

Seminář Java

Výjimky

Radek Kočí

Fakulta informačních technologií VUT

Únor 2008

Co a k čemu jsou výjimky

- výjimka je mechanismus umožňující psát robustní, spolehlivé programy
- robustní ve smyslu odolné proti chybám "okolí" – uživatele, systému, ...
- výjimkami *v žádném případě* neošetřujeme chyby programu samotného! (hrubé zneužití)
- režie spojená s vyvoláním výjimky je vysoká!

Reprezentace výjimky

- výjimka (exception) je objekt třídy `java.lang.Exception`
- Objekty (výjimky) jsou vytvářeny (vyvolávány) buďto
 - automaticky běhovým systémem Javy, nastane-li nějaká běhová chyba, např. dělení nulou, nebo
 - jsou vytvořeny samotným programem, zdetekuje-li nějaký chybový stav, na nějž je třeba reagovat – např. do metody je předán špatný argument

Zpracování výjimky

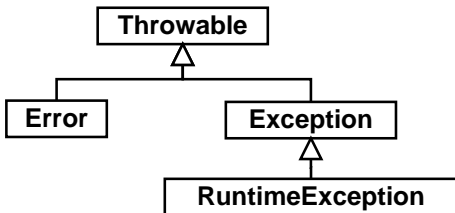
- Vzniklý objekt výjimky je předán buďto:
 - v rámci metody, kde výjimka vznikla, do bloku `catch` ⇒ výjimka je v bloku `catch` tzv. zachycena
 - výjimka "propadne" do nadřazené (volající) metody, kde je buďto v bloku `catch` zachycena nebo opět propadne atd.
- Výjimka tedy "putuje programem" tak dlouho, než je zachycena
- ⇒ pokud není, běh JVM skončí s hlášením o výjimce

```
try {  
    //zde může vzniknout výjimka  
}  
catch (TypVýjimky proměnnáVýjimky) {  
    // zde je výjimka ošetřena  
    // je možné přistupovat k proměnnéVýjimky  
}
```

-
- Bloku `try` se říká hlídaný blok, protože výjimky (příslušného hlídaného typu) zde vzniklé jsou zachyceny.
 - V bloku `catch` jsou zachycené výjimky ošetřeny

Hierarchie výjimek

package `java.lang`



- **Throwable** – pouze objekty této třídy (a podtříd) mohou být generovány jako výjimky
- **Error** – vážné chyby JVM (*Out Of Memory, Stack Overflow, ...*)
- **Exception** – hlídané výjimky (checked exceptions)
- **RuntimeException** – běhové (runtime, nehlídané – unchecked) výjimky, takové výjimky nemusejí být zachytávány

Hlídané výjimky

- `java.lang.Exception`
- indikuje podmínky (stavy), které může aplikace chtít ošetřovat
- musí se explicitně uvádět
- hlídaná výjimka musí být zpracována!
- např. `java.io.FileNotFoundException`

Propuštění hlídané výjimky

- někdy není nutné či vhodné ošetřovat výjimku na místě
- metoda může deklarovat, že *propouští* výjimku (`throws`)
- klauzule `throws` říká, že během zpracování metody může být vygenerovaná uvedená výjimka, která není ošetřena
- volající metoda musí výjimku zpracovat (zachytit nebo propustit)

```
public FileReader(String fileName)
    throws FileNotFoundException;

public void close() throws IOException;
```



```
import java.io.*;
public class OtevreniSouboru {
    public static void main(String[] args) {
        String jmeno = args[0];
        FileReader r;
        System.err.println("Otviram soubor "+jmeno);
        r = new FileReader(jmeno);
        System.err.println("Soubor otevren");
        r.close();
    }
}
```

unreported exception

`java.io.FileNotFoundException`; must be caught
or declared to be thrown

```
    r = new FileReader(jmeno);
```

unreported exception `java.io.IOException`; must
be caught or declared to be thrown

```
    r.close();
```

- výjimku jsou hierarchicky uspořádané podle dědičnosti jejich tříd
- nadřazená výjimka pokrývá všechny odvozené výjimky
- př.: `FileNotFoundException` je speciálním případem `IOException`
- zpracováním `IOException` zpracujeme i `FileNotFoundException`

Zachycení výjimky

```
public static void main(String[] args) {  
    String jmeno = args[0];  
    FileReader r;  
    System.err.println("Otviram soubor "+jmeno);  
  
    try {  
        r = new FileReader(jmeno);  
        System.err.println("Soubor otevren");  
        r.close();  
    } catch (IOException ex) {  
        System.err.println("Chyba pri manipulaci  
                               se souborem.");  
    }  
}
```

Je možné zpracovat každou výjimku zvlášť.
Pozor na řazení podle hierarchie!

```
public static void main(String[] args) {  
    String jmeno = args[0];  
    FileReader r;  
    System.err.println("Otviram soubor "+jmeno);  
    try {  
        r = new FileReader(jmeno);  
        System.err.println("Soubor otevren");  
        r.close();  
    } catch (FileNotFoundException ex) {  
        ...  
    } catch (IOException ex) {  
        ...  
    }  
}
```

Zachycení výjimky – chyba

```
public static void main(String[] args) {
    String jmeno = args[0];
    FileReader r;
    System.err.println("Otviram soubor "+jmeno);
    try {
        r = new FileReader(jmeno);
        System.err.println("Soubor otevren");
        r.close();
    } catch (IOException ex) {
        ...
    }
    // Nasledujici blok se nikdy neprovede !!!
    catch (FileNotFoundException ex) {
        ...
    }
}
```

Propuštění výjimky

```
public FileReader(String fileName)
    throws FileNotFoundException;
public void close() throws IOException;
```

```
public class OtevreniSouboru {
    static void otevri(String jmeno) {
        System.err.println("Otviram soubor "+jmeno);
        FileReader r = new FileReader(jmeno);
        r.close();
    }
    public static void main(String[] args) {
        otevri(args[0]);
        System.err.println("Soubor otevren");
    }
}
```


Propuštění výjimky

```
static void otevri(String jmeno)
    throws IOException
{
    System.err.println("Otviram soubor "+jmeno);
    FileReader r = new FileReader(jmeno);
    r.close();
}

public static void main(String[] args) {
    try {
        otevri(args[0]);
        System.err.println("Soubor otevren");
    } catch (IOException ioe) {
        System.err.println("Nelze otevrit soubor");
    }
}
```

Nehlídané výjimky

- `java.lang.RuntimeException`
- výjimky, které mohou být generovány během standardních operací JVM
- nemusí se explicitně uvádět
- nemusí se zachytávat
- např.
`java.lang.ArrayIndexOutOfBoundsException`,
`java.lang.NullPointerException`

Generování výjimky

- výjimku lze generovat (klíčové slovo `throw`)

```
public abstract class Reader ... {
    public void mark(int limit)
        throws IOException
    {
        throw new IOException("mark() not supported");
    }

    public InputStreamReader(.., String charsetName)
    {
        if (charsetName == null)
            throw new NullPointerException("charsetName")
        }
    }
}
```

Klauzule (blok) `finally`:

- může následovat ihned po bloku `try` nebo až po blocích `catch`
- slouží k "úklidu v každém případě", tj.
 - když je výjimka zachycena blokem `catch`
 - i když je vygenerovaná jiná než ošetřovaná výjimka
 - i když je výjimka propuštěna do volající metody
- Používá se typicky pro uvolnění systémových zdrojů – uzavření souborů ...

Vlastní výjimky

- typy (tj. třídy) výjimek si můžeme definovat sami
- bývá zvykem končit názvy tříd (výjimek) na `Exception`
- *je lepší využívat standardní výjimky!*

```
class MyException extends Exception {  
    protected int pocetParametru;  
  
    public MyException(int pocet) {  
        pocetParametru = pocet;  
    }  
  
    public int getPocetParametru() {  
        return pocetParametru;  
    }  
}
```

Ukázka vlastní výjimky a klauzule finally

```
public static void main(String[] args) {
    int pocetParametru = args.length;
    try {
        if (pocetParametru < 2)
            throw new MyException(pocetParametru);
        System.out.println("Spravny pocet: "
            + pocetParametru);
    } catch (MyException me) {
        System.out.println("Malo parametru: "
            + me.getPocetParametru());
    } finally {
        System.out.println("Konec");
    }
}
```

Jak můžeme reagovat?

- Napravit příčiny vzniku chybového stavu – např. znovu nechat načíst vstup
- Poskytnout za chybný vstup náhradu – např. implicitní hodnotu
- Operaci neprovést ("vzdát") a sdělit chybu výše tím, že výjimku "propustíme" z metody

Výjimková pravidla:

- Vždy nějak reagujeme! Neignorujeme, nepotlačujeme, tj.
- blok `catch` nenecháme prázdný, přinejmenším vypíšeme `e.printStackTrace()`
- Nelze-li reagovat na místě, propustíme výjimku výše (a popíšeme to v dokumentaci...)

Ukázka nesprávného použití výjimky

```
public static void main(String[] args) {  
    try {  
        otevri(args[0]);  
    }  
    catch (ArrayIndexOutOfBoundsException e) {  
        System.err.println("Neni zadan argument.");  
    }  
}
```

```
public static void main(String[] args) {  
    if (args.length == 0) {  
        System.err.println("Neni zadan argum.");  
        return;  
    }  
    otevri(args[0]);  
}
```


Ukázka nesprávného použití výjimky

```
try {  
    int i = 0;  
    while(true) {  
        a[i++] = i;  
    }  
} catch (ArrayIndexOutOfBoundsException e) {}
```

```
int length = a.length;  
for(int i = 0; i < length; ) {  
    a[i++] = i;  
}
```