Analýza a návrh informačných systémov II 6

objektovo orientovaný návrh

Peter Bednár

Kolekcie

Usporiadanie objektov

- Pre usporiadanie objektov je potrebné implementovať rozhranie `java.util.Comparable`, ktoré definuje iba jednu metódu `int compareTo(Object obj)`
- Metóda `ob1.compareTo(obj2)` vracia:
  - záporné číslo ak má byť `obj1` umiestnený pred `obj2` (tzn. platí `obj1 < obj2`)
  - 0 ak sa objekty rovnajú (tzn. malo by platiť `obj1.equals(obj2) == true`)
  - kladné číslo ak má byť `obj1` umiestnený za `obj2` (tzn. platí `obj1 > obj2`)

Usporiadanie objektov - príklad

```
public class Adresa implements Comparable {

    private String ulica;
    private int cislo;
    ...

    @Override
    public int copareTo(Object obj) {
        Adresa adr = (Adresa)obj;
        int cmp = ulica.compareTo(adr.ulica);  // najprv porovnáme ulice
        if (cmp != 0) {   // ak sa ulice nerovnajú
            return cmp;  // primárne usporiadame adresy podľa ulice
        } else {
            // ak sa ulice rovnajú, usporiadame ich podľa čísla
            return cislo < adr.cislo ? -1 : cislo > adr.cislo ? 1 : 0;
        }
    }

}
```

Objekty ktoré chceme usporiadať by mali implementovať rozhranie java.util.Comparable

Usporiadané množiny

- `java.util.TreeSet`
  - Prvky sú usporiadané v danom poradí
  - Objekty musia implementovať rozhranie `Comparable` aby sa dali usporiadať
  - Implementuje rozšírené rozhranie `SortedSet`
    - `Object first()`
    - `Object last()`
    - `SortedSet headSet(Object to)`
    - `SortedSet tailSet(Object from)`

Usporiadané množiny – príklad (1)

```
SortedSet set1 = new TreeSet();
set1.add("jablko");
set1.add("hruška");
set1.add("broskyňa");

System.out.println(set1.first());
```

Reťazce implementujú rozhranie Comparable a usporiadané sú podľa lexikografického poradia

```
SortedSet set2 = set1.tailSet("jablko");
for (Object s : set2) {
    System.out.println(s);
}
set2.clear();
System.out.println(se1.size());    // po odstránení prvkov set2
                                   // zostane len broskyňa
```

set2 má prvky jablko, hruška tailSet/headSet vráti pohľad na pôvodnú kolekciu, tzn. keď modifikujeme set2 zmení sa aj set1

Usporiadané množiny – príklad (2)

```
SortedSet set1 = new TreeSet();

set1.add(new Adresa("Letná", 9));
set1.add(new Adresa("Letná", 1));
set1.add(new Adresa("Vysokoškolská", 4));

for (Object obj : set1) {
    System.out.println(obj);    // štandardne sa pre výpis
}                               // objektov zavolá toString
```

Objekty sa usporiadajú štandardne podľa metódy compareTo rozhrania Comparable

Mapy (1)

- Ukladajú dvojice kľúč-hodnota
- Kľúč je jedinečný – v mape môže byť len jeden kľúč s jednou hodnotou
- Mapy neimplementujú rozhranie `java.util.Collection` (keďže prvky nie sú jednotlivé objekty ale dvojice kľúč-hodnota)

Mapy (2)

- Viacero implementácii
- `java.util.HashMap`
  - Využíva hešovanie
  - Pre vlastnú implementáciu `equals()` je potrebné pridať vlastnú implementáciu metódy `int hashCode()`
- `java.util.TreeMap`
  - Kľúče sú usporiadané v danom poradí
  - Kľúče musia implementovať rozhranie `Comparable` aby sa dali usporiadať

Mapy – metódy (1)

Metóda	Popis
`Object put(Object kluc, Object hodnota)`	Nastaví hodnotu pre daný kľúč
`Object get(Object kluc)`	Vráti hodnotu asociovajú s daným kľúčom, alebo null ak mapa neobsahuje daný kľúč
`Object remove(Object kluc)`	Odstráni kľúč a jeho hodnotu z mapy
`void clear()`	Odstráni všetky prvky
`booean contains(Object key)`	Vráti `true` ak mapa obsahuje mapovanie pre daný kľúč
`boolean isEmpty()`	Vráti true ak je kolekcia prázdna
`int size()`	Vráti aktuálny počet dvojíc kľúč-hodnota

Mapy – metódy (2)

- `Set entrySet()`
  - Vráti množinu objektov typu `Map.Entry`, ktoré definujú metódy `Object getKey()` a `Object getValue()` pre prístup k asociovaným kľúčom a hodnotám
- `Set keySet()`
  - Vráti množinu kľúčov
- `Collection values()`
  - Vráti kolekciu hodnôt (hodnoty sa môžu opakovať, tzn. jedna hodnota môže byť priradené viacerým kľúčom)
- Všetky kolekcie sú pohľady (tzn. ak sa modifikujú upraví sa aj mapa)

Mapy – iterovanie

- Je možné iterovať cez kolekciu prvkov mapy ktorú vráti metóda `Set entrySet()`
- Podobne možno využiť aj `entrySet().iterator()`

```
for (Object obj : map.entrySet()) {
    Map.Entry prvok = (Map.Entry)obj;
    Object kluc = prvok.getKey();
    Object hodnota = prvok.getValue();
    ...
}
```

Množinu prvkov pre iterovanie vráti metóda entrySet()

Pretypujeme prvok na Map.Entry

Získame objekty kľúča a hodnoty

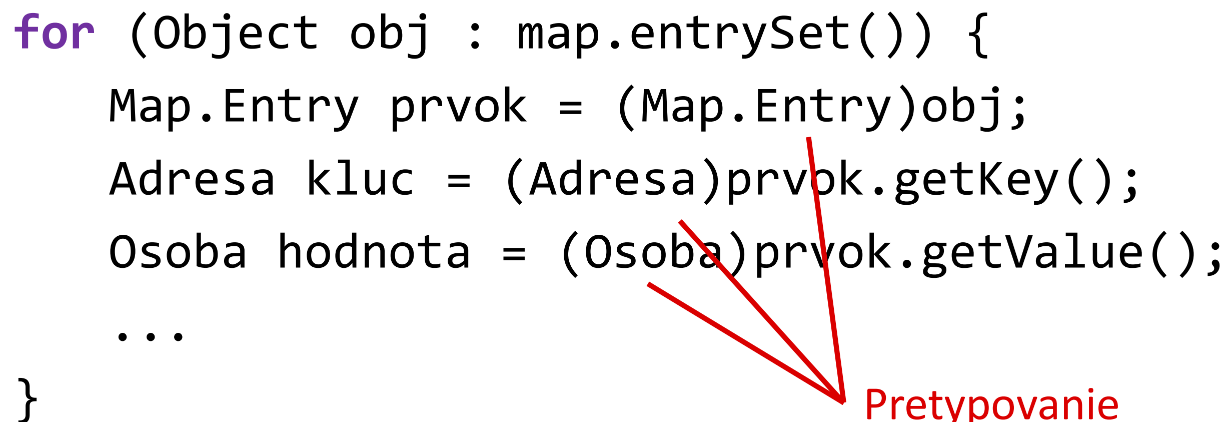
Mapy – príklad

```
Map mesiace = new HashMap();

mesiace.put("december", "zima");
mesiace.put("január", "zima");
mesiace.put("jún", "leto");
mesiace.put("október", null); // pre väčšinu implementácii
                              // kľúč aj hodnota môže byť null

String obdobie = (String)mesiace.get("január");
obdobie = (String)mesiace.get("február"); // vráti null

if (mesiace.contains("október")) {
    obdobie = (String)mesiace.get("október");
}
```

Generické typy

Generické typy (1)

- Štandardne, metódy kolekcií vracajú typ Object, ktorý je spoločný pre všetky objekty
  - Ak chceme ďalej pracovať s danou hodnotu, musíme ju pretypovať na požadovaný typ



```
for (Object obj : map.entrySet()) {
    Map.Entry prvok = (Map.Entry)obj;
    Adresa kluc = (Adresa)prvok.getKey();
    Osoba hodnota = (Osoba)prvok.getValue();
    ...
}
```

Generické typy (2)

- Do kolekcie štandardne môžeme pridať objekty rôznych typov a kompilátor nekontroluje, či kolekcia obsahuje iba správne typy – môže dôjsť k chybe pri pretypovaní

```
List programatori = new LinkedList();
programatori.add(new Programator("Anna"));
programatori.add(new Programator("Peter"));
programatori.add(new Manager("Ján"));

for (Object obj : programatori) {
    String meno = ((Programator) obj).getMeno();
}
```

Do kolekcie štandardne môžete pridať objekty rôznych typov

CHYBA – nemožno pretypovať typ Manager na Programator

Generické typy (3)

- Generické triedy a rozhrania majú pri svojej definícii uvedený parameter, ktorý popisuje typ ich premenných a metód
- Generický typ danej triedy/rozhrania potom definujeme uvedením konkrétneho typu pre parameter
- Zápis parametrov:

  Názov triedy <Zoznam parametrov>
- Napr.:

```
List<String>
HashMap<String, Adresa>
```

Generické zoznamy - príklad

Medzi < > je uvedený typ prvkov v zozname

```
List<String> retazce = new LinkedList<String>();
retazce.add("jedna");
String s = retazce.get(0);

List<Adresa> adresy = new ArrayList<Adresa>();
adresy.add(new Adresa("Letná", 9));
adresy.add("Letná 9");

for (Adresa adresa : adresy) {
    System.out.println(adresa.getCislo());
}
```

Metóda get už priamo pracia reťazec a nemusíme hodnotu pretypovať

Tu bude kompitátor hlásiť chybu, pretože sa pokúšame pridať reťazec do kolekcie adries

Aj pri iterácii už nemusíme pretypovať hodnoty a premennú definujeme priamo ako adresu

Generické mapy - príklad

Pre mapy je potrebné uviesť dva parametre – typ kľúčov a typ hodnôt

```
Map<String, Adresa> adresy = new HashMap<String, Adresa>();

adresy.put("Anna", new Adresa("Letná", 9));
adresy.put("Peter", new Adresa("Zimná", 6));

for (Map.Entry<String, Adresa> prvok : adresy.entrySet()) {
    String meno = prvok.getKey();
    Adresa adresa = prvok.getValue();
}
```

Pri iterovaní aj Map.Entry definujeme ako generický typ, takže už nemusíme pretypovať kľúč ani hodnotu

Definovanie generických tried a rozhraní

- Pri definícii triedy/rozhrania uvedieme zoznam generických parametrov

```
class Názov <Zoznam gen. parametrov> extends ... {
    ...
}
```

- Parameter potom môžeme použiť ako typ premennej, alebo typ argumentov a návratovej hodnoty metód

Definovanie generických tried a rozhraní – príklad (1)

```
public class Uloha {
    ...
}

public interface ClenTimu {
    void vykonajUlohu(Uloha uloha);
}

public class Programator implements ClenTimu {
    ...
}

public class Tester implements ClenTimu {
    ...
}
```

Definujeme si objekty pre popis úloh a členov tímu. Tieto typy nie sú generické

Definovanie generických tried a rozhraní – príklad (2)

```
public class ProjektovyTim<T> {
    private T lider;
    private Set<T> clenovia;

    public ProjektovyTim() {
        this(null);
    }
    public ProjektovyTim(T lider) {
        this.lider = lider;
        this.clenovia = new HashSet<T>();
    }
    public T getLider() {
        return lider;
    }
    public Set<T> getClenovia() {
        return clenovia;
    }
}
```

Definujeme generický parameter (názov si môžete zvoliť – odporúčajú sa veľké písmená)

Parameter môžete potom použiť ako typ premenných alebo typ argumentov a návratovej hodnoty metód (môžete ho použiť aj pri parametrizovaní iných generických tried a rozhraní)

Definovanie generických tried a rozhraní – príklad (3)

```
Programator jan = new Programator();
Programator anna = new Programator();

ProjektovyTim<Programator> programatori = new
    ProjektovyTim<Programator>(jan);
programatori.getClenovia().add(anna);
Programator lider = programatori.getLider();

Tester zuzana = new Tester();
Tester peter = new Tester();

ProjektovyTim<Tester> testeri = new ProjektovyTim<Tester>(zuzana);
testeri.getClenovia().add(peter);

for (Tester tester : testeri.getClenovia()) {
    tester.vykonajUlohu(new Uloha());
}
```

Definujeme špecifické typy tímov - pre programátorov a pre testerov

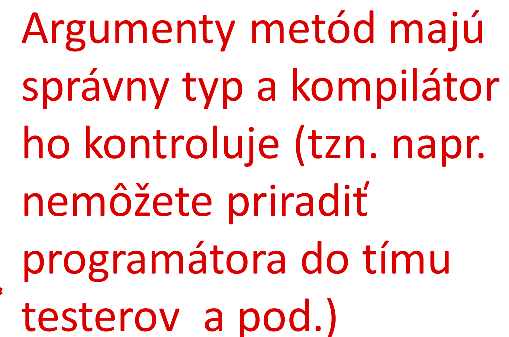
Definovanie generických tried a rozhraní – príklad (4)

```
Programator jan = new Programator();
Programator anna = new Programator();

ProjektovyTim<Programator> programatori = new
    ProjektovyTim<Programator>(jan);
programatori.getClenovia().add(anna);
Programator lider = programatori.getLider();

Tester zuzana = new Tester();
Tester peter = new Tester();

ProjektovyTim<Tester> testeri = new ProjektovyTim<Tester>(zuzana);
testeri.getClenovia().add(peter);

for (Tester tester : testeri.getClenovia()) {
    tester.vykonajUlohu(new Uloha());
}
```

Návratové hodnoty majú správny typ, takže objekty už nemusíme nijak pretypovať

Definovanie generických tried a rozhraní – príklad (5)

```
Programator jan = new Programator();
Programator anna = new Programator();

ProjektovyTim<Programator> programatori = new
    ProjektovyTim<Programator>(jan);
programatori.getClenovia().add(anna);
Programator lider = programatori.getLider();

Tester zuzana = new Tester();
Tester peter = new Tester();

ProjektovyTim<Tester> testeri = new ProjektovyTim<Tester>(zuzana);
testeri.getClenovia().add(peter);

for (Tester tester : testeri.getClenovia()) {
    tester.vykonajUlohu(new Uloha());
}
```



Ohraničenie typu

- Typ ktorý sa dosadí za generický parameter môžeme ohraničiť pomocou slova **extends**
- Typ môžeme ohraničiť triedou alebo rozhraním, napr.:

```
public class ProjektovyTim<T extends ClenTimu> {
    ...
}
```

Za T môžeme dosadiť iba typ, ktorý implementuje rozhranie ClenTimu

```
...
ProjektovyTim<ClenTimu> tim1 = new ProjektovyTim<ClenTimu>();
ProjektovyTim<Tester> tim2 = new ProjektovyTim<Tester>();
ProjektovyTim<String> tim3 = new ProjektovyTim<String>();
```

CHYBA: String neimplementuje ClenTimu

Definovanie generických tried a rozhraní – príklad (6)

```
public class Uloha {
    ...
}
public class Programovanie extends Uloha {
    ...
}
public class Testovanie extends Uloha {
    ...
}

public interface ClenTimu<U extends Uloha, V> {
    V vykonajUlohu(U uloha);
}
```

Rozšírime hierarchiu úloh a pre člena tímu definujeme generický parameter pre typ úlohy a typ výstupu (výstup nie je ohraničený)

Definovanie generických tried a rozhraní – príklad (7)

```
public class Programator<V> implements ClenTimu<Programovanie, V> {
    ...
    @Override
    public V vykonajUlohu(Programovanie uloha) {
        ...
    }
}
public class Tester implements ClenTimu<Testovanie, Boolean> {
    ...
    @Override
    public Boolean vykonajUlohu(Testovanie uloha) {
        ...
    }
}
```

Programator je generická trieda a môžeme definovať rôzne typy podľa výstupu

Tester má dosadené všetky generické parametre rozhrania ClenTimu (výstup je Boolean)

Atomické typy ako objekty

Atomické typy ako objekty (1)

- Niekedy chceme využiť kolekcie aj s atomickými typmi ako sú napr. čísla alebo Boolovské hodnoty
- Java poskytuje triedy, ktoré "obalia" atomické hodnoty ako objekt, tieto typy potom môžete použiť aj pri definovaní generických kolekcií
- Java tak isto automaticky prevedie tieto objekty na atomické hodnoty

Atomické typy ako objekty (2)

Atomický typ	Trieda objektov
char	Character
int	Integer
short	Short
long	Long
float	Float
double	Double
boolean	Boolean

- Podobne ako pri reťazcoch (trieda String) tieto typy sú vždy dostupné a nemusíte ich importovať

Atomické typy ako objekty – príklad

```
Integer cislo = new Integer(3);
Boolean pravda = new Boolean(true);

int cislo1 = cislo;
Integer cislo2 = cislo1;

List<Integer> cisla = new LinkedList<Integer>;
cisla.add(new Integer(1));
cisla.add(2);
cisla.add(3);

int suma = 0;
for (Integer c : cisla) {
    suma += c;
}
```

Z atomických hodnôt môžeme urobiť objekty

Kompilátor automaticky prevedie objekt na hodnotu a naopak (tzv. "autoboxing")

Typ objektov môžeme použiť ako generický parameter napr. pri kolekciách

Pri iterácií prvkov zoznamu premennú definujeme ako objekt, ktorý sa automaticky prevedie na hodnotu

Zhrnutie

- Kolekcie
  - Usporiadanie objektov
  - Usporiadané množiny
  - Mapy
- Generické typy
- Atomické typy ako objekty