Prednáška 7: Grafické používateľské rozhranie v Jave a vzor Model-View-Controller

Objektovo-orientované programovanie 2012/13

Valentino Vranić

Ústav informatiky a softvérového inžinierstva
Fakulta informatiky a informačných technológií
Slovenská technická univerzita v Bratislave

10. apríl 2013

Obsah prednášky

Grafické používateľské rozhranie v Jave

Používateľské rozhranie

- Používateľské rozhranie umožňuje interakciu s používateľom
- Príkazový riadok (command line interface, CLI)
- Grafické používateľské rozhranie (graphical user interface, GUI)
- Prevládajú systémy založené na oknách
- WIMP štýl (window, icon, menu, pointing device) vyvinutý v PARC a popularizovaný Macintoshom (1984)[1]
- Iné prístupy – v mobilných zariadeniach (senzory), v počítačových hrách, virtuálnej realite...

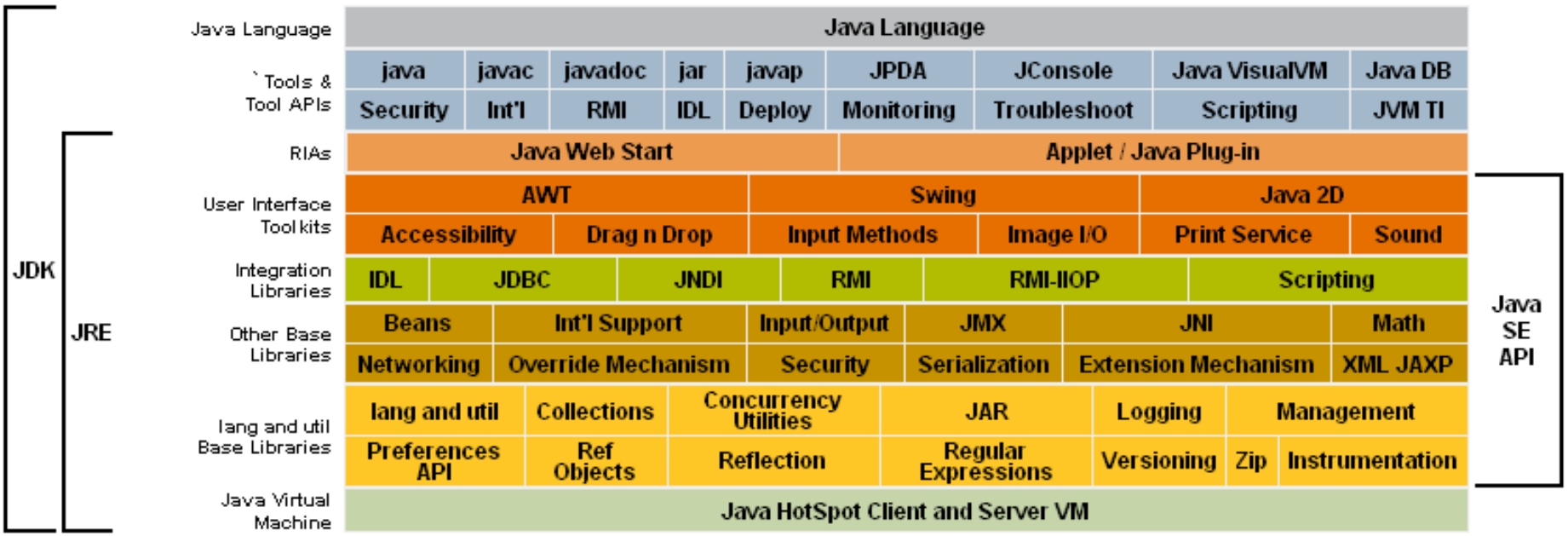
[1] http://en.wikipedia.org/wiki/History_of_the_GUI

Swing

- Java Foundation Classes – obsahujú Swing, AWT a Java2D
- Swing – aplikačný rámec (framework) pre grafické používateľské rozhranie (GUI)
- AWT (Abstract Window Toolkit) – definuje model udalosti
  - Aplikačný rámec pre GUI využívajúci prvky GUI, ktoré poskytuje daný operačný systém
  - Značné obmedzenia v zmysle GUI, ale vhodný model udalostí, na ktorom je postavený aj Swing
- JavaBeans – model komponentov
- SWT (Standard Widget Toolkit, Eclipse) – alternatíva k Swingu[2]

[2] http://www.eclipse.org/swt/

JFC v Java SE



`http://download.oracle.com/javase/7/docs/`

Editory pre GUI

- Využitie editorov pre GUI je možné vďaka tomu, že je Swing založený na komponentovom prístupe nazývanom JavaBeans
  - JavaBean je reprezentovaný triedou (transparentné z hľadiska OO prístupu)
  - Musí byť dodržaná konvencia pomenovania (set/get) a prístupnosti metód a atribútov (označených ako *properties* – vlastnosti)
  - JavaBean je riadený udalosťami (events)
- Podpora je možná priamo v integrovanom vývojovom prostredí (napr. v NetBeans a Eclipse)
- Jestvujú aj samostatné generátory GUI[3]

[3] http://www.fullspan.com/articles/java-gui-builders.html

Swing komponenty

- Kontejnery – odvodené od `JContainer`
  - Poskytujú priestor pre iné komponenty
  - Do kontejnerov možno pridávať komponenty
  - Komponent v kontejneri sa označuje ako jeho potomok – *child*
  - Rozloženie komponentov riadi *layout manager*
- Atomické komponenty – odvodené od `JComponent`
  - Umožňujú interakciu s používateľom
  - Typické komponenty sú tlačidlá, textové polia, označenia (*labels*) a pod.

Swing kontejnery

- Kontejnery na vrchnej úrovni (*top-level containers*):
  - `JFrame` (a `JWindow`) – okná
  - `JDialog` – dialógy
  - `JApplet` – applety
- Sprostredkovateľské kontejnery (*intermediate containers*):
  - `JPanel`, `JScrollPane`, `JSplitPane`, `JTabbedPane`, `JToolBar`, `JInternalFrame`, `JLayeredFrame`, `JRootPane`
- `JPanel` sa dá využiť aj na priame kreslenie grafických útvarov (príklad v TiJ, Kapitola 14, *A catalog of Swing components*, *Drawing*)

Swing dialógy

- Okrem tvorby vlastného dialogu pomocou triedy `JDialog` možno využiť predpripravené štandardné dialogy:
  - `JOptionPane`
  - `JFileChooser`
  - `JColorChooser`
  - `JProgressBar` a `ProgressMonitor`
- Takéto dialógy sú *modálne* – musia sa ukončiť, aby mohla pokračovať ďalšia interakcia používateľa so zvyškom GUI

Tvorba okien

- Okná sa implementujú pomocou triedy `JFrame`, napr.:

```
JFrame w = new JFrame("Moje okno");
```

- Lepšie je však odvodiť vlastné okno dedením od triedy `JFrame`
- Trieda `JWindow` predstavuje tiež okná, ale „holé“

Príklad: jednoduché okno

```
import javax.swing.*;

class C {
    public static void main(String[] args) {
        JFrame w = new JFrame("Moje okno");
        w.setSize(300, 300);
        w.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        w.setVisible(true);
    }
}
```

Príklad: okno dedením od JFrame

```
import javax.swing.*;

class MyWindow extends JFrame {
    public MyWindow() {
        setTitle("Moje okno");
        setSize(300, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
class C {
    public static void main(String[] args) {
        JFrame w = new MyWindow();
        w.setVisible(true);
    }
}
```

Pridávanie komponentov do JFrame

- Komponenty sa vytvárajú podobne ako kontejnery:

```
JButton b1 = new JButton("Tlacidlo1");
```

- Ak sa vytvára viac podobných, špecializovaných komponentov (napr. podobné tlačidlá) je vhodné použiť dedenie
- Do `JFrame` možno pridať komponenty metódou `add()`:

```
w.add(b1);
```

- Komponenty sa vlastne pridávajú do objektu typu `JPanel` daného `JFrame`
- Do JDK 5 sa to muselo písať explicitne:

```
w.getContentPane().add(b1);
```

Príklad: okno s tlačidlom (1)

```
import javax.swing.*;

class C {
    public static void main(String[] args) {
        JFrame w = new JFrame("Moje okno");
        w.setSize(300, 300);
        JButton b1 = new JButton("Tlacidlo1");
        w.add(b1);
        w.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        w.setVisible(true);
    }
}
```

Príklad: okno s tlačidlom (2)

Rozloženie prvkov v okne

- Swing umožňuje nastavovanie rozloženia prvkov pomocou `LayoutManagerov`
- Najčastejšie používané rozloženia:
  - `FlowLayout` – prednastavené pre `JPanel`
  - `BorderLayout` – prednastavené pre `contentPane`
  - `GridLayout`
  - `BoxLayout`
  - `GridBagLayout`
- Je možné aj absolútne umiestňovanie (*null layout*) – nie je vhodné

Príklad: rozloženie prvkov v okne (1)

```
import javax.swing.*;
import java.awt.*;

class C {
   public static void main(String[] args) {
      JFrame w = new JFrame("Moje okno");
      w.setSize(300, 300);
      JButton b1 = new JButton("Tlacidlo1");
      JButton b2 = new JButton("Tlacidlo2");
// prednastavené rozloženie:
// w.setLayout(new BorderLayout());
      w.add(b1, BorderLayout.WEST);
      w.add(b2, BorderLayout.EAST);
      w.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      w.setVisible(true);
   }
}
```

Príklad: rozloženie prvkov v okne (2)

Poznámka ohľadom appletov

- Applet – „malá“ aplikácia, ktorá sa spúšťa v rámci webového prehliadača
- Odvodené od triedy `JApplet`
  - Takmer všetky príklady v Kapitole 14 v TiJ sa týkajú appletov
- Výhoda: možno ich spúšťať cez web prostredníctvom webového prehliadača
- Obmedzenia týkajúce sa lokálneho disku
- Problémy s prenositeľnosťou (a iné)[4]

[4] `http://www.leepoint.net/notes-java/10background/10applications_and_applets/70applets.html`

Spracovanie udalosti vo Swingu

Spracovanie udalosti

- Swing komponenty generujú udalosti (*events*) vo forme objektov
- Aby trieda mohla spracovávať udalosti (*event handling*), musí implementovať zodpovedajúce rozhranie prijímača (*listener*), napr.:

```
public class MyListner implements ActionListener {
  ...
}
```

- Trieda potom musí implementovať metódy rozhrania, napr.:

```
public void actionPerformed(ActionEvent e) {
  ...
}
```

Spracovanie udalosti (2)

- Objekt takejto triedy treba registrovať ako prijímač udalostí pre príslušný komponent, napr.:

```
tlacidlo1.addActionListener(new MyListner());
```

- Príklady udalosti a zodpovedajúcich rozhraní:
  - Kliknutie tlačidla, výber položky z menu, stlačenie enter pri zadávaní textu – `ActionListener`
  - Zavretie hlavného okná – `WindowListener`
  - Kliknutie tlačidla myši nad komponentom – `MouseListener`
  - Posun myšou nad komponentom – `MouseMotionListener`
- Práca so zoznamami prijímačov je bezpečna z hľadiska odosielacej nite Swingu

Príklad: kliknutie na tlačidlo

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*; // niekedy aj javax.swing.event.*

class MyWindow extends JFrame {
    private JButton t = new JButton("Tlacidlo1");
    public MyWindow() {
        setSize(300, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new FlowLayout());
        add(t);
        t.addActionListener(new MyListener());
    }
...
```

Príklad: prijímač kliknutia na tlačidlo (2)

```
    // trieda vhniezdená v MyWindow
    private class MyListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            if (t.getText() != "XXX")
                t.setText("XXX");
            else
                t.setText("YYY");
        }
    }
} // class MyWindow

class C {
    public static void main(String[] args) {
        JFrame w = new MyWindow();
        w.setVisible(true);
    }
}
```

Príklad: prijímač pomocou anonymnej triedy

```
class MyWindow extends JFrame {
    private JButton t = new JButton("XXX");
    public MyWindow() {
        setSize(300, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new FlowLayout());
        add(t);
        t.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                if (t.getText() != "XXX")
                    t.setText("XXX");
                else
                    t.setText("YYY");
            }
        });
    }
}
```

Príklad: sledovanie pohybu myši

```
t.addMouseListener(new MouseListener() {
    public void mouseEntered(MouseEvent e) {
        if (t.getText() != "XXX")
            t.setText("XXX");
        else
            t.setText("YYY");
    }
    public void mouseReleased(MouseEvent e) {
    }
    public void mousePressed(MouseEvent e) {
    }
    public void mouseExited(MouseEvent e) {
    }
    public void mouseClicked(MouseEvent e) {
    }
});
```

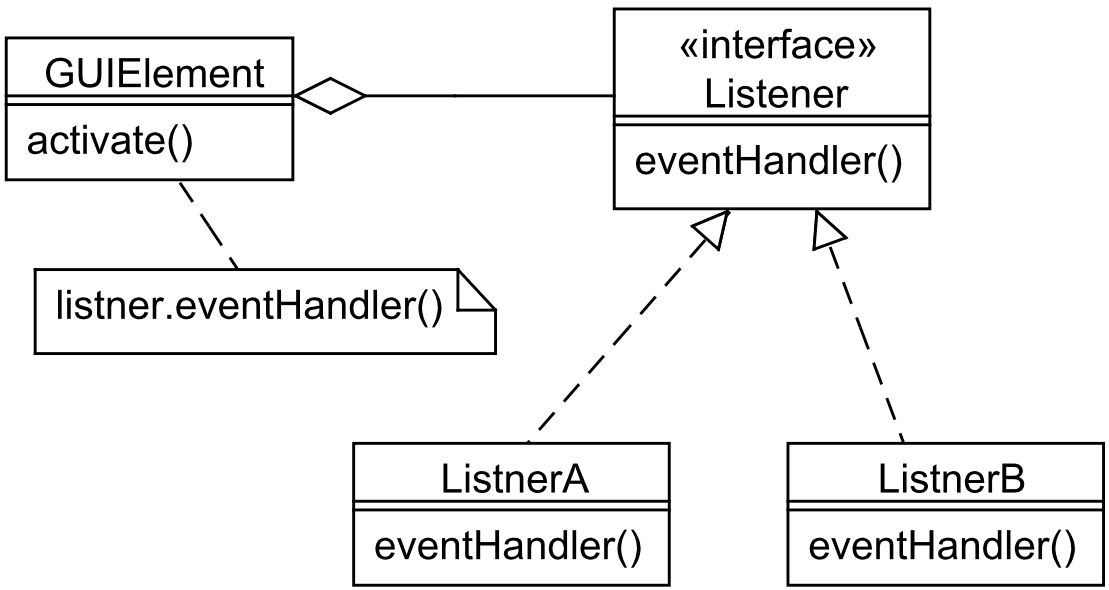
Využitie adaptérov na spracovanie udalosti

- Niektoré rozhrania prijímačov predpisujú viac metód pre rôzne udalosti
- Často však sledujeme len jednu alebo dve udalosti
- Dajú sa pritom využiť *adaptéry* – triedy, ktoré implementujú všetky metódy daného rozhrania prijímača ako prázdne
- Metódu pre príslušnú udalosť jednoducho prekonáme, ostatné necháme tak ako sú

Príklad: použitie adaptéra

```
t.addMouseListener(new MouseAdapter() {
    public void mouseEntered(MouseEvent e) {
        if (t.getText() != "XXX")
            t.setText("XXX");
        else
            t.setText("YYY");
    }
});
```

Spracovanie udalosti vo Swingu – schéma



Niť na odosielanie udalosti vo Swingu

Pravidlo jednej nite

- Swing event-dispatching thread
- Pravidlo jednej nite (Single-Thread Rule):[5]
  - Po realizácii komponentu Swingu všetok kód, ktorý by mohol vplývať na stav tohto komponentu alebo závisieť od neho sa má vykonať v niti na odosielanie udalosti
- Pri porušení hrozí narábanie s nekonzistentným stavom a uviaznutie
- Realizácia komponentu znamená, že bola zavolaná jeho metóda `paint()`
- Pre okno to znamená zavolanie `setVisible(`**`true`**`)`, `show()` alebo `pack()`
- Niektoré metódy je bezpečné volať mimo odosielacej nite (v dokumentácii Swing API)

[5] `http://java.sun.com/products/jfc/tsc/articles/threads/threads1.html`

Ako dodržať pravidlo jednej nite

- Kód, ktorý pracuje so stavom komponentov vo Swingu, treba vykonávať prostredníctvom odosielacej nite Swingu
- Taký kód treba zabaliť do vykonateľného objektu (`Runnable`) a zaradiť na vykonávanie prostredníctvom volaní:
  - `invokeLater()` – metóda sa vráti hneď po zaradení kódu
  - `invokeAndWait()` – metóda sa vráti až keď odosielacia niť vykoná kód

Príklad: korektná zmena GUI po realizácii (1)

```
class MyWindow extends JFrame {
    private JLabel l;

    public MyWindow() {
        setTitle("Moje okno");
        setSize(300, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        l = new JLabel("Text");
        add(l);
    }
    public void changeText(String s) {
        l.setText(s);
    }
}
```

Príklad: korektná zmena GUI po realizácii (2)

```
class C {
    public static void main(String[] args) throws Exception {
        final MyWindow w = new MyWindow();
        w.setVisible(true);

        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                w.changeText("Novy text");
            }
        });
    }
}
```

Swing a viacniťovosť

- Kód, ktorý sa spúšťa prostredníctvom GUI, sa vo Swingu spúšťa vlastne prostredníctvom prjímačov
- To znamená, že prebehne v rámci odosielacej nite Swingu
- Ak ide o zložitejšiu operáciu, celé GUI bude blokované

Príklad: spustenie náročného výpočtu z GUI (1)

```
public class NastyComputation {
    private MyWindow w;

    public void setWindow(MyWindow w) {
        this.w = w;
    }
    public void compute() {
        w.setBusy();
        for (int i = 0; i < 100000; i++)
            System.out.println("c");
        w.setFree();
    }
}
```

Príklad: spustenie náročného výpočtu z GUI (2)

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class MyWindow extends JFrame {
    private NastyComputation c;
    private JButton PressButton = new JButton("Press");
    private JButton CompButton = new JButton("Comp");
    private JLabel l = new JLabel("Free");
    public JPanel p = new JPanel();
    ...
```

Swing a viacniťovosť

```
...
public MyWindow(NastyComputation c) {
    super("MyWindow");
    this.c = c;
    setSize(200, 75);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLayout(new FlowLayout());
    add(PressButton);
    add(CompButton);
    add(l);
    ...
```

Príklad: spustenie náročného výpočtu z GUI (3)

```
    ...
    PressButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            if (PressButton.getText() != "Press") {
                PressButton.setText("Press");
            }
            else {
                PressButton.setText("Again");
            }
        }
    });
    CompButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            MyWindow.this.c.compute();
        }
    });
} // MyWindow(NastyComputation c)
```

Príklad: spustenie náročného výpočtu z GUI (4)

```
    public void setBusy() {
        l.setText("Busy");
    }
    public void setFree() {
        l.setText("Free");
    }

    public static void main(String[] args) {
        NastyComputation c = new NastyComputation();
        MyWindow w = new MyWindow(c);
        c.setWindow(w);
        w.setVisible(true);
    }
} // MyWindow
```

Príklad: spustenie náročného výpočtu z GUI (6)

- GUI sa zablokuje
- JLabel sa neaktualizuje

Príklad: spustenie náročného výpočtu z GUI (7)

- Riešenie je logické: spustiť danú operáciu vo vlastnej niti

```
public class NastyComputation {
    private MyWindow w;

    public void setWindow(MyWindow w) {
        this.w = w;
    }
    public void compute() {
        w.setBusy();
        new Thread() {
            public void run() {
                for (int i = 0; i < 100000; i++)
                    System.out.println("c");
                SwingUtilities.invokeLater(new Runnable() {
                    public void run() {
                        w.setFree();
                    }
                });
            }
        }.start();
    }
}
```

Príklad: spustenie náročného výpočtu z GUI (8)

- Erudovanejšie riešenie berie do úvahy, že Swing je na takéto veci pripravený[6]
- Výpočet zbavíme závislosti od GUI:

```
public class NastyComputation {
    public void compute() {
        for (int i = 0; i < 10000; i++)
            System.out.println("c");
    }
}
```

[6] http://download.oracle.com/javase/tutorial/uiswing/concurrency/

Príklad: spustenie náročného výpočtu z GUI (9)

- Upravíme prjímač tlačidla na spustenie výpočtu:

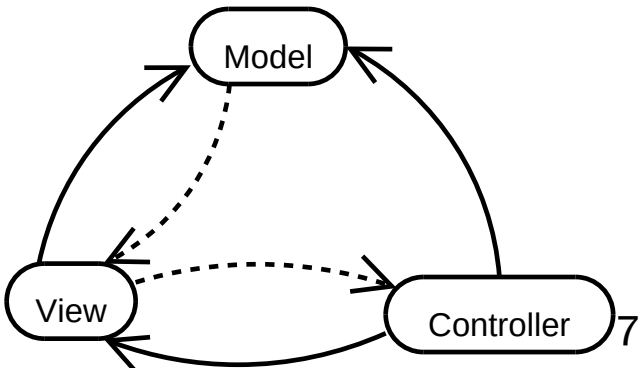
```
CompButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        setBusy();
        new SwingWorker<Object, Object>() {
            public Object doInBackground() {
                c.comp();
                return null;
            }
            protected void done() {
                setFree();
            }
        }.execute();
    }
});
```

Vzor Model-View-Controller

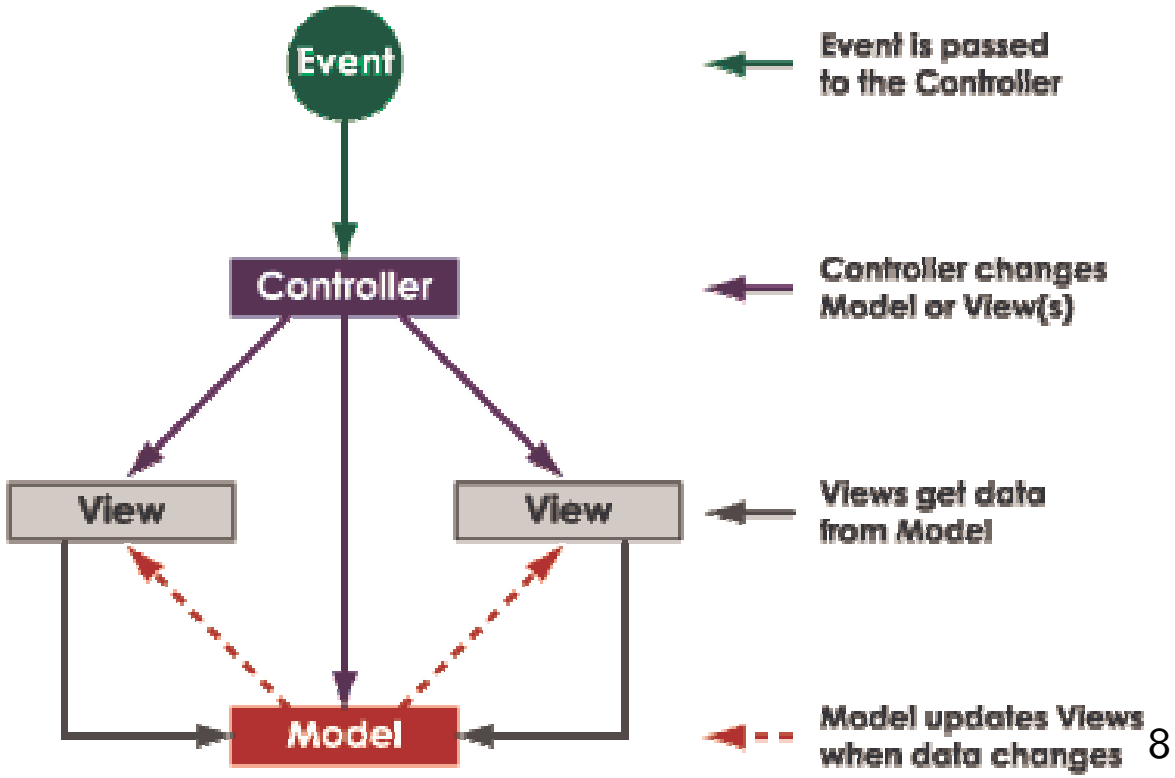
Model-View-Controller (MVC)

- Architektonický vzor
- Ďalšie druhy vzorov (patterns) zahŕňajú návrhové vzory, idiómy a analytické vzory
- MVC pochádza od Trygve M. H. Reenskauga, 1979; prvé použitie v Smalltalku
- Predstavuje základ pre GUI
- Základné pojmy:
  - *Model* – model spracovávanej oblasti (aplikačná logika a údaje)
  - *View* – pohľad na model
  - *Controller* – riadenie modelu a pohľadu: zabezpečenie reakcie aplikácie na udalosti

Model-View-Controller (2)



[7]



[8]

[7] http://ootips.org/mvc-pattern.html

[8] http://www.enode.com/x/markup/tutorial/mvc.html

Použitie MVC

- Swing ako rámec (*framework*) je implementovaný podľa vzoru MVC (modifikovaného[9])
- Vo vlastnej aplikácii treba minimalizovať zviazanie (*coupling*) medzi modelom, pohľadom a riadením[10]
- Model by mal byť čím nezávislejší od pohľadu a riadenia
  - Model by nemal poznať detaily pohľadu a riadenia
  - Niekedy je notifikácia zo strany modelu nevyhnutná
- Pre menšie aplikácie môže byť vhodné spojenie pohľadu a riadenia do jedného celku – vzor *Presentation-Model*[11]

[9] http://java.sun.com/products/jfc/tsc/articles/architecture/
[10] http://www.leepoint.net/notes-java/GUI/structure/40mvc.html
[11] http://www.leepoint.net/notes-java/GUI/structure/30presentation-model.html

Návrhové vzory v MVC (1)

- Architektonický vzor Model-View-Controller sa skladá predovšetkým z troch návrhových vzorov:[12]

$$MVC = Observer + Strategy + Composite$$

[12] E. Gamma et al. Design Patterns: Elements of Reusable Object-Oriented Design. Addison Wesley, 1995.

Návrhové vzory v MVC (2)

- Observer: vzťah Model–View (Model = Subject, View = Observer)
- Strategy: vzťah View–Controller
- Composite: vhniezdené pohľady (View)

Sumarizácia

Sumarizácia

- Rámec Swing
- Swing komponenty: kontejnery (sprostredkovateľské a na vrchnej úrovni) a atomické komponenty
- Spracovanie udalosti vo Swingu – pozor na pravidlo jednej nite
- Vzor MVC

Čítanie

- Dnešná prednáška: OJA, kapitola 13 a časť 15.2
- Kapitola 14 v TiJ podľa potrieb projektu
- Ďalšia literatúra citovaná v prednáške (dostupná na webe)
- Ďalšia prednáška bude o OO modelovaní a jazyku UML – kapitola 14 v OJA