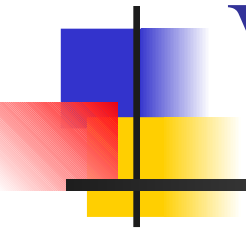
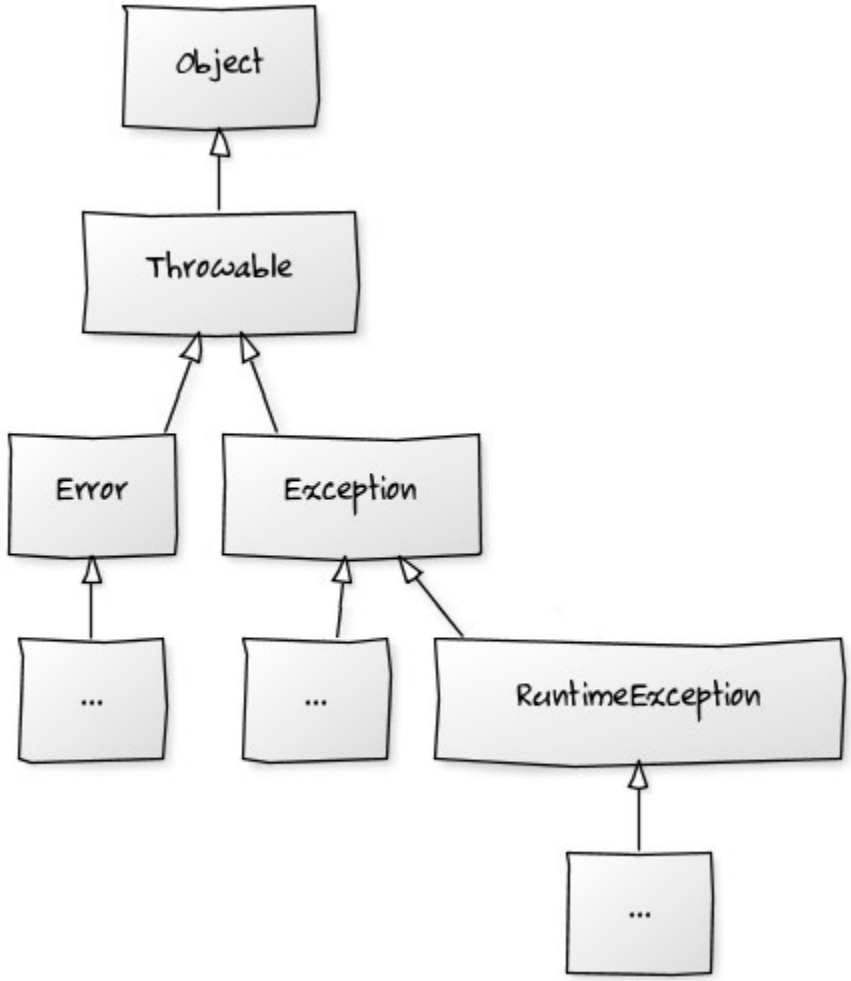


Výjimky

Výjimky

- Existují tři typy výjimek:
  - **`Error`**
    - Závažná chyba, není nutné ošetřovat
    - `OutOfMemoryError`
  - **`Exception`**
    - Klasická výjimka, kterou JE nutné ošetřovat
    - `IOException`, `SQLException`
  - **`RuntimeException`**
    - Není nutné ošetřovat
    - `NullPointerException`

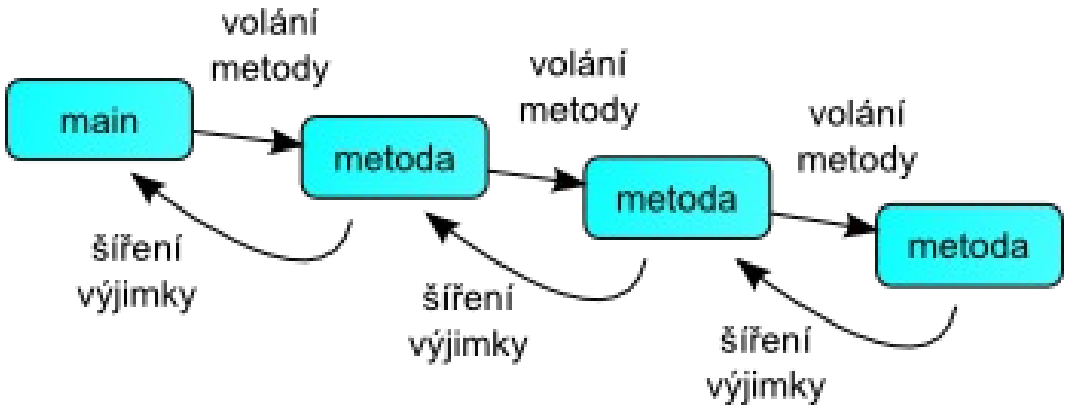


Vznik a šíření výjimky

- Nejprve je nutné vytvořit výjimku a poté ji „vyhodit“ (začít její šíření) zavoláním operátoru throw:

```
throw new Exception("výjimka");
```

- Jakmile se vyhodí výjimka, tak se hledá obsluha (handler) výjimky. Pokud se nenajde v metodě, kde se výjimka vyskytla, hledá se v metodě, která příslušnou metodu zavolala atd.



Stacktrace

- V případě, že se nenalezne obsluha výjimky v metodě main, pak se aplikace ukončí a na obrazovku se vypíše stacktrace:
- Příklad:

```
int a = Integer.parseInt("10x");
```

- Výsledek:

```
Exception in thread "main"
 java.lang.NumberFormatException: For input string: "10x"
at java.lang.NumberFormatException.forInputString
 (Unknown Source)
at java.lang.Integer.parseInt(Unknown Source)
at java.lang.Integer.parseInt(Unknown Source)
at cz.skoleni.java.hw.Aplikace.main(Aplikace.java:6)
```

Obsluha výjimky I.

- V případě, že po Vás Java chce ošetřit výjimku a vy ji nechcete v aktuální metodě ošetřovat (ošetří ji metoda, která Vaši metodu zavolala), pak přidejte k hlavičce metody klíčové slovo `throws` a za něj typ výjimky, kterou nechcete ošetřovat:

```
public void readFile() throws IOException {
  // čtení dat ze souboru
}
```

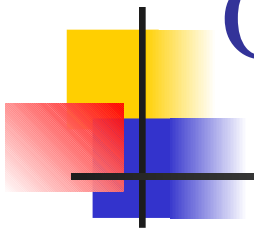
- Za operátorem `throws` můžete definovat více typů výjimek, které budou oddělené čárkou.
- Když přidáte operátor `throws` k hlavičce metody `main`, pak se při výskytu chyby aplikace ukončí a vypíše se na obrazovku stacktrace.

Obsluha výjimky II.

- Obsluha výjimky se provádí pomocí bloku `try - catch - finally`:

```
try {
  // blok, kde může dojít k výjimce
} catch (TypVyjimky1 e1) {
  // obsluha výjimky typu TypVyjimky1
} catch (TypVyjimky2 e2) {
  // obsluha výjimky typu TypVyjimky2
} finally {
  // blok, který se zavolá nezávisle na tom, jestli
  // výjimka nastala, a to i při použití příkazu
  // return v "try" bloku
}
```

Obsluha výjimky III.



- Pár poznámek:
  - V bloku `catch` můžete pomocí operátoru `throw` znovu vyhodit výjimku a pokračovat v jejím šíření:

    ```
    throw e;
    ```

  - Nebo můžete výjimku přetransformovat na jiný typ výjimky:

    ```
    throw new TypVyjimky(e);
    ```

- Abyste zachytili úplně všechny výjimky které mohou nastat, musíte také ošetřit výjimky typu `Error`.
- Blok `finally` se používá pro uzavření připojení do souboru, databáze apod.
- Můžete si také definovat vlastní výjimky (stačí udělat třídu, která bude potomkem třídy `Exception`).

Best practices I.

- Je blbost používat návratové kódy:

```
/**
 * Tato metoda vrati pocet souboru a adresaru ve vstupnim adresari
 * @param directory Vstupni adresar
 * @return Pocet souboru a adresaru ve vstupnim adresari, nebo -1 pokud je
 *         vstup null, nebo -2 pokud vstupni adresar neni adresar
 */
public int countFiles(File directory) {
   if (directory == null) { return -1; }
   if (!directory.isDirectory()) { return -2; }
   return directory.listFiles().length;
}
```

Best practices II.

- Místo toho se používají výjimky:

```
public class UnsupportedInputException extends RuntimeException { }


/**
 * Tato metoda vrati pocet souboru a adresaru ve vstupnim adresari
 * @param directory Vstupni adresar
 * @return Pocet souboru a adresaru ve vstupnim adresari
 * @throws UnsupportedInputException Pri nespravnem vstupu
 */
public int countFiles(File directory) {
   if (directory == null || !directory.isDirectory()) {
      throw new UnsupportedInputException();
   }
   return directory.listFiles().length;
}
```