Generički tipovi

Programski jezici II

Generički tipovi

- *compile-time bug*-ovi
- *run-time bug*-ovi
- generički tipovi daju stabilnost – omogućavaju detekciju određenih grešaka za vrijeme kompajliranja
- *Java Collections Framework* – koristi generičke tipove

Generički tipovi

- prije:

```
public class Holder{
  private Object object;
  public void set(Object object) {
    this.object = object;
  }
  public Object get() {
    return object;
  }
}

public class Test {
   public static void main(String[] args) {
     Holder integerHolder = new Holder();
     integerHolder.set("10");
     Integer otherInteger = (Integer) integerHolder.get();
     System.out.println(otherInteger);
   }
}
```

java.lang.ClassCastException

Generički tipovi

- generički tip je referencni tip u čijoj se deklaraciji iza naziva tipa nalazi lista parametara
- ovi parametri se nazivaju i tipski parametri, promjenljive tipa ili tipske promjenljive
- ovi tipovi moraju biti poznati u vrijeme kompajliranja kako bi se generički tip mogao koristiti

Generički tipovi

- sad:

```
public class Holder<T>{
  private T t;
  public void set(T t) {
    this.t = t;
  }
  public T get() {
    return t;
  }
}
```

deklaracija generičkog tipa

tipska promjenljiva

- T – tipska promjenljiva:
  - može se koristiti bilo gdje u klasi. Ista tehnika se može primijeniti i na interfejse.
  - može biti bilo koji klasni tip,
  - može biti bilo koji tip interfejsa,
  - ne može biti primitivni tip podataka.
- Konvencija davanja imena – jedno veliko slovo:
  - E – Element, K – Key, N – Number, T – Type, V – Value, …

Generički tipovi

```
Holder<Integer> integerHolder = new
  Holder<Integer>();

public class Test {
   public static void main(String[] args) {
     Holder<Integer> integerHolder = new
   Holder<Integer>();
     integerHolder.set("10");
     Integer otherInteger =
   integerHolder.get();
     System.out.println(otherInteger);
   }
}
```

- pokušaj ubacivanja nekompatibilnog tipa u Holder - greška

Generički tipovi

- tipskoj promjenljivoj T može se pristupiti iz nestatičkog konteksta
- tipskoj promjenljivoj T ne može se pristupiti iz statičkog konteksta
- razlog ovog je što je tipska promjenljiva nestatička i ne može se koristiti u statičkom kontekstu iz istog razloga kao i bilo koja druga nestatička promjenljiva
- generički tipovi obuhvataju i generičke interfejse

```
public interface GenericHolderInterface<T> {
        public void set(T object);
        public T get();
}
```

- deklaracija generičkih interfejsa se ne razlikuje od deklaracije interfejsa, osim uvođenja tipske promjenljive

Parametrizovani tipovi

- parametrizovani tip predstavlja korištenje generičkog tipa gdje su tipske promjenljive zamijenjene parametrima stvarnog tipa
- ovaj koncept sličan je deklaraciji i korištenju metoda
- da bi koristili metodu moramo, pri njenom pozivu, proslijediti stvarne parametre
- isto tako, da bi instancirali generički tip, potrebno je da mu proslijedimo parametre stvarnog tipa
- deklarisanje referenci i kreiranje objekata parametrizovanih tipova, kao i pozivi metoda nad ovim objektima, slično je kao kod negeneričkih klasa

Parametrizovani tipovi

```
public static void main(String[] args) {
	GenericHolder<Integer> integerHolder = new
GenericHolder<Integer>();
	integerHolder.set(10);
	System.out.println(integerHolder.print());
	Integer otherInteger = integerHolder.get();
	System.out.println(otherInteger);
}
```

- instanciranja generičkog tipa GenericHolder kojem se kao parametar stvarnog tipa prosljeđuje Integer
- ovako se tipska promjenljiva T mijenja stvarnim tipom Integer – nastaje parametrizovani tip GenericHolder<Integer>
- referenca integerHolder može referencirati samo objekte parametrizovanog tipa GenericHolder<Integer>
- pri pozivima set i get metoda stvarni tipovi argumenta i povratnog tipa određuju se na osnovu tipa reference nad kojom se metode pozivaju - kako je referenca integerHolder tipa GenericHolder<Integer>, jasno je da metoda set kao argument prima, a metoda get kao rezultat vraća objekat tipa Integer - kako je ovo poznato kompajleru (u trenutku prevođenja programa), nisu potrebna eksplicitna kastovanja tipova.

Parametrizovani tipovi

- Dijamant (The Diamond)

```
GenericHolder<Integer> intHolder = new GenericHolder<>();
```

- Tipska promjenljiva se može zamijeniti i parametrizovanim tipom

```
GenericHolder<GenericBase<Integer>> genericHolder = new
GenericHolder<GenericBase<Integer>>();
```

- Raw tipovi
- Mnoge Java klase nisu bile generičke prije verzije JDK 5
- Pri korišćenju raw tipova – dobija se *pre-generics* ponašanje – generički tip radi s Object-ima
- Zbog kompatibilnosti unazad, dozvoljeno je dodijeliti parametrizovani tip raw tipu:
  - GenericHolder<String> holder1 = new GenericHolder<>();
  - GenericHolder rawHolder = holder1; // OK
- Kod dodjele raw tipa parametrizovanom tipu, dobija se upozorenje (warning):
  - GenericHolder rawHolder2 = new GenericHolder();
  - GenericHolder<Integer> intHolder2 = rawHolder2;
- Upozorenje se dobija i kada se koristi raw tip za poziv generičke metode definisane u odgovarajućem generičkom tipu:
  - Box<String> stringBox = new Box<>();
  - Box rawBox = stringBox;
  - rawBox.set(8); // warning: unchecked invocation to set(T)

- **Potrebno je izbjegavati korišćenje raw tipova!!!**

Parametrizovani tipovi

- zamjenom tipskih promjenljivih parametrima različitih stvarnih tipova nastaju različiti parametrizovani tipovi – između njih ne postoji veza
  - tako su parametrizovani tipovi GenericHolder<Integer> i GenericHolder<String> različiti i između njih ne postoji veza
- kompajler će prijaviti svaki pokušaj nepravilne upotrebe parametrizovanog tipa
  - na ovaj način potencijalne greške mogu biti detektovane ranije, za vrijeme kompajliranja
  - upotreba eksplicitnog kastovanja u izvornom kodu je minimalizovana

Parametrizovani tipovi

- parametrizacija generičkih interfejsa vrši se na isti način kao i parametrizacija generičkih klasa

```
class GenericHolder2<T> implements GenericHolderInterface<T>
{
	private T object;

	public T get() {
		return object;
	}

	public void set(T object) {
		this.object = object;
	}

	public static void main(String[] args) {
		GenericHolderInterface<Integer> integerHolder =
new GenericHolder2<Integer>();                          // 1
		integerHolder.set(10);
		Integer otherInteger = integerHolder.get();
		System.out.println(otherInteger);
	}
}
```

- različita upotreba konstrukcije <T> u deklaraciji klase GenericHolder2
  - prvo pojavljivanje konstrukcije <T> označava tipsku promjenljivu
  - drugo pojavljivanje konstrukcije <T> parametrizuje generički interfejs istom tipskom promjenljivom (T)

Nasljeđivanje generičkih tipova

- generički tip koji nije deklarisan kao final može biti nasljeđen

```
public class GenericBase<T> {
	private T baseVariable;

	public T get() {
		return baseVariable;
	}

	public void set(T t){
		baseVariable = t;
	}
}

class GenericSub<T> extends GenericBase<T>{
	private T subVariable;

	public T get2(){
		return subVariable;
	}
}
```

- kompajler provjerava da li tipske promjenljive roditeljske klase navedene u extends klauzuli mogu biti razriješene – u prethodnom primjeru tipska promjenljiva T generičke klase GenericSub (klase nasljednice) koristi se i kao parametar klase GenericBase (osnovna klasa) – ova veza obezbjeđuje da će tipske promjenljive biti zamijenjene istim stvarnim tipom u obje klase

Nasljeđivanje generičkih tipova

```
public class GenericTest{
	public static void main(String args[]){
		GenericBase<Integer> base = new
GenericBase<Integer>();
		GenericBase<Integer> sub = new
GenericSub<Integer>();
//		GenericBase<Integer> sub2 = new
GenericSub<String>();
		base.set(3);
		sub.set(5);
		System.out.println(base.get());
		System.out.println(sub.get());
	}
}
```

Nasljeđivanje generičkih tipova

- ponekad je potrebno da se ograniče tipovi koji se mogu koristiti za parametrizaciju

```
class GenericNumberSub<T extends Number> extends
GenericBase<T>{
//...
}
```

- generička klasa GenericNumberSub nasljeđuje generičku klasu GenericBase postavljajući gornju granicu tipa koji može biti prosljeđen kao parametar
- parametrizacija ove generičke klase može se izvršiti prosljeđivanjem klase Number ili bilo koje klase koja nasljeđuje klasu Number
- ključna riječ extends se koristi u širem smislu, i obuhvata semantiku standardnog korištenja ključnih riječi extends (kod nasljeđivanja) i implements (kod implementacije interfejsa)
  - ako je potrebno dodatno navesti i jedan ili više interfejsa koje je potrebno implementirati potrebno je koristiti znak „&“ između naziva interfejsa ili između naziva klase i interfejsa
    - naziv klase mora doći prije naziva interfejsa

Nasljeđivanje generičkih tipova

- moguće je da generički tip naslijedi negenerički

```
public class GenericHolder3<T> extends Holder {

//...

}
```

- moguće je i da konkretan tip naslijedi parametrizovani tip

```
public class IntegerHolder extends GenericHolder<Integer> {
      public void add(Integer i){
            Integer temp = get();
            temp += i;
            set(temp);
      }
      public static void main (String args[]){
            IntegerHolder holder = new IntegerHolder();
            holder.set(3);
            holder.add(2);
            System.out.println(holder.get());
      }
}
```

- konkretna klasa ne može naslijediti generički tip

Generičke metode

- Generičke metode uvode svoje tipske promjenljive
  - slično deklaraciji generičkog tipa, ali vidljivost im je ograničena na metodu u kojoj su deklarisane
- Postoje statičke i nestatičke metode, kao i generički konstruktori
- Sintaksa:
  - generička metoda: tipska promjenljiva unutar <> koja se pojavljuje prije povratnog tipa metode

Wildcards

- U generičkom kodu, *wildcard* znak „?“ predstavlja nepoznati tip
- može se koristiti u različitim situacijama:
  - kao tip parametra, polja, lokalne varijable, a ponekad i kao povratni tip (mada se ovo ne preporučuje)

Wildcards

- *upper bounded wildcard*

```
public static void method(List<? extends Foo>
list) { /* ... */ }
```

- *unbounded wildcard*

```
public static void method(List<?> list) { /*
... */ }
```

- *lower bounded wildcard*

```
public static void method(List<? super Foo>
list) { /* ... */ }
```

Brisanje tipova

- kompajler prevodi generičku klasu tako da briše informacije o tipskim promjenljivim
- ovaj proces naziva se brisanje tipova (eng. *type erasure*)
- ako se program kompajlira bez *unchecked* upozorenja, kompajler garantuje da će sigurnost tipova za vrijeme izvršavanja biti obezbjeđena

Brisanje tipova

- generičke klase moguće je koristiti bez parametrizacije, kao negeneričke klase
- kompajler će prijaviti *unchecked* upozorenja ako korištenje generičke klase bez parametrizacije može rezultirati potencijalnim problemom za vrijeme izvršavanja
- ovakvo korištenje generičkih klasa je dozvoljeno zbog zadržavanja kompatibilnosti sa ranijim verzijama Jave, u kojim generičke klase nisu postojale, ali se ne preporučuje kod razvoja novih aplikacija

```
public class MixTest {
      public static void main(String[] args) {
            GenericHolder holder = new
GenericHolder<Integer>();
            holder.set("10");
//          Integer i = (Integer) holder.get();
            GenericHolder<Byte> holder2 = holder;
            Byte s = holder2.get();
            System.out.println(holder.get());
      }
}
```

- uvijek je moguće dodijeliti vrijednost reference parametrizovanog tipa referenci neparametrizovanog tipa, jer je on supertip parametrizovanog tipa
  - ova referenca može se koristiti na takav način da se naruši sigurnost tipova za vrijeme izvršavanja

Prednosti generičkih tipova

- „Snažnija“ provjera tipova za vrijeme kompajliranja
  - Ako je narušena sigurnost tipova kompajler će prijaviti grešku
- Izbjegavanje kastovanja
  - Bez korišćenja generičkih tipova:
    - List list = new ArrayList();
    - list.add("hello");
    - String s = (String) list.get(0);
  - Sa korišćenjem generičkih tipova:
    - List<String> list = new ArrayList<String>();
    - list.add("hello");
    - String s = list.get(0); // nema kastovanja
- Implementacija generičkih algoritama