 5) u kojima nije postojala mogućnost korištenja generičkog programiranja
- Iako je to moguće, preporuča se izbjegavanje korištenja nedefiniranih tipova, jer se u tom slučaju mogu koristiti svi mogući tipovi podataka i povećava se vjerojatnost generiranja iznimke „ClassCastException”

Nedefinirani tipovi

- Osim toga prevoditelj javlja upozorenje (engl. *warning*) u slučaju korištenja generičkog tipa:

Stack is a raw type. References to generic type Stack<T> should be parameterized

- Kod dodavanja podataka u generičku klasu se također javlja upozorenje, a kod dohvaćanja podataka iz nje je potrebno koristiti *cast* operaciju:

```
rawStack.push(123);
Integer broj = (Integer) rawStack.pop();
```

Type safety: The method push(Object) belongs to the raw type Stack. References to generic type Stack<T> should be parameterized

Zamjenski simboli

- engl. *Wildcards*
- Ako je potrebno napisati funkciju koja izračunava sumu numeričkih vrijednosti spremljenih u zbirku kao što je lista, također je moguće koristiti generičku metodu
- Jedina stvar koju u tom slučaju treba ograničiti je da zbirka sadržava samo numeričke vrijednosti, odnosno objekte koji izravno ili neizravno nasljeđuju klasu „Number” što se označava na sljedeći način:

```
public static double sum(List<? extends Number> list){
      double total = 0;
      for (Number element : list) {
         total += element.doubleValue();
      }
      return total;
}
```

Zamjenski simboli

- Zamjenski simbol „?” označava „nepoznati tip” koji mora nasljeđivati klasu „Number”
- Mana korištenja zamjenskih simbola je u tome što zbog korištenja „?” nije moguće znati o kojem se točno tipu radi pa taj tip nije ni moguće koristiti unutar tijela metode (ili klase), za razliku od slučaja kad se koristi simbol poput „T”
- Ako je to potrebno omogućiti, onda je moguće koristiti sljedeće označavanje:

```
public static <T extends Number> double sum(List<T> list){
```

Pitanja?