Seminář Java
V

Radek Kočí

Fakulta informačních technologií VUT

7. března 2007

Téma přednášky

- Výjimky
- JAR, ANT
- Enums, Varargs, Static import

Výjimky

Co a k čemu jsou výjimky

- výjimka je mechanizmus 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!

Výjimky

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

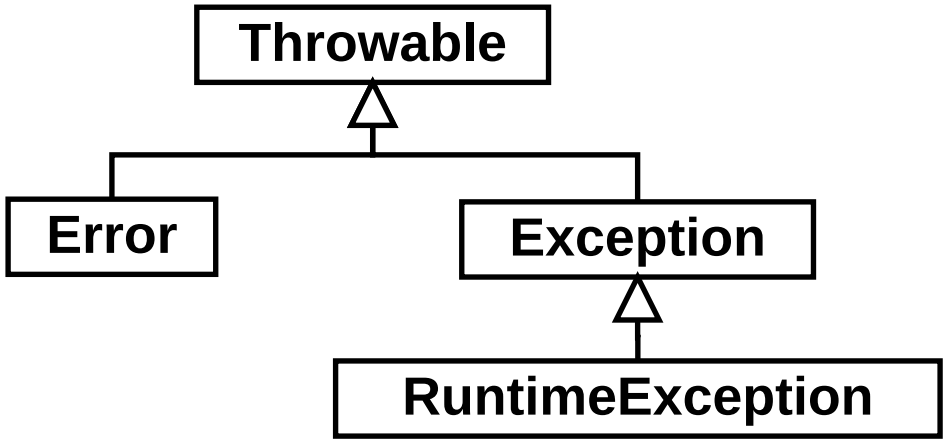
Syntaxe kódu s ošetřením výjimek

```
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;
```

Hlídané výjimky – ukázka

```
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();
   }
}
```

Hlídané výjimky – ukázka

```
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();
```

Ošetření výjimek – hierarchie

- 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.");

    }

}
```

Zachycení výjimky

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

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 (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

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 `finally`

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");
      }
   }
}
```

Reakce na výjimku

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 reagujme! Neignorujme, nepotlačujme, tj.
- blok `catch` nenechávejme prázdný, přinejmenším vypišme `e.printStackTrace()`
- Nelze-li reagovat na místě, propusťme výjimku výše (a popišme 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;
}
```

Java Archive (JAR)

- platformově nezávislý
- archiv (zip) obsahující
  - hierarchii balíků a class soubory
  - jakékoliv jiné soubory (obrázky pro aplety apod.)
  - speciální adresář `META-INF`

`META-INF`

- obsah je interpretován JVM
- konfigurace aplikace
- konfogurace rozšíření
- konfigurace zavaděčů tříd a služeb

Zkompilovani zdrojovych textu

```
-- src
    |---- xml
            |---- XMLDemo.java
-- dest
-- dom4j-1.5.2.jar
```

```
javac -classpath "src:dom4j-1.5.2.jar"
              -d dest src/xml/XMLDemo.java
```

Zkompilovani zdrojovych textu

```
-- src
    |---- xml
            |---- XMLDemo.java
-- dest
    |---- xml
            |---- XMLDemo.class
-- dom4j-1.5.2.jar
```

Java Archive (JAR)

```
-- dest
    |---- xml
            |---- XMLDemo.class
-- dom4j-1.5.2.jar
```

```
jar -cvf xml.jar -C dest xml
```

```
MANIFEST.MF:

Manifest-Version: 1.0
Created-By: 1.5.0_05 (Sun Microsystems Inc.)
```

Java Archive (JAR)

```
-- dest
    |---- xml
            |---- XMLDemo.class
-- dom4j-1.5.2.jar
```

```
jar -cvfm xml.jar mymanifest.mf -C dest xml
```

```
MANIFEST.MF:
Manifest-Version: 1.0
Class-Path: dom4j-1.5.2.jar
Created-By: 1.5.0_05-b05 (Sun Microsystems Inc.)
Ant-Version: Apache Ant 1.6.2
Main-Class: xml.XMLDemo
```

Java Archive (JAR)

Spuštění JAR souboru

```
java -jar xml.jar
```

---

http://java.sun.com/j2se/1.5.0/docs/guide/jar/

Apache Ant

Apache Ant

- nástroj pro sestavování aplikací
- v principu podobný nástroji `make`
- sestavovací soubory založeny na XML formátu

Sestavovací soubor (buildfile)

- projekt (project)
- cíle (targets)
- úlohy (tasks)
- závislosti

`http://ant.apache.org`

Apache Ant

Soubor `build.xml`

```
<project>
  <target name="clean">
    <delete dir="build"/>
  </target>

  <target name="compile">
    <mkdir dir="build/classes"/>
      <javac srcdir="src" destdir="build/classes"/>
  </target>
```

Apache Ant

```
    <target name="jar">
        <mkdir dir="build/jar"/>
        <jar destfile="build/jar/xml.jar"
                        basedir="build/classes">
            <manifest>
                <attribute name="Main-Class"
                        value="xml.XMLDemo"/>
            </manifest>
        </jar>
    </target>

    <target name="run">
        <java jar="build/jar/xml.jar" fork="true"/>
    </target>
</project>
```

Apache Ant

```
ant compile
ant jar
ant run
```

Enums

Reprezentace výčtového typu (Java $<$ 5.0)

- int Enum pattern

```
public static final int SEASON_WINTER = 0;
public static final int SEASON_SPRING = 1;
public static final int SEASON_SUMMER = 2;
public static final int SEASON_FALL = 3;
```

- není typově bezpečný
- hodnoty nejsou informativní (vždy int; ”jiné” názvy, stejná hodnota)

Výčtový typ (Java $>=$ 5.0)

- typově bezpečný
- `enum Season {WINTER, SPRING, SUMMER, FALL}`
- plnohodnotná třída!
- rozšiřuje třídu `java.lang.Enum`

Enums

Reprezentace výčtového typu (Java $<$ 5.0)

- int Enum pattern

```
public static final int SEASON_WINTER = 0;
public static final int SEASON_SPRING = 1;
public static final int SEASON_SUMMER = 2;
public static final int SEASON_FALL = 3;
```

- není typově bezpečný
- hodnoty nejsou informativní (vždy int; ”jiné” názvy, stejná hodnota)

Výčtový typ (Java $>=$ 5.0)

- typově bezpečný
- `enum Season {WINTER, SPRING, SUMMER, FALL}`
- plnohodnotná třída!
- rozšiřuje třídu `java.lang.Enum`

Enums

```
class EnumsDemo {
    public enum Season { WINTER, SPRING, SUMMER,
                         FALL };
    private Season season;

    public static void main(String[] args) {
        new EnumsDemo();
    }
    public EnumsDemo() {
        season = Season.WINTER;
        System.out.println(season);
        for (Season s : Season.values())
            System.out.println("Value of Season: "
                                + s);
    }
}
```

Enums

```
public enum Planet {
    MERCURY (3.303e+23, 2.4397e6),
    VENUS   (4.869e+24, 6.0518e6),
    EARTH   (5.976e+24, 6.37814e6);

    private final double mass;   // in kilograms
    private final double radius; // in meters

    Planet(double mass, double radius) {
        this.mass = mass;
        this.radius = radius;
    }
```

Enums

```
    public static final double G = 6.67300E-11;

    double surfaceGravity() {
        return G * mass / (radius * radius);
    }

    double surfaceWeight(double otherMass) {
        return otherMass * surfaceGravity();
    }
}
```

Enums

```
public static void main(String[] args) {
  double earthWeight = Double.parseDouble(args[0]);
  double mass =
     earthWeight/Planet.EARTH.surfaceGravity();

  for (Planet p : Planet.values())
    System.out.printf("Your weight on %s is %f%n",
                      p, p.surfaceWeight(mass));
}
```

---

http://java.sun.com/j2se/1.5.0/docs/guide/language/enums.html

Varargs

Varargs

- metoda s proměnným počtem parametrů
- pole objektů třídy `Object`

```
Object[] arguments = {
    new Integer(7),
    new Date(),
    "a disturbance in the Force"
};

String result = MessageFormat.format(
    "At {1,time} on {1,date}, there was {2} " +
    "on planet {0,number,integer}.", arguments);
```

Varargs

Varargs (Java $>=$ 5.0)

- vlastnost skrývající nutnost obalení polem
- poslední argument: `Object... name`

Varargs

```
class Args {

    public static void main(String[] args) {
        Args a = new Args();
        a.print("Hi ", "hou ", "ha");
    }

    public void print(String s, Object... args) {
        String str = "";
        System.out.println(args.length);
        for (Object o : args)
            str += (String) o;
        System.out.println(s + str);
    }
}
```

Static import

Přístup ke statickým členům

- `double r = Math.cos(Math.PI * theta);`
- Využití statického importu

```
import static java.lang.Math.PI;
//import static java.lang.Math.*;
...

double r = cos(PI * theta);
```

- používat velice opatrně! (kolize identifikátorů, těžko čitelný kód, ...)