Výjimky (exceptions)

Výjimky jsou objekty sloužící k indikaci a nápravě závad v běhu programu.

Typem patří do stromu třídy Throwable - odtud dědí téměř všechny metody.

Výjimky pojednává:

- **Kompilátor v čase kompilace (compile-time) rozeznává výjimky:**
  - **kontrolované (checked) u nichž vyžaduje ošetření klauzulí try – catch anebo vyznačení throws v hlavičkách metod. Za tyto závady programátor zpravidla nemůže, ale měl by zajistit jejich zvládnutí.**
  - **nekontrolované (unchecked), u nichž nevyžaduje vyznačení ani ošetření. Tyto závady se považují za programátorovy chyby, které doufejme posléze odstraní.**
  - **pokud výjimka patří do stromu java.lang.RuntimeException, je nekontrolovaná, jinak je kontrolovaná.**

- **JVM v čase běhu (run-time) případně vytvoří objekt příslušného typu popisující vzniklou závadu, ten se tzv. vyhodí (throw) v naději, že se v nějaké klauzuli try najde řešení, které problém zmírní nebo alespoň vhodně navenek vyjeví.**

Hierarchie výjimek

```
java.lang.Object
    |
    |<--- java.lang.Throwable
              |<--- java.lang.Error
              |          |<--- java.lang.VirtualMachineError
              |
              |<--- java.lang.Exception
                        |<--- java.lang.RuntimeException   // Unchecked tree
                        |          |<--- java.lang.ArithmeticException
                        |          |<--- java.lang.NullPointerException
                        |          |<--- java.lang.IndexOutOfBoundsException
                        |          |<--- java.util.NoSuchElementException
                        |
                        |<--- java.io.IOException
                        |          |<--- java.net.SocketException
                        |          |          |<--- java.net.ConnectException
```

Třída Throwable

Třída Throwable obsahuje několik užitečných metod pro:

- **String toString( ) - výpis výjimky v textovém tvaru,**
- **String getMessage( ) - získání doprovodné zprávy z výjimky,**
- **výpis zásobníku pro vysledování volaných metod,**
- **manipulace s předchozími výjimkami.**

- **Strom Throwable obsahuj podstrom Error, který obsahuje třídy indikující těžké a neopravitelné chyby a to i systému Java.**
- **Všechny Error jsou kompilátorem nekontrolované (unchecked).**
- **Error či její podtyp lze záměrně vyhodit, ale patrně to nemá smysl, kromě AssertionError.**

Výjimky

Výjimky jsou buď JVM – vyhazuje je JVM (např. NullPointerException), či programatické – záměrně vyhazované pomoci throw v programu či API.

Java má mechanizmus vytváření, vyhazování a odchytávání výjimek:

- **Klauzule try se skládá nejméně ze dvou bloků. Bloky jsou tří typů:**
  hlídač - try (právě jeden)
  řešitel - catch (libovolný počet)
  uklízeč - finally (nanejvýš jeden)

- **Klauzule try může mít tři tvary:**
  try - catch - finally
  try - catch
  try - finally

- **Klauzule try - může být vnořena do kteréhokoli bloku a tedy i do bloku jiné klauzule try.**

Výjimky

- **Konkrétně:**

```
try {
   ....                                          //  hlídač - hlídá tento blok
}
catch ( xxxException ex )  { … }       //  řešitel - blok řešící xxx
catch ( yyyException ex )  { … }       //  řešitel - blok řešící yyy
   ......
catch ( zzzException ex )   { … }       //  řešitel - blok řešící zzz
finally                                { … }       //  uklízeč - úklidový blok
//  následné příkazy ( anebo ukončovací závorka nadbloku )
```

- **Je-li uklízeč přítomen, pak vždy dostane řízení těsně před ukončením zpracování klauzule - aby se dalo leccos uklidit. Nemá přímý přístup k vyhozené výjimce a nemůže.**
- **Řešitelé řeší jen ty výjimky patřící do typu jejich parametru. V seznamu musejí být uváděni od specifického k obecnému - jinak je specifický řešitel nedostupným kódem - to kompilátor nepřipustí.**

Hlídač

Hlídač dozírá, byla-li v bloku vyhozena výjimka (nezáměrně či záměrně):

- Ne: Před odchodem z hlídaného bloku předá řízení uklízeči.

- Ano: Další příkazy v try bloku se již neprovedou. Hlídač zjistí typ výjimky a procházením řešitelů shora dolů se snaží najít prvního kompetentního řešitele, tj. takového, jehož parametr je typem nebo nadtypem výjimky:

- **Nalezne-li ho, předá mu řízení.**
- **Nenalezne-li řešitele, předá řízení uklízeči a pak nevyřešenou výjimku vyhodí do nadbloku. Je-li tím blokem obalová try klauzule postupuje se obdobně. Je-li tím blokem metoda, pak tato metoda vyhodí výjimku do příkazu metody odkud byla zavolána a tam se postupuje obdobně. Nenalezne-li se žádný kompetentní řešitel, vyhodí se výjimka do obalové klauzule JVM - odtud byla zavolána metoda main(String[ ] args) - pak JVM vypíše hlášení a ukončí běh.**

Řešení výjimek

Řešitel má prostřednictvím parametru referenci k aktuální výjimce i dalším proměnným a může situaci trochu napravit anebo alespoň řádně ohlásit.
Nelze však nijak zařídit pokračování v nedokončeném hlídaném bloku.
I když řešitel vůbec nic neudělá, je odchycená výjimka vyřešena.

- **Vede-li odchod z bloků přes:**
  - **zavírací závorku, pak se řízení předá (event. po úklidu) následujícímu příkazu za klauzulí try - catch.**
  - **příkaz (return, break, continue, throw), pak se řízení předá (event. po úklidu) onomu příkazu.**

- **I řešitelé a uklízeč mohou vyhodit nějakou výjimku - tu však zpracuje (dynamicky) obalová try klauzule. Též lze sestrojit novou výjimku a jako její příčinu vložit tu původní.**

Vyznačení výjimek

Metody a konstruktory vyznačují ve svých hlavičkách typy kontrolovaných výjimek, které mohou vyhazovat. Řešení je pak na volajících metodách.

Syntax:

```
... ... m1( ... ) [ throws AaaException, BbbException, ... ] { … }
... ... m2( ... ) [ throws AaaException, BbbException, ... ] ;
```

- **Mechanizmus výjimek je časově náročný - má se využívat jen při závadách a nikoli pro testy podmínek.**
- **Deklarované výjimky se objeví ve standardní dokumentaci metod.**
- **Při jednoduchých pokusech lze obejít nepřehledné try klauzule připsáním throws Exception do hlaviček.**

Vlastní výjimky

- Vlastní výjimky se vytvoří jako potomek existující. Většina výjimek nepřidává žádné vlastní atributy ani metody – obvykle jen definuje dva konstruktory (jeden bez parametrů, druhý s parametrem String pro uložení doprovodné zprávy). Vše ostatní dědí z třídy Throwable. Hlavním indikátorem závady je (často velmi dlouhé) jméno výjimky – tím vzniká integrovaný "chybník" pro run-time.

- Příklad ukazuje definice výjimky pro případ chyby v tabulce.

```
public class TableException  extends Exception {
    int row, col;
    public TableException( int row, int col ) {        // konstruktor vytvoří
      super( "Wrong item at: " +row+ "  " +col );      // zprávu
      this.row = row;  this.col = col;                 // i explicitní indikaci
    }
}
```

- Výjimka se vytvoří a vyhodí např. takto:

```
throw  new  TableException( 333, 7 );
```

Příkaz assert

- byl přidán od v. 1.4 ke kontrole platnosti podmínek zejména při ladění programu. Příkazy assert jsou zevně ovladatelné - v běžném provozu bývají potlačené. proto. Užívat jen v private metodách či částech které by se neměly vlastně provádět.
- Syntax:

  assert podmínka [: výraz vracející hodnotu] ;

  při podmínce false se vyhodí chyba. Nepovolit vedlejší efekty výrazu.

  java.lang.AssertionError: [hodnota výrazu jako řetěz]

  tuto chybu nemá smysl odchytávat pomocí try - catch

- Má-li se chápat assert jako příkaz a ne jako jméno, nutno kompilovat:
  javac –source { 1.4 | 1.5 | 1.6 ... }
- Příkazy assert se ovládají spuštěním: java -ea ... / java -da …
  ClassLoader umožní en/dis-abled dle stromu balíčků či tříd.
  java -ea -da:com.pack1 či java -ea -da:com.pack1.MyProgram
- Více: http://java.sun.com/j2se/1.4.2/docs/guide/lang/assert.html