Analýza a návrh informačných systémov II 5

objektovo orientovaný návrh

Peter Bednár

Ošetrenie chýb - výnimky

Zachytenie výnimiek

- Je možné uviesť viacero klauzúl `catch` s rôznymi typmi
- Výnimka sa spracuje v prvom bloku pre ktorý platí, že vyvolaná výnimka je daného typu (vrátane ak je odvodená z daného typu)
- Najprv by mal byť uvedený najšpecifickejší typ, potom všeobecné

Zachytenie výnimiek – príklad

```
public static void main(String args[]) {
    try {
        double hodnota;
        ...
        if (hodnota == 0) {
            throw new IllegalStateException();
        }
        if (hodnota < 0) {
            throw new IllegalArgumentException();
        }
    } catch (IllegalArgumentException chyba) {
        System.out.println(chyba);
    } catch (Exception chyba) {
        System.out.println(chyba);
    }
}
```

Tu sa zachytí a spracuje výnimka typu IllegalArgumentException

Tu sa zachytí výnimka typu IllegalStateException, keďže všetky výnimky sú odvodené od Exception

Kontrolované vs. nekontrolované výnimky

- Nekontrolované – kompilátor nekontroluje ich vyvolanie, alebo spracovanie
  - Všetky výnimky odvodené z triedy `RuntimeException`
- Kontrolované – kompilátor vyžaduje, aby programátor buď chybu ošetril alebo deklaroval, že daná metóda môže chybu vyvolať
  - Ak metóda neošetruje daný typ kontrolovanej výnimky, musí ho uviesť v zozname za kľúčovými slovom **`throws`**

Vlastné typy chýb

- Je možné si definovať vlastné typy chýb
- Trieda odvodená z triedy `Exception` (pre kontrolované výnimky) resp. `RuntimeException` (pre nekontrolované výnimky)
  - Nová trieda by mala implementovať bezparametrický konštruktor a konštruktor so správou
  - Môže definovať ľubovoľné ďalšie metódy a členské premenné

Vlastné typy chýb – príklad (1)

```
public class MotorException extends Exception {

      protected Motor motor;

      public MotorException() {
            this(null, null);
      }

      public MotorException(String sprava) {
            this(sprava, null);
      }

      public MotorException(String sprava, Motor motor) {
            super(sprava);
            this.motor = motor;
      }

      public Motor getMotor() {
            return motor;
      }

}
```

trieda MotorException je odvodená od Exception, tzn. ide o kontrolovanú výnimku

Definujeme členskú premennú, ktorá bude odkazovať na motor pri ktorom došlo k chybe a konštruktory

Vlastné typy chýb – príklad (2)

```
public class Motor {

     protected int otacky;
     ...

     public nastartuj() throws MotorException {
          if (otacky == 0) {
               otacky = 2000;
          } else {
               throw new MotorException("motor už beží", this);
          }
     }

     public nastavOtacky(int otacky) throws MotorException {
          if (otacky < 0 || otacky > 7000) {
               throw new IllegalArgumentException();
          }
          if (this.otacky == 0) {
               throw new MotorException("motor nebeží", this);
          }
          this.otacky = otacky;
     }
     ...
}
```

Keďže metódy vyvolajú a nespracujú kontrolovanú výnimku, musia ju uviesť za kľúčovým slovom throws

IllegalArgumentException je nekontrolovaná výnimka takže nemusí byť uvedená v zozname throws (ale je lepšie ak ju tam uvedieme kvôli čitateľnosti kódu, kompilátor to však nebude kontrolovať)

Vlastné typy chýb – príklad (3)

```
public class Motor {

     protected int otacky;
     ...

     public nastavOtacky(int otacky) throws MotorException {
          if (otacky < 0 || otacky > 7000) {
               throw new IllegalArgumentException();
          }
          if (this.otacky == 0) {
               throw new MotorException("motor nebeží", this);
          }
          this.otacky = otacky;
     }

     public nastavPlyn(double plyn) throws MotorException {
          nastavOtacky(plyn * 500 + 2000);
     }

}
```

Keďže metóda nastavPlyn volá metódu nastavOtacky ktorá môže vyvolať nekontrolovanú výnimku a neošetrí ju, musí tiež deklarovať výnimku v zozname throws

Vlastné typy chýb – príklad (4)

```
public static void main(String args[]) {
     try {
          Motor motor1 = new Motor();
          Motor motor2 = new Motor();

          motor1.nastartuj();
          motor2.nastavOtacky(2000);
     } catch (MotorException chyba) {
          System.out.print(chyba.getMessage());
     }
}
```

Keďže sme ošetrili kontrolovanú výnimku MotorException pri volaní nastartuj() a nastavOtacky(), samotná metóda main už nedefinuje zoznam throws

Vlastné typy chýb – príklad (4)

```
public static void main(String args[]) throws MotorException {
     try {
          Motor motor1 = new Motor();
          Motor motor2 = new Motor();

          motor1.nastartuj();
          motor2.nastavOtacky(2000);
     } catch (MotorException chyba) {
          System.out.print(chyba.getMessage());
          chyba.getMotor().nastartuj();
     }
}
```

Ak upravíme program tak že sa znova môže vyvolať kontrolovaná výnimka pri obsluhe chýb, musíme ju uviesť v zozname throws

Vlastné typy chýb – príklad (4)

```
public static void main(String args[]) {
     try {
          Motor motor1 = new Motor();
          Motor motor2 = new Motor();

          motor1.nastartuj();
          motor2.nastavOtacky(2000);
     } catch (MotorException chyba) {
          System.out.print(chyba.getMessage());
          try {
               chyba.getMotor().nastartuj();
          } catch (MotorException chyba) {
               System.out.print(chyba.getMessage());
          }
     }
}
```

Príkazy try/catch môžu byť vnorené, tzn. v tomto prípade sú ošetrené znovu všetky kontrolované výnimky a main neuvádza žiadne v zozname throws

Vlastné typy chýb – príklad (5)

```
public static void main(String args[]) throws MotorException {
     try {
          Motor motor1 = new Motor();
          Motor motor2 = new Motor();

          motor1.nastartuj();
          motor2.nastavOtacky(2000);
     } catch (MotorException chyba) {
          System.out.print(chyba.getMessage());
          throw chyba;
     }
}
```

Chybu môžeme iba čiastočne ošetriť a znova vyvolať príkazom throw

Ošetrenie viacero typov v jednom bloku

- Namiesto viacerých klauzúl `catch` s rovnakou obsluhou:

```
try {
     Motor motor1 = new Motor();
     ...
} catch (MotorException chyba) {
     System.out.print(chyba.getMessage());
} catch (IllegalArgumentException chyba) {
     System.out.print(chyba.getMessage());
}
```

- Môžeme ich spojiť cez | v jednej klauzule:

```
try {
     Motor motor1 = new Motor();
     ...
} catch (IllegalArgumentException | MotorException chyba) {
     System.out.print(chyba.getMessage());
}
```

Výnimky - zhrnutie

- Vyvolanie cez príkaz `throw`
  - Výnimky je možné znovu vyvolať pri obsluhe
- Ošetrenie cez `try/catch/finally`
  - Naraz je možné spojiť obsluhu viacero typov cez |
- Príkazy môžu byť vnorené
- Je možné si definovať vlastné typy chýb odvodené od:
  - `Exception` – kontrolované výnimky
  - `RuntimeException` – nekontrolované výnimky
- Ak metóda môže vyvolať kontrolovanú výnimku, musí ju uviesť v zozname `throws`

Kolekcie

Kolekcie

- Kolekcie slúžia na vytvorenie množiny objektov, resp. hodnôt
- Rozlišujú sa tri hlavné typy kolekcií, ktoré sú definované ako rozhrania v balíku `java.util`
  - Zoznamy – `java.util.List`
  - Množiny – `java.util.Set`
  - Mapy – `java.util.Map`

Kolekcie - prehľad

Zoznam	Množina	Mapa
`java.util.List`	`java.util.Set`	`java.util.Map`
Prístup podľa indexovania (podobne ako polia) Môžu obsahovať tie isté hodnoty	Jedna hodnota sa môže vyskytovať iba raz	Prístup podľa kľúča Každý kľúč môže byť v mape iba raz a môže mať jednu hodnotu
`java.util.ArrayList` `java.util.LinkedList`	`java.util.HashSet` `java.util.TreeSet`	`java.util.HashMap` `java.util.TreeMap`

Kolekcie – spoločné metódy

- Definované v spoločnom rozhraní `java.util.Collection`

Metóda	**Popis**
`boolean add(Object e)`	Pridá nový prvok do kolekcie
`boolean addAll(Collection c)`	Pridá do kolekcie všetky prvky z druhej kolekcie
`boolean remove(Object o)`	Odstráni prvok z kolekcie
`void clear()`	Odstráni všetky prvky
`booean contains(Object o)`	Vráti `true` ak kolekcia obsahuje daný prvok
`boolean isEmpty()`	Vráti true ak je kolekcia prázdna
`int size()`	Vráti aktuálny počet prvkov

Kolekcie – iterovanie (1)

- Pre iterovanie prvkov sa používajú iterátory – objekty typu `java.util.Iterator`

```
Iterator itr = kolekcia.iterator();
while (itr.hasNext()) {
    Object obj = itr.next();
    if (obj.equals(element)) {
        itr.remove();
    }
}
```

Získame iterátor kolekcie

Metóda iterátora boolean hasNext() vráti true ak existuje ďalší prvok

Metóda Object next() vráti aktuálny prvok a posunie iterátor na ďalší

Pomocou iterátora môžeme aktuálny prvok aj odstrániť

Kolekcie – iterovanie (2)

- Ak sa pri iterovaní kolekcia nemení (tzn. neodstraňujete prvky), môžete použiť skrátený zápis príkazu for:

```
for (Object obj : kolekcia) {
    System.out.println(obj.toString());
}
```

Do obj sa budú postupne priradzovať jednotlivé prvky kolekcie

Zoznamy (1)

- K prvkom je možné pristupovať podľa číselného indexu podobne ako pri poliach
  - `Object get(int index)`
  - `void add(int index, Object obj)`
  - `Object set(int index, Object obj)`
  - `Object remove(int index)`
- Na rozdiel od polí je možné do nich pridávať a odoberať nové prvky metódami `add(Object obj)` a `remove(Object obj)`

Zoznamy (2)

- Viacero tried, ktoré sa odlišujú vnútornou implementáciou
- `java.util.ArrayList`
  - Vnútorne sú prvky uložené v poly
  - Rýchli prístup na ľubovoľný index
  - Pomalšie pridávanie a odoberanie prvkov
- `java.util.LinkedList`
  - Implementácia obojsmerným zoznamom
  - Pomalší prístup k ľubovoľným prvkom (ale postupné iterovanie je rýchle)
  - Rýchlejšie pridávanie a odoberanie prvkov

Zoznamy – príklad (1)

```
List zoznam1 = new ArrayList();
zoznam1.add("jedna");
zoznam1.add("dva");
zoznam1.add(1, "jedna");

List zoznam2 = new LinkedList(zoznam1);
zoznam2.remove("jedna");
zoznam2.remove(0);

for (Object obj : zoznam2) {
    System.out.println(obj.toString());
}
```

Zoznam môže obsahovať viac krát tie isté hodnoty

Kolekcie majú zvyčajne konštruktor, ktorý umožňuje skopírovať prvky z inej kolekcie

Zoznamy – príklad (2)

```
List zoznam1 = new ArrayList();
zoznam1.add("jedna");
zoznam1.add("dva");
zoznam1.add(1, "jedna");

List zoznam2 = new LinkedList(zoznam1);
zoznam2.remove("jedna");
zoznam2.remove(0);

for (Object obj : zoznam2) {
    System.out.println(obj.toString());
}
```

Metóda remove pre zoznam odstráni prvý výskyt danej hodnoty

Prvky môžeme odstrániť aj podľa indexu

Množiny (1)

- Každý prvok sa môže vyskytovať iba raz
- Prvky sa porovnávajú pomocou metódy `equals`
- Umožňujú pridávanie, odoberanie a testovanie, či sa prvok v množine nachádza

Množiny (2)

- Viacero tried, ktoré sa odlišujú vnútornou implementáciou
- `java.util.HashSet`
  - Využíva sa hešovanie
  - Pre vlastnú implementáciu `equals()` je potrebné pridať vlastnú implementáciu metódy `int hashCode()`
- `java.util.TreeSet`
  - Prvky sú usporiadané v danom poradí
  - Objekty musia implementovať rozhranie `Comparable` aby sa dali usporiadať

Hešovanie

- Prvky sú rozdelené podľa hešovacieho kódu – čísla ktoré by malo byť vo väčšine prípadov rôzne pre rôzne objekty (môžu však byť aj konflikty)
- Pre každý objekt je možné vypočítať hešovací kód metódou `int hashCode()` definovanú v triede `Object`
- Keď predefinujeme vo vlastnej triede metódu `boolean equals(Object obj)` potom by sme mali vždy predefinovať aj metódu `hashCode`

Hešovanie - príklad

```
public class Adresa {

     private String ulica;
     private int cislo;
     ...

     @Override
     public boolean equals(Object obj) {
          if (obj instanceof Adresa) {
               Adresa adr = (Adresa)obj;
               return ulica.equals(adr.ulica) && cislo == adr.cislo;
          } else {
               return false;
          }
     }

     @Override
     public int hashCode() {
          return ulica.hashCode() + cislo;
     }

}
```

Všetky členské premenné ktoré porovnávame v equals by sme mali zahrnúť do hešovacieho kódu objektu. Pre objekty (aj reťazce) môžeme zahrnúť priamo ich hashCode(), čísla môžeme zahrnúť priamo

Množiny - príklad

```
List zoznam = new LinkedList();
zoznam.add("jedna");
zoznam.add("dva");
zoznam.add("jedna");

Set mnozina = new HashSet(zoznam);
System.out.println(mnozina.contains("jedna"));

Adresa adr1 = new Adresa("Letna", 9);
Adresa adr2 = new Adresa("Letna", 9);
mnozina.add(adr1);
mnozina.add(adr2);

for (Object obj : mnozina) {
    System.out.println(obj.toString());
}
```

Z prvkov zoznamu vytvoríme množinu, každý prvok sa v množine bude vyskytovať iba raz (tzn. mnozina bude mať 2 prvky)

Aj keď sú adr1 a adr2 samostatné objekty, podľa equals sú rovnaké, takže mnozina bude obsahovať iba jeden objekt adresy

Všimnite si, že štandardne môžete do jednej kolekcie pridávať rôzne typy (to platí aj pre zoznamy)

Zhrnutie

- Ošetrenie chýb
  - Definovanie vlastných typov
  - Kontrolované výnimky
- Kolekcie
  - Zoznamy
  - Množiny a hešovanie