Výjimky

Tomáš Pitner, upravil Marek Šabo

K čemu jsou výjimky

- Výjimky jsou mechanizmem umožňujícím reagovat na nestandardní (tj. chybové) běhové chování programu, které může mít různé příčiny:
  - chyba okolí: uživatele, systému
  - vnitřní chyba programu: tedy programátora.

Proč výjimky

- Mechanizmus, jak psát robustní, spolehlivé programy odolné proti chybám "okolí" i chybám v samotném programu.
- Výjimky v Javě fungují podobně jako v C++, C#, Ruby, Pythonu a dalších zejména objektových jazycích.
- v Javě jsou ještě lépe implementovány než v C++ (navíc klauzule `finally`)
- Podobně jako jinde, ani v Javě není dobré výjimkami potlačovat *očekávané* chyby programu samotného — to by bylo hrubé zneužití.

Co technicky jsou výjimky

- Výjimka (*Exception*) je objektem třídy `java.lang.Exception` nebo její podtřídy.
- Příbuzným typem jsou rovněž *vážné běhové chyby* — objekty třídy `java.lang.Error`.
- Každopádně v obou případech rozšiřuje třídu `java.lang.Throwable`.
- Ta je také často používaná jako deklarativní typ pro výjimky v širším slova smyslu.

Kdy výjimka vznikne

Objekt výjimky je vyhozen (`throw`) buďto:

1. *automaticky* běhovým systémem Javy, nastane-li nějaká běhová chyba — např. dělení nulou, nebo
2. jsou *vytvořeny a vyhozeny samotným programem*, zdetekuje-li nějaký chybový stav, na nějž je třeba reagovat — např. do metody je předán špatný argument.

Příklad vzniku výjimky

```
try {
   // First, the exception can be raised in JVM when
   // dereferencing using a bad index in an array:
   String name = args[i]; // ArrayIndexOutOfBoundsException
   // or the exception(s) can be raised by programmer
   // in the constructor of Person
   // NullPointerException or IllegalArgumentException
   p = new Person(name);
} catch(ArrayIndexOutOfBoundsException e) {
   System.err.println("No valid person name specified");
   System.exit(1);
} catch(NullPointerException e) {
   // reaction upon NullPointerException occurence
} catch(IllegalArgumentException e) {
   // reaction upon IllegalArgumentException occurence
}
```

Programové bloky pro práci s výjimkami

- Bloky programu pro práci s výjimkami:

  `try`
  : blok vymezující místo, kde může výjimka vzniknout

  `catch`
  : blok, který se vykoná, nastane-li výjimka odpovídající typu v `catch`

  `finally`
  : blok, který se vykoná vždy, viz dále.

Co se s vyhozenou výjimkou stane

Vyhozený objekt výjimky je buďto:

1. **Zachycen** v rámci metody, kde výjimka vznikla → a to v bloku `catch`.
   - Takto to fungovalo ve výše uvedeném příkladu — všechny výjimky byly zachyceny a řešeny svými bloky `catch`.
2. Nebo výjimka **propadne** do nadřazené (volající) metody, kde je buďto v bloku `catch` zachycena nebo opět propadne atd.
   - Výjimka tedy "putuje programem" tak dlouho, než je zachycena.
   - Pokud není nikde zachycena, běh JVM skončí s hlášením o výjimce.

Syntaxe kódu s ošetřením výjimek

```
try {
   //zde může vzniknout výjimka
} catch (ExceptionType exceptionVariable) {
   // zde je výjimka ošetřena
   // je možné zde přistupovat k "exceptionVariable"
}
```

- Bloku `try` se říká *hlídaný blok*,
- protože výjimky příslušného typu uvedeného v hlavičce/-ách bloků `catch` zde vzniklé jsou zachyceny.

Reakce na výjimku — možnosti

- Jak můžeme na vyhozenou výjimku reagovat?

  Napravit příčiny vzniku chybového stavu
  - např. *znovu* nechat načíst vstup nebo
  - poskytnout za chybný vstup *náhradu* — např. implicitní hodnotu;

  Operaci neprovést

  a sdělit chybu výše tím, že
  - výjimku *propustíme* z metody (propadne z ní).
  - Možnost úniku výjimky (propuštění) z metody se deklaruje pomocí `throws` v hlavičce metody.

Reakce na výjimku — pravidla

- Vždy nějak reagujme, tzn. výjimku neignorujme, nepotlačujme.
- Tudíž blok `catch` *nenechávejme prázdný*,
- přinejmenším vypišme `e.printStackTrace()`.
- Nelze-li rozumně reagovat na místě, propusťme výjimku výše a popišme to v dokumentaci, viz následující příklad.

Příklad komplexní reakce na výjimku

```
int i = 0;
String param;
int cislo = 0;
boolean ok = false;
do {
   try {
      // zde může vzniknout výjimka ArrayIndexOutOfBoundsException
      param = args[i];
      // zde může vzniknout výjimka NumberFormatException
      cislo = Integer.parseInt(param);
      // sem se dostane, jen když nevznikla žádná výjimka
      ok = true;
   } catch (NumberFormatException nfe) {
      // sem se dostane, byl-li vstup ve špatném formátu (ne číslo)
      System.err.println("Parametr "+i+" neni platne cele cislo");
      i++; // zkusíme další parametr
   } catch (ArrayIndexOutOfBoundsException iob) {
      // sem se dostane, byl-li překročena velikost pole -> chybový výstup err
      System.err.println("Nepredan zadny ciselny parametr");
      // sami vyhodíme výjimku
      throw new IllegalArgumentException(
         "Nezadan zadny celociselny parametr.");
   }
} while (!ok);
System.out.println("Zadano cislo="+cislo);
```

Kaskády bloků `catch`

- V některých blocích `try` mohou vzniknout *výjimky více typů* →
- pak můžeme uvádět více `catch` po sobě, viz přechozí příklad.
- Pokud `catch` takto řetězíme, musíme respektovat, že výjimka bude zachycena nejbližším příhodným `catch` a
- překladač si ohlídá, že kód neobsahuje *nedosažitelné* catch-bloky.
- Pozor tedy na řetězení `catch` s výjimkami typů z jedné hierarchie tříd.
- Vždy musí být výjimka z podtřídy (tj. speciálnější) uvedena a tedy zachycována dříve než výjimka obecnější.

Příklad kaskády `catch` špatně (1)

- Definice vlastních výjimek — jedna podtřídou druhé

```
class MyException1 extends Exception {}
class MyException2 extends MyException1 {}
```

Příklad kaskády `catch` špatně (2)

- Jejich chybné použití v kaskádě

```
try {
   if (args.length == 0 || args[0].equals("1"))
      throw new MyException1();
   else if (args[0].equals("2"))
      throw new MyException2();
// CHYBA: v kaskádě je nejprve zachycována obecnější výjimka MyException1
} catch (MyException1 mv) {
   System.out.println("Zachycena vyjimka typu MyException1: " + mv);
   // speciálnější výjimka MyException2 by nikdy nebyla zachycena,
   // proto překladač takovou konstrukce ani nepřeloží.
} catch (MyException2 mv) {
   System.out.println("Zachycena vyjimka typu MyException2: "+mv);
}
```