

Seminář Java

IX

Radek Kočí

Fakulta informačních technologií VUT

4. dubna 2007

- GUI
- Zavádění tříd

Swing

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

AWT (Abstract Window Toolkit)

- starší varianta dostupná od verze 1.x.x
- od verze 1.1.x událostně řízená
- omezené možnosti

Balíčky

- `java.awt` – základní komponenty AWT GUI
- `java.awt.event` – události AWT GUI
- + další balíky v `java.awt`

- `javax.swing` – základní komponenty Swing GUI
- `javax.swing.event` – události komponenty Swing GUI
- + další balíky v `javax.swing`

Řízení programu událostmi

- obecnější pojem označující typ asynchronního programování
- základní princip tvorby GUI
- tok programu je řízen událostmi (zpracování událostí určuje běh aplikace)
- událostní aplikace by měly být programovány jako vícevláknové

Komponenty GUI

- grafické (uživatelské) elementy – tabulka, text, ...
- grafické kontejnery

Základní přístup

- událostní řízení
- kontejnery obsahují elementy a/nebo jiné kontejnery
- vzhled GUI je dán způsobem poskládání grafických elementů a kontejnerů

`http://java.sun.com/docs/books/tutorial/uiswing/components/`

Ukázka Swing aplikace

```
public class HelloWorld {
    public static void main(String[] args) {
        // vytvoreni okna aplikace
        JFrame okno = new JFrame("Hello World");

        // vytvoreni textu a vlozeni do okna
        JLabel text = new JLabel("Nazdarek ...");
        okno.getContentPane().add(text);

        // implicitni operace pri zavreni okna
        okno.setDefaultCloseOperation(
            JFrame.EXIT_ON_CLOSE);

        // otevreni okna
        okno.pack();
        okno.setVisible(true);
    }
}
```

Rozmístění komponent

- komponenty se vkládají do grafických kontejnerů
- umístění komponenty není dáno absolutní polohou
- umístění komponenty je vztaženo relativně ke kontejneru, ve kterém je vložena
- způsob umístění je dán *správce umístění*
 - velikost, tvar a rozmístění závisí na typu správce
 - záleží na pořadí vložení (**add()**)

Nastavení správce rozmístění

- v konstruktoru při vytváření kontejneru

```
converterPanel =  
    new JPanel(new GridLayout(2, 2));
```

- metodou `setLayout()` kontejneru

```
JFrame okno =  
    new JFrame("Hello World application");
```

```
okno.getContentPane().  
    setLayout(new FlowLayout());
```

FlowLayout

- komponenty jsou ukládány zleva doprava na jeden řádek
- při zaplnění řádku se přechází na nový řádek
- implicitní

GridLayout

- komponenty jsou ukládány do mřížky (tabulky)
- komponenty jsou ukládány zleva doprava a shora dolů do buněk mřížky
- počet sloupců a řádků se určuje v konstruktoru
- mřížka je rovnoměrná

GridBagLayout

- nejmocnější (také nejsložitější) správce umístění
- o umístění prvků můžeme rozhodovat naprosto volně
- využíván především při automatickém generování kódu (RAD nástroje)

BoxLayout

- vychází z **GridLayout**
- umožňuje vodorovné nebo svislé umístění
- umožňuje nastavit rozteče (mechanismus "rozpěry a tmel")

BorderLayout

- implicitně umísťuje komponenty na střed a roztahuje na celou velikost kontejneru
- lze definovat oblast vložení v přeepsané metodě `add()`
 - `BorderLayout.NORTH`
 - `BorderLayout.SOUTH`
 - `BorderLayout.EAST`
 - `BorderLayout.WEST`
 - `BorderLayout.CENTER` (implicitní)

Bez správce

- `setLayout(null)`

Řízení programu událostmi

- událost vzniká obvykle uživatelskou akcí (kliknutí, změna polohy myši, ...)
- událost může vzniknout v libovolné komponentě GUI
- každá komponenta má definovaného "posluchače" (**listener**) událostí
- při vyvolání události zašle systém zprávu posluchači – ten událost zpracuje

Události lze rozdělit podle uživatelské akce nad

- oknem – **WindowEvent**
- klávesnicí – **KeyEvent**
- myší (klikání, pohyb) – **MouseEvent**
- fokusem (získání, ztráta) – **FocusEvent**
- GUI (obecná akce, např. stisk tlačítka) – **ActionEvent**
- ... (viz [java.awt.event](#) a [java.swing.event](#))

Posluchač událostí

- objekt, jehož třída implementuje příslušné rozhraní

Rozhraní posluchačů událostí

- **ActionListener** (awt)
- **MouseListener** (awt)
- **MouseMotionListener** (awt)
- **MouseListener** (swing)
- ... (viz [java.awt.event](#) a [java.swing.event](#))

Přidání posluchače události

- komponenta musí registrovat posluchače událostí, aby příslušná událost mohla být ošetřena
- každá komponenta knihovny Swing obsahuje
 - metodu `addXXXListener()`
 - metodu `removeXXXListener()`
 - kde `XXX` reprezentuje název události (`Mouse, ...`)
- např. `addActionListener(ActionListener listener)`

Implementace posluchače událostí

- anonymní třída
- vnitřní třída
- top-level třída

```
class myActionListener implements ActionListener {  
    public void actionPerformed(ActionEvent event)  
        ...  
    }  
}
```

```
...  
convertTemp = new JButton("Convert");  
convertTemp.  
    addActionListener(new MyActionListener());  
...
```

`http://java.sun.com/docs/books/tutorial/
uiswing/learn/`

viz **CelsiusConverter**

```
button.addActionListener(new DoSomething());
```

```
class DoSomething implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        try {
            Thread.sleep(10000);
        } catch (InterruptedException e) {}
    }
}
```

GUI a paralelismus

```
class DoSomething implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        Runnable r = new Runnable() {
            public void run() {
                label.setText("Time consuming operation");
                try {
                    Thread.sleep(10000);
                } catch (InterruptedException e) {}
                label.setText("");
            }
        };
        new Thread(r).start();
    }
}
```

GUI a paralelismus

```
public static void main(String[] args) {  
    //Schedule a job for the event-dispatching thread  
    //creating and showing this application's GUI.  
    javax.swing.SwingUtilities.  
        invokeLater(new Runnable()  
    {  
        public void run() {  
            createAndShowGUI();  
        }  
    });  
}
```

`java.awt.Component`

`java.awt.Container`

`java.swing.JComponent`

Třída **JComponent**

- metoda `void paintComponent(Graphics g)`
- metoda `void repaint()`

`java.awt.Graphics`

`java.awt.Graphics2D`

- Rozhraní `java.awt.Shape`
- Grafická primitiva v balíku `java.awt.geom`
 - `Line2D.Double`, `Line2D.Float`
 - `Arc2D.Double`, `Arc2D.Float`
 - `Ellipse2D.Double`, `Ellipse2D.Float`
 - ...
- Základní operace
 - `contains()`
 - `getPathIterator(...)`


```
public void paintComponent(Graphics g) {  
    Graphics2D g2 = (Graphics2D) g;  
    ...  
    Ellipse2D.Double ellipse =  
        new Ellipse2D.Double(x,y,width,height);  
    g2.draw(ellipse);  
    ...  
}
```

<http://java.sun.com/docs/books/tutorial/uiswing/>
<http://java.sun.com/docs/books/tutorial/2d/>

JVM zavádí třídy dynamicky

Class loader

- objekt schopný zavádět třídy
- abstraktní třída `java.lang.ClassLoader`
- každá třída (`java.lang.Class`) obsahuje referenci na svůj class loader

Implementace class loaderu

- zajištění specifického chování při zavádění tříd
- více verzí stejné třídy

Spuštění aplikace:

```
java Test
```

Kroky při spouštění:

- JVM zjistí, že třída **Test** není zavedena
- JVM použije zavaděč (class loader) pro zavedení třídy **Test**

Proces zavádění třídy:

- načtení byte-code (loading)
- linkování (linking)
- inicializace (initialization)

Loading

- vyhledání a načtení byte-code třídy
- různé zavaděče mají různou politiku (souborový systém, http, ...)

Linking

- operace nutné pro to, aby byla třída použitelná
 - verification
 - verifikace korektnosti binární reprezentace třídy
 - preparation
 - vytváří statické členy třídy, implicitní inicializace
 - resolution (optional)
 - závislosti na jiných třídách

Initialization

- explicitní inicializace statických členů třídy
- před inicializací se musí inicializovat nadřazená třída (pokud ještě není)
- k inicializaci třídy/rozhraní **T** dochází před akcí:
 - **T** je třída a instance **T** je vytvářena
 - **T** je třída a její statická metoda je volána
 - statická proměnná deklarovaná v **T** je přiřazena
 - statická proměnná deklarovaná v **T** je použita

Zavádění tříd

```
class Super {
    static { System.out.print("Super "); }
}
class One {
    static { System.out.print("One "); }
}
class Two extends Super {
    static { System.out.print("Two "); }
}
class Test {
    public static void main(String[] args) {
        One o = null;
        Two t = new Two();
        System.out.println((Object)o == (Object)t);
    }
}
```

⇒ `Super Two false`

Třída **One** nebude nikdy linkována (není aktivně použita).

Zavádění tříd

```
class Super { static int taxi = 1729; }

class Sub extends Super {
    static { System.out.print("Sub "); }
}

class Test {
    public static void main(String[] args) {
        System.out.println(Sub.taxi);
    }
}
```

1729 (Sub není inicializována)

Zavádění tříd

```
class Super { static int taxi = 1729; }

class Sub extends Super {
    static { System.out.print("Sub "); }
}

class Test {
    public static void main(String[] args) {
        System.out.println(Sub.taxi);
    }
}
```

1729 (Sub není inicializována)

Zavádění tříd

```
class Super { static int taxi = 1729; }

class Sub extends Super {
    static { System.out.print("Sub "); }
}

class Test {
    public static void main(String[] args) {
        Sub s = new Sub();
        System.out.println(Sub.taxi);
    }
}
```

Sub 1729

Zavádění tříd

```
class Super { static int taxi = 1729; }

class Sub extends Super {
    static { System.out.print("Sub "); }
}

class Test {
    public static void main(String[] args) {
        Sub s = new Sub();
        System.out.println(Sub.taxi);
    }
}
```

Sub 1729

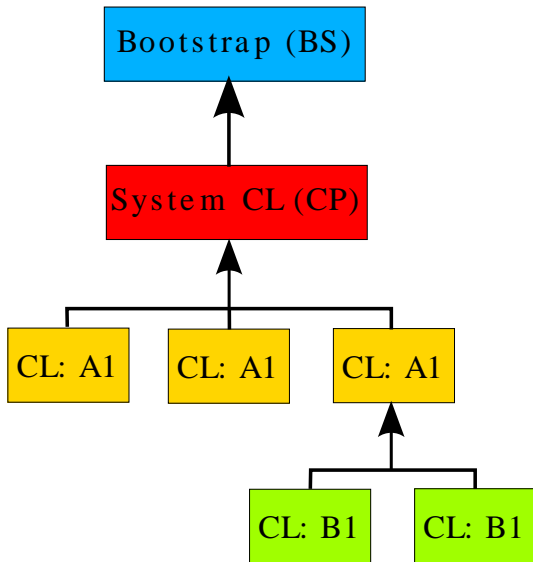
Zavaděče tříd (class loaders)

- vytvářejí hierarchickou strukturu
- každý zavaděč má svého předka (kromě vrcholového)
- zavaděč deleguje požadavky na svého předka
- každá třída je definována svým typem a svým zavaděčem
- každá třída je zavedena pouze jednou
- každá třída je uchována v zavaděči

Hierarchie zavaděčů

- **bootstrap loader**
zavádí základní třídy JVM (**jr.jar**) + **extensions**
- **system loader**
zavádí třídy dostupné přes **CLASSPATH** (nebo **-classpath**)

Hierarchie zavaděčů



Metody třídy `java.lang.ClassLoader`

- `loadClass()`
 - test, zda už je třída zavedena (`findLoadedClass`)
 - deleguje zavádění (`loadClass`)
 - *fail* ⇒ načte třídu (`findClass`)
- `findClass()`
 - čte byte-code reprezentující třídu
 - vytváří třídu (`defineClass`)
 - *odvozené zavaděče by měly předefinovat tuto metodu*
- `defineClass()`
 - jako parametr má pole bytů
 - konvertuje pole bytů na třídu
- `resolveClass()`
 - ověření symbolický referencí ze zaváděné třídy
 - zavedení závislých tříd (volitelné)

Zavádění tříd

```
protected Class findClass(String pClassName)
throws ClassNotFoundException {
    try {
        File lClassFile = ...
        InputStream lInput = ...
        ByteArrayOutputStream lOutput = ...
        int i = 0;
        while ((i = lInput.read()) >= 0) {
            lOutput.write( i );
        }
        byte[] lBs = lOutput.toByteArray();
        return defineClass(pClassName, lBs, 0,
                          lBs.length);
    } catch (Exception e) {
        throw new ClassNotFoundException(...);
    }
}
```

Uvolnění objektu (*garbage collecting*)

- pokud již neexistuje reference na objekt z živého vlákna
- finalizace objektu

Třída může být uvolněna:

- pokud její zavaděč (class loader) podlehe **garbage collectoru**
- třídy zavedené *bootstrap* zavaděčem nemohou být nikdy uvolněny
- třída *nemůže* být uvolněna, pokud je její zavaděč potenciálně dostupný

- Třídy jsou zaváděny dynamicky
- Každá třída má svůj zavaděč
- Existují implicitní zavaděče
- Každá třída "dědí" zavaděč třídy, ze které bylo vyvoláno zavedení
- Je možné definovat vlastní zavaděče