Výjimky

```
<link rel="stylesheet"  href="http://cdnjs.cloudflare.com/ajax/libs/font-
awesome/3.1.0/css/font-awesome.min.css">
```

Vtip

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 dalších objektových jazycích (C++, Python).

Vytvoření výjimky

- Výjimka, `Exception` je objektem třídy (nebo podtřídy) `java.lang.Exception`.
- Existují 2 základní konstruktory:

Constructor	Description
NullPointerException()	Constructs a NullPointerException with no detail message.
NullPointerException(String s)	Constructs a NullPointerException with the specified detail message.

Preferujeme konstruktor se zprávou.

Vyhození výjimky

Objekt výjimky je vyhozen:

1. automaticky běhovým systémem Javy, nastane-li nějaká běhová chyba — např. dělení nulou
2. samotným programem použitím klíčového slova `throw`, zdetekuje-li nějaký chybový stav, na nějž je třeba reagovat
    - např. do metody je předán špatný argument

```
if (x <= 0) {
  throw new IllegalArgumentException("x was expected to be positive");
}
```

Co se stane s vyhozenou výjimkou?

Vyhozený objekt výjimky je buďto:

1. **Zachycen** v rámci metody, kde výjimka vznikla (v bloku `catch`).
2. 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, program skončí s hlášením o výjimce.

Jak reagovat na výjimku

- Jestli výjimku nezachytíme, způsobí pád programu.
- Proto existuje mechanismus `try-catch` bloku, který umožňuje reagovat na vyhození výjimky.

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

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

Try & catch

Příklad zachycení výjimky I

- Blok `catch` výjimku zachytí.
- Vyhození výjimky programátorem, výjimka je zachycena v `catch` bloku:

```
try {
  throw new NullPointerException();
} catch(NullPointerException e) {
   System.out.println("NPE was thrown and I caught it");
}
```

Příklad zachycení výjimky II

- Vyhození výjimky chybou programu (a její zachytění):

```
int[] array = new int[] { 16, 25 };
try {
   int x = array[2];
} catch(ArrayIndexOutOfBoundsException e) {
   System.err.println("Only index 0 or 1 can be used");
}
```

Try a catch pod lupou

- Jak funguje `try` blok?
  - vykonává se příkaz za příkazen
  - jestli dojde k vyhození výjimky, **další kód v `try` se přeskočí** a kontroluje se `catch` blok
  - jestli **nedojde** k vyhození výjimky, kód se vykoná a bloky s `catch` se ignorují
- Jak funguje `catch` blok?
  - syntax je `catch(ExceptionType variableName) { ⋯ }`
  - jestli se typ výjimky **zhoduje anebo je nadtřídou**, vykoná se kód `catch` bloku
  - jestli se typ výjimky **nezhoduje**, výjimka není zachycena
  - `catch` bloků může být víc, pak se prochází postupně
  - vždy se vykoná maximálně jeden `catch` blok

Příklad více `catch` bloky

```
try {
   String s = null;
   s.toString(); // NPE exception is thrown
   s = "This will be skipped";
} catch(IllegalArgumentException iae) {
   System.err.println("This will not be called");
} catch(NullPointerException npe) {
   System.err.println("THIS WILL BE CALLED");
} catch(ArrayIndexOutOfBoundsException e) {
   System.err.println("This entire block will be skipped");
}
```

Proměnná v `catch`

- Catch blok kromě typu výjimky obsahuje i proměnnou, která se dá použít v rámci bloku:

```
try {
   new Long("xyz");
} catch (NumberFormatException e) {
   System.err.println(e.getMessage());
}
```

Sloučení `catch` bloků

```
try {
   Person p = new Person(null);
} catch(NullPointerException e) {
   System.err.println("Invalid name.");
} catch(IllegalArgumentException e) {
   System.err.println("Invalid name.");
}
```

Operátor `|` sloučí stejné `catch` bloky:

```
try { ... }
  catch(NullPointerException | IllegalArgumentException e) {
   System.err.println("Invalid name.");
}
```

Reakce na výjimku — možnosti

Jak můžeme na vyhozenou výjimku reagovat?

Napravit příčiny vzniku chybového stavu

- opakovat akci (např. znovu nechat načíst vstup)
- poskytnout náhradu za chybný vstup (např. implicitní hodnotu)

Operaci neprovést

- sdělit chybu výše tím, že výjimku *propustíme* z metody (propadne z ní)

Oracle Java Tutorials: Lesson: Handling Errors with Exceptions

Kaskády bloků `catch`

- Pokud `catch` řetězíme, musíme respektovat, že výjimka bude zachycena nejbližším příhodným `catch`.
- Překladač si ohlídá, že kód neobsahuje *nedosažitelné* catch-bloky, např:

```
try {
  ...
} catch (Exception e) {
  ...
} catch (IllegalArgumentException e) {
  // won't compile, unreachable code
}
```

Výjimka z podtřídy (speciálnější) musí být zachycována dříve než výjimka obecnější.

Výjimky — jak ne I

```
try {
  ...
} catch ( Exception e ) {
  ...
}
```

Problém: Zachytáváme všechny výjimky, některé výjimky ale vždy chceme propouštět.

Řešení: Použít v `catch` speciálnější typ třídy.

Výjimky — jak ne II

```
try {
  ...
} catch ( NullPointerException e ) {
}
```

Problém: Prázdny `catch` blok — nedozvíme se, že výjimka byla vyhozena.

Řešení: Logovat, vypsat na chybový výstup nebo použít `e.printStackTrace();`

Výjimky — jak ne III

```
try {
  throw new NoSuchMethodException();
} catch ( NoSuchMethodException e ) {
  throw e;
}
```

Problém: Kód zachytí a následně vyhodí stejnou výjimku.

Řešení: Blok `catch` smazat — je zbytečný.

Repl.it demo k výjimkám - základy

- https://repl.it/@tpitner/PB162-Java-Lecture-09-exceptions