

Programiranje 1

Poglavje 8: Generiki

Luka Fürst

Primer

- preprost ovojni razred za spremenljivko poljubnega referenčnega tipa

```
class Ovojnik {  
    private Object a;  
  
    public Ovojnik(Object a) {  
        this.a = a;  
    }  
  
    public Object vrni() {  
        return this.a;  
    }  
}
```

Primer

- uporaba razreda

```
Ovojn timer p = new Ovojn timer("dober dan");  
String s = (String) p.vrni();
```

```
Ovojn timer q = new Ovojn timer(42);  
Integer n = (Integer) q.vrni();
```

```
Ovojn timer r = new Ovojn timer(new Cas(10, 35));  
Cas c = (Cas) r.vrni();
```

Slabosti razreda Ovojnik

- poznati moramo tip objekta, ki ga hrani objekt tipa Ovojnik
- zaradi prevajalnika je potrebna pretvorba tipa
- napačna pretvorba sproži izjemo

```
Ovojnik p = new Ovojnik("dober dan");  
Integer n = (Integer) p.vrni(); // ClassCastException
```

- raje bi imeli napako pri prevajanju

Generični razred

- razred, parametriziran z enim ali več **referenčnimi** tipi
- primer razreda, parametriziranega s tremi tipi

```
public class Primer<T, U, V> {  
    // lahko uporabljamo tipe T, U in V  
    ...  
}
```

- pri uporabi razreda navedemo konkretne referenčne tipe

```
Primer<String, Integer, Cas> p =  
    new Primer<String, Integer, Cas>();
```

- tip T se nadomesti s tipom String
- tip U se nadomesti s tipom Integer
- tip V se nadomesti s tipom Cas

Terminologija

```
public class Primer<T, U, V> {  
    ...  
}
```

- T, U, V: **tipni parametri**
- Primer: **generični razred** (in tudi **generični tip**)

```
Primer<String, Integer, Cas> p =  
    new Primer<String, Integer, Cas>();
```

- String, Integer, Cas: **tipni argumenti**

Generični razred Ovojnik

```
public class Ovojnik<T> {  
    private T a;  
    public Ovojnik(T a) {  
        this.a = a;  
    }  
    public T vrni() {  
        return this.a;  
    }  
}
```

```
Ovojnik<String> p = new Ovojnik<String>("dober dan");  
String s = p.vrni();    // pretvorba tipa ni potrebna!
```

```
Ovojnik<Integer> q = new Ovojnik<Integer>(42);  
Integer n = q.vrni();
```

```
Ovojnik<Cas> r = new Ovojnik<Cas>(new Cas(10, 35));  
Cas c = r.vrni();
```

Sintaktična olajšava

- namesto

```
R<Tip1, Tip2, ...> r = new R<Tip1, Tip2, ...>(...);
```

lahko pišemo

```
R<Tip1, Tip2, ...> r = new R<>(...);
```


Generiki pri prevajanju in izvajanju

- če uporabljamo generike, nas prevajalnik varuje pred neskladjem tipov

```
Ovojniki<String> p = new Ovojniki<>(42);  
// napaka pri prevajanju
```

```
Ovojniki<String> p = new Ovojniki<>("tralala");  
Integer n = p.vrni(); // napaka pri prevajanju
```

Generiki pri prevajanju in izvajanju

- izvajalnik generikov sploh ne vidi
- prevajalnik jih nadomesti s tipom `Object` in pretvorbami tipa

```
public class Ovojnik<T> {  
    private T a;  
  
    public Ovojnik(T a) {  
        this.a = a;  
    }  
  
    public T vrni() {  
        return this.a;  
    }  
}
```

```
Ovojnik<String> p =  
    new Ovojnik<>("abc");  
String s = p.vrni();
```

→

```
public class Ovojnik {  
    private Object a;  
  
    public Ovojnik(Object a) {  
        this.a = a;  
    }  
  
    public Object vrni() {  
        return this.a;  
    }  
}
```

```
Ovojnik p =  
    new Ovojnik("abc");  
String s = (String) p.vrni();
```

Generična metoda

- metoda, parametrizirana z enim ali več referenčnimi tipi
- splošna oblika glave
določila `<tipniParametri> izhodniTip ime(parametri)`
- klic statične generične metode
Razred . `<tipniArgumenti>ime(parametri)`
- klic nestatične generične metode
objekt . `<tipniArgumenti>ime(parametri)`
- tipne argumente lahko praviloma izpustimo
- prevajalnik jih skoraj vedno lahko določi sam

Primer

- metoda za iskanje indeksa elementa v tabeli poljubnega referenčnega tipa

```
public class Iskanje {  
  
    public static <T> int poisci(T[] tabela, T iskani) {  
        for (int i = 0; i < tabela.length; i++) {  
            if (tabela[i].equals(iskani)) {  
                return i;  
            }  
        }  
        return -1;  
    }  
}
```

Primer

- klic metode

```
Integer[] stevila = {100, 50, 170, 80, 20, 130, 160, 40, 60, 90};  
  
// polna oblika  
System.out.println(Iskanje.<Integer>poisci(stevila, 20));  
  
// kratka oblika (zadošča v veliki večini primerov)  
System.out.println(poisci(stevila, 20));  
  
Cas[] casi = {new Cas(10, 20), new Cas(20, 10), new Cas(15, 45)};  
System.out.println(Iskanje.<Cas>poisci(casi, new Cas(15, 45)));  
System.out.println(poisci(casi, new Cas(13, 50)));
```

Določili extends in super

- lahko določimo, da mora biti tipni parameter T podtip (`extends`) ali nadtip (`super`) nekega drugega tipa R
- v obeh primerih je T lahko tudi enak R
- `super` lahko uporabljamo samo v generičnih metodah, `extends` pa tudi v generičnih razredih

```
public class StevilskiOvojniki<T extends Number> {  
    ...  
}
```

- tip T lahko nadomestimo le s tipom `Number` ali njegovim podtipom
- nad objekti tipa T lahko uporabljamo metode razreda `Number`

Določili extends in super

- primerjava objektov tipa StevilskiOvojniki, parametriziranih z istim podtipom tipa Number:

```
public class StevilskiOvojniki<T extends Number> {  
    ...  
    public boolean jeVecjiKot(StevilskiOvojniki<T> drugi) {  
        return this.a.doubleValue() > drugi.a.doubleValue();  
    }  
}
```

- primerjava objektov, parametriziranih z različnima podtipoma tipa Number

```
public class StevilskiOvojniki<T extends Number> {  
    ...  
    public <U extends Number> boolean jeVecjiKot(  
        StevilskiOvojniki<U> drugi) {  
        return this.a.doubleValue() > drugi.a.doubleValue();  
    }  
}
```

Omejitve

- naj bo T tipni parameter generičnega razreda
- objektov tipa T ni mogoče ustvarjati

```
T obj = new T(...); // napaka pri prevajanju
```

- tabelo tipa T[] sicer lahko ustvarimo, a tudi ne brez »ovinka«

```
T[] obj = new T[10]; // napaka pri prevajanju
...
T[] obj = (T[]) new Object[10]; // opozorilo
...
@SuppressWarnings("unchecked")
T[] obj = (T[]) new Object[10]; // OK
```


Generični razred Vektor

```
public class Vektor<T> {  
    private T[] elementi;  
    ...  
  
    @SuppressWarnings("unchecked")  
    public Vektor(int kapaciteta) {  
        this.elementi = (T[]) new Object[kapaciteta];  
        this.stElementov = 0;  
    }  
  
    public T vrni(int indeks) {  
        return this.elementi[indeks];  
    }  
  
    public void nastavi(int indeks, T vrednost) {  
        this.elementi[indeks] = vrednost;  
    }  
    ...  
}
```

Generični razred Slovar

```
public class Slovar<K, V> {  
  
    private static class Vozlisce<K, V> { // tipni parametri so vidni  
        K kljuc;                          // samo v nestatičnih elementih  
        V vrednost;                       // (vsak objekt ima svoj  
        Vozlisce<K, V> naslednje;         // tipni argument)  
        ...  
    }  
  
    private Vozlisce<K, V>[] podatki;  
    private int stParov;  
  
    @SuppressWarnings("unchecked")  
    public Slovar(int velikostTabele) {  
        this.podatki = (Vozlisce<K, V>[]) new Vozlisce[velikostTabele];  
        this.stParov = 0;  
    }  
  
    public void shrani(K kljuc, V vrednost) { ... }  
    public V vrni(K kljuc) { ... }  
    ...  
}
```