Seminář Java

VII

Rekapitulace

- Vstup a výstup
  - Koncepce proudů
  - Práce se soubory a adresáři
  - Binární proudy, třídy `InputStream`, `OutputStream`
  - Znakové proudy, třídy `Reader`, `Writer`
  - Filtrované proudy
  - Standardní vstup a výstup
- Komprimace, `GZIPInputStream` a `GZIPOutputStream`
- Serializace

Obsah

- Grafické uživatelské rozhraní
- Swing vs AWT
- Aplety
  - Aplikační rámec, `JApplet`
  - spouštení v prohlížeči, `Appletviewer`
- Událostní model knihovny Swing
  - události
  - posluchači
- Jednotlivé komponenty knihovny Swing
  - přehled, příklady
- Rozmístění komponent
  - Správci rozmístění
- Vizuální programování a komponenty JavaBeans

Swing

Swing

- nové grafické rozhraní dostupné od verze 1.2.x
- součást JFC (Java Foundation Classes)
- konečná verze GUI pro Javu

Awt (Abstract Window Toolkit)

- starší varianta, od verze 1.x.x
- od verze 1.1.x, událostně řízená varianta
- velice omezené možnosti
- v budoucnu se s ní nepočítá

Aplety

- bezpečné, "sandbox"
- velice omezené možnosti
- programování na straně klienta
- rozšíření funkčnosti WWW stánky

Omezení

- žádný přístup k obsahu pevného disku lokálního počítače
  - digitální podpis apletů umožní zmírnit některá omezení (důvěryhodné aplety)
- pomalé zobrazení a načítání
  - ke stažení třídy je nutná cesta na server
  - balit do `jar` archivů

Aplety - výhody

- žádné požadavky na instalaci (plugin)
  - platformově nezávislý
  - automatická instalace a spuštění
- bezpečný
  - omezení vyplývající z běhu v "sandbox"

Aplikační rámec

- knihovny navrženy jako "stavební kameny", používané při tvorbě nových tříd (pomoc při vytváření nových aplikací)
- definuje řadu tříd určitého chování
- vlástní třída (potomek), překryje tyto metody, upraví na požadované chování

Aplety jsou vytvářeny pomocí aplikačního rámce.

Aplikační rámec - `JApplet` *(1)*

Hlavní třídu odvozujeme od třídy `javax.swing.JApplet` a pouze překrýváme příslušné metody.

- `init()`
  - volána automaticky k vykonání inicializace apletu, včetně rozložení komponent
  - vždy překrývat
- `start()`
  - volána pokaždé když se aplet přesune **do** viditelné oblasti okna prohlížeče
  - volána po dokončení `init()`
  - umožňuje spustit operace zastavené v metodě `stop()`

Aplikační rámec - `JApplet` *(2)*

- `stop()`
  - volána pokaždé, když se aplet přesune **z** viditelné oblasti okna prohlížeče
  - umožňuje zastavit operace zbytečně zabírající systémové prostředky
  - volána před metodou `destroy()`
- `destroy()`
  - volána v okamžiku, kdy je aplet uvolňen ze stránky
  - konečné uvolnění systémových prostředků, aplet již nebude používán

Příklad - první aplet

```
public class Applet1 extends JApplet {

  public void init() {
    getContentPane().add(new JLabel("Ahoj svete"));
  }

}
```

- všechny komponenty musí být umístěny v "podokně obsahu" (`getContentPane()`)

Spouštění apletů - prohlížeč

- umístit aplet na stránku
- předat požadované parametry
- předat libovolné argumenty

```
<APPLET codebase="../.." code="ija/examples/Applet1.class"
  width=350 height=200></APPLET>

<PARAM name="indentifikator" value="hodnota">
```

Parametry

- `code` - název souboru `.class`, v němž je aplet uložen
- `width, height` - počáteční velikost apletu (v pixelech)
- `codebase` - místa dalších souborů `.class`
- `align` - zarovnání
- `name` - identifikátor pro komunikaci mezi aplety

Spouštění apletů - `Appletviewer`

- `appletviewer` - prohlížeč apletů
- zdarma, obsažen v JDK
- vyhledá v soborech HTML všechny značky <applet> a spustí je, aniž by zobrazoval okolní HTML text

Událostní model knihovny Swing

Základnám principem tvorby aplikací s GUI je **řízení programu událostmi.** Netýka se však pouze GUI, je to obecnější pojem označující typ asynchronního programování.

- tok programu řízen událostmi
- události jsou vyvolány obvykle určitou uživatelskou akcí
- událost může spustit (iniciovat) každá komponenta
- po vyvolání je událost přijata jedním nebo více "handlery" (listener), jež na ni reagují
- událostní aplikace musí být většinou programovány jako vícevláknové

Řízení událostmi

Životní cyklus události

- událost vznikne (typicky uživatelskou akcí nad komponentou GUI)
- nad komponentu musí být zavěšen posluchač (listener) dané události (event)
- systém vyvolá příslušnou metodu posluchače - implementujeme ji aby realizovala námi potřebnou akci

Balíčky

- `java.awt` - základní komponenty GUI AWT
- `java.awt.event` - události GUI AWT
- a v ostatních balících `java.awt.*`
- `javax.swing` - základní komponenty GUI Swing
- `javax.swing.event` - události GUI Swing
- a v ostatních balících `javax.swing.*`

Příklad - Ahoj svete, Swing

```
public class Swing1 {

  public static void main(String[] args) {
    // Vytvoříme hlavní okno aplikace
    JFrame okno = new JFrame("Test Swingu");
    // Vložíme do něj text
    final JLabel label = new JLabel("Ahoj svete");
    okno.getContentPane().add(label);
    // Definujeme implicitní operaci při zavření okna
    okno.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    okno.pack();
    // zobrazíme ho
    okno.setVisible(true);
  }

}
```

Typy událostí

Události související s uživatelskou akcí nad/s:

- oknem - `WindowEvent`
- klávesnicí - `KeyEvent`
- myší (klikání a pohyb) - `MouseEvent`
- získáním nebo ztrátou fokusu - `FocusEvent`
- (obecnou) akcí nad GUI (stisk tlačítka) - `ActionEvent`

Každá komponenta knihovny Swing obsahuje metodu `addXXXListener()` a `removeXXXListener()`, kde `XXX` zastupuje také argument této metody.

```
addWindowListener(WindowListener l)
```

Přidání posluchače událostí

Aby událost mohla být někde ošetřena, tj. mohlo se na ni někde reagovat, je třeba k dané komponentě přidat objekt posluchače událostí. Velmi často se jako objektu posluchače používá anonymní vnitřní třídy

- třída je (jakoby "on-the-fly") definována a ihned - jen jedenkrát - použita
- ve skutečnosti se samozřejmě vytvoří a přeloží s ostatními hned při překladu mateřské třídy

V případě posluchačů událostí má vnitřní třída obvykle jen jednu metodu.

Příklad - událost zavření okna

Pomocí vnitřní třídy

- výhody
  - vnitřní třída má přístup (i k chráněným) prvkům mateřské třídy
- nevýhody
  - nepřehledné, třída je skryta v ostatním kódu
  - pokud si nepamatujeme odkaz, nemůžeme jej z paměti odstranit

```
okno.addWindowListener(
  new WindowListener() {
    public void windowClosing(WindowEvent e) {
      System.exit(0);
    }
  }
);
```

Příklad - událost zavření okna

Totéž bez anonymní vnitřní třídy

```
class MyWindowListener implements WindowListener {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
}
  .
  .
  .
okno.addWindowListener(new MyWindowListener());
```

Komponenty knihovny Swing - přehled

- JApplet, JFrame, JWindow, JDialog
- JButton
- JPanel
- JTextField, JTextArea, JTextPane
- JScrollPanes
- JCheckBox, JRadioButton
- JComboBox, JList
- JTabbedPane
- JOptionPanePane
- JMenuBar, JMenu, JMenuItem, JPopUpMenu
- JFileChooser
- JTree
- JTable
- ...

Příklad - vytvoření tlačítka (1)

```
public class Swing2 extends JFrame {

    private JButton bt1, bt2;
    private JTextField tf;

    public Swing2() {
        // Definujeme implicitní operaci při zavření okna
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        initComponents();
        setVisible(true);
    }

    public static void main(String[] args) {
        Swing2 okno = new Swing2();
    }
    .
    .
    .
```

Příklad - vytvoření tlačítka (2)

```
.
.
.
private void initComponents() {
    Container cp = getContentPane();
    cp.setLayout(new FlowLayout());
    BL listener = new BL();
    bt1 = new JButton("Tlacitko 1");
    bt1.addActionListener(listener);
    cp.add(bt1);
    bt2 = new JButton("Tlacitko 2");
    bt2.addActionListener(listener);
    cp.add(bt2);
    tf = new JTextField(10);
    cp.add(tf);
    pack();
}
.
.
.
```

Příklad - vytvoření tlačítka (3)

```
    .
    .
    .

    class BL implements ActionListener {

      public void actionPerformed(ActionEvent e) {
        String text = ((JButton) e.getSource()).getText();
        tf.setText(text);
      }
    }
 }
```

Rozmístění komponent

- vše je zapsáno v kódu (žádné zdroje "resources")
- umístění není řízeno absolutní polohou prvku
- o umístění rozhoduje "správce umístění"
  - rozhodnutí záleží na pořadí vložení (`add()`)
  - velikost, tvar a rozmístění závisí na typu správce

Třídy `JApplet`, `JFrame`, `JWindow` a `JDialog` poskytují objekt typu `Container` metodou `getContentPane()`. Ten obsahuje metodu `setLayout()`, která slouží k volbě správce rozmístění.

Správce rozmístění `FlowLayout`

Tato třída nechává ovladácí prvky na formulář "volně padat".

- komponenty jsou ukládány zleva doprava až zaplní celý řádek, následuje přechod na nový řádek atd.

Správce rozmístění `BorderLayout`

Implicitním chováním při neuvedení dalších instrukcí je dané komponenty umístit doprostřed a roztáhnout mezi okraji zobrazovací plochy.

Správce `BorderLayout` rozpozná čtyři oblasti a jednu středovou oblast, při použití přetížené metody `add()`, kde prvním argumentem zvolíme oblast.

- `BorderLayout.NORTH` (horní okraj)
- `BorderLayout.SOUTH` (dolní okraj)
- `BorderLayout.EAST` (pravý okraj)
- `BorderLayout.WEST` (levý okraj)
- `BorderLayout.CENTER` (střed - implicitně)

Příklad - BorderLayout

```
private void initComponents() {
  Container cp = getContentPane();
  cp.add(BorderLayout.NORTH, new JButton("Sever"));
  cp.add(BorderLayout.SOUTH, new JButton("Jih"));
  cp.add(BorderLayout.EAST, new JButton("Vychod"));
  cp.add(BorderLayout.WEST, new JButton("Zapad"));
  cp.add(BorderLayout.CENTER, new JButton("Stred"));
  pack();
}
```

Správce rozmístění `GridLayout`

- umožňuje vytvářet tabulku komponent
- komponenty ukládány zleva doprava a shora dolů unitř mřížky
- počet sloupců a řádků se určí v konstruktoru
- mřížka je rovnoměrná

Příklad - GridLayout

```
private void initComponents() {
  Container cp = getContentPane();
  cp.setLayout(new GridLayout(7, 3));
  for(int i = 0; i < 20; i++)
    cp.add(new JButton("Tlacitko " + i));
  pack();
}
```

Správce rozmístění `GridBagLayout`

- nejsložitější a nejmocnější správce rozmístění
- můžeme rozhodovat naprosto volně o umístění prvků
- určen především pro automatické generování kódu vytvořeného RAD nástroji

Správce rozmístění `BoxLayout`

- jako `GridBagLayout`
- umožňuje rozmístění vodorovné nebo svislé
- umožňuje stanovit rozteče, mechanizmus "rozpěry a tmel" (`Struts`, `Glue`)

Správce rozmístění - absolutní rozmístění

- nastavení metodou `setLayout(null)`
- využíváno některými nástroji pro tvorbu GUI, vhodnější je použití `GridBagLayout`

Příklad - načtení souboru do editoru (1)

```
public class Swing5 extends JFrame {

  JMenuBar menuBar;
  JMenu fileMenu;
  JMenuItem fileMenuOpen,
            fileMenuExit;
  JScrollPane scrollPane;
  JTextPane textPane;
  JFileChooser fileChooser;

  public Swing5() {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(640,  480);
    initComponents();
  }
    .
    .
    .
```

Příklad - načtení souboru do editoru (2)

```
    .
    .
    .
  private void initComponents() {
    Container cp = getContentPane();
    setJMenuBar(createMenuBar());

    textPane = new JTextPane();
    scrollPane = new JScrollPane(textPane);
    cp.add(scrollPane);
  }

  public static void main(String[] args) {
    JFrame okno = new Swing5();
    okno.setVisible(true);
  }
    .
    .
    .
```

Příklad - načtení souboru do editoru (3)

```
    .
    .
    .
  private JMenuBar createMenuBar() {
    menuBar = new JMenuBar();
    fileMenu = new JMenu("File");
    fileMenu.setMnemonic('F');
    fileMenuOpen = new JMenuItem("Open");
    fileMenuOpen.setMnemonic('O');

    fileMenuOpen.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        openFileAction();
      }
    });
    fileMenu.add(fileMenuOpen);
    fileMenu.addSeparator();
    .
    .
    .
```

Příklad - načtení souboru do editoru (4)

```
    .
    .
    .
    fileMenuExit = new JMenuItem("Exit");
    fileMenuExit.setMnemonic('x');

    fileMenuExit.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        System.exit(0);
      }
    });

    fileMenu.add(fileMenuExit);
    menuBar.add(fileMenu);
    return menuBar;
  }
    .
    .
    .
```

Příklad - načtení souboru do editoru (5)

```
    .
    .
    .
  private void openFileAction() {
    fileChooser = new JFileChooser();
    int retValue = fileChooser.showOpenDialog(this);
    if (retValue == JFileChooser.APPROVE_OPTION) {
      try {
        File soubor = fileChooser.getSelectedFile();
        BufferedInputStream in = new BufferedInputStream(
          new FileInputStream(soubor));
        textPane.read(in, soubor);
        in.close();
      }catch (IOException ex) {
         JOptionPane.showMessageDialog(fileChooser,
           ex.getLocalizedMessage());
      }
    }
  }
}
```

Vizuální programování

- založeno na myšlence skládání samostatných komponent
- programování tažením/skládáním komponent z příslušné palety nástrojů na fomulář a následným automatickým generováním kódu
- velmi oblíbené a rozšířené (veškeré RAD aplikace pro tvorbu GUI)
- ke zjištění potřebných informací se používá reflexe
- velmi významně urychlý tvorbu celé aplikace

JavaBeans

- objektový model vizuálního programování
- nuntno dodržovat konvence pro pojmenování metod
  - pro vlastnost nazvanou `xxx` se vytvářejí dvě metody, `getXxx` a `setXxx`. Datový typ vrácený metodou "`get`" je shodný s argumentem metody "`set`"
  - booleovských vlastností je povolena možnost nahradit metodu "`get`" metodou "`is`"
  - běžné metody se nepodřízují těmto konvencím, ale jsou **veřejné**
  - pří definici událostí postupuje obdobně jako u vlastností

Příklad - FrogBean (1)

```
class Skvrny{}

public class Frog {
  private int skoky;
  private Color barva;
  private Spots skvrny;
  private boolean skokanek;

  public int getSkoky() { return skoky;}

  public void setSkoky(int noveSkoky) {
    skoky = noveSkoky;
  }

  public Color getBarva() {return barva;}
    .
    .
    .
```

Příklad - FrogBean (1)

```
    .
    .
    .
  public void setBarva(Color novaBarva) {
    barva = novaBarva;
  }

  public boolean isSkokan() {return skokanek;}
  public void setSkokan(boolean j) {skokanek = j; }
    ...
```

Dále viz TIJ Bruce Eckel a přiložené zdroje.

Balení komponenty Bean

- vložit do standarního kontejneru `JavaBeans` což je soubor **.jar** a soubor osvědčení **.mf**, jenž informuje "Toto je komponenta `JavaBean`"

```
//FrogBean.mf
Manifest-Version: 1.0

Name: frogbean/Frog.class
Java-Bean: True
```

  - první řádek je verze schématu osvedčení (dokud Sun nezmění)
  - druhý řádek uvádí název souboru včetně balíčku
  - třetí řádek značí že jde o komponentu `JavaBean`

- balení klasicky pomocí nástroje **jar**.

```
jar cmf FrogBean.jar FrogBean.mf frogbean
```