Swing

Vlákna

Přehled

- obsluha událostí a vykreslování GUI
  - **jedno** vlákno (*event-dispatching thread*)
  - zajišťuje postupné obsluhování událostí
    - každá událost se obslouží až po skončení předchozí obsluhy
    - události nepřerušují vykreslování
- SwingUtilities.invokeLater(Runnable doRun)
- SwingUtilities.invokeAndWait(Runnable doRun)
- SwingUtilities.isEventDispatchingThread()
  - test, zda aktuální vlákno je *event-dispatching thread*
- obsluha události
  - nesmí trvat dlouho!
  - pokud trvá dlouho → do zvláštního vlákna

SwingWorker<T, V>

- třída pro obsluhu dlouhotrvajících událostí
- součást JDK až od verze 6
  - pro starší JDK nutno stáhnout

- abstraktní třída
  - nutno naimplementovat metodu
    protected abstract T doInBackground()
    - provede dlouhotrvající činnost
  - metoda execute() spustí nové vlákno provede v něm metodu doInBackground()

SwingWorker<T, V>

```
public void actionPerformed(ActionEvent e) {
   ...
   final SwingWorker<Object, Object> worker =
                              new SwingWorker<Object, Object>(){
     public Object doInBackground() {
        ...
        return someValue;
     }
   };
   worker.execute();
   ...
}
```

- doInBackground() vrací hodnotu
  - lze ji získat metodou get()
    - pokud činnost neskončila, volání se zablokuje
- metoda done()
  - volána po skončení doInBackground()
  - provede se ve vláknu obsluhujícím GUI **(!)**

SwingWorker<T, V>

- typové parametry
  - T
    - typ návratové hodnota z workeru
  - V
    - typ pro předávání dat z workeru během výpočtu
      - *Intermediate results (průběžné výsledky)*
    - protected void publish(V... chunks)
      - **„odesílá“** data
      - volá se z doInBackground()
    - protected void process(List<V> chunks)
      - zpracovává předaná data
      - určena k přeimplementovaní v potomku
      - volá se ve vláknu obsluhujícím GUI **(!)**
- stav wokeru
  - public SwingWorker.StateValue getState()
    - hodnoty PENDING, STARTED, DONE

SwingWorker<T, V>

- aktuální postup výpočtu
  - int getProgress()
  - void setProgress(int progress)
    - nenastavuje se automaticky
    - nutno ručně volat z doInBackground()
      - ale není to nutné
- addPropertyChangeListener(PropertyChangeListener listener)
  - listener pro změnu stavu a postupu výpočtu
- zrušení background workeru
  - metoda cancel()
    - doInBackground() musí spolupracovat pomocí metody *isCancel();*

Swing Timer

- třída javax.swing.Timer
  - Vykonání činnosti později, opakovaně
- časovač pro použití s GUI
  - vhodné použít, pokud naplánovaná činnost obsluhuje GUI – existuje přímo vyhrazené Timer vlákno, které kooperuje s event-dispatch vláknem
  - "normální" Timer by se neměl používat pro obsluhu GUI
- vytvoření
  - Timer(int delay, ActionListener listener)
- **Action listener** provede činnost ve vláknu obsluhujícím GUI **(!)**
- metody
  - start(), stop()
  - setRepeats(boolean b) – implicitně true

Swing

Vlastní kreslení

Přehled

- u GUI komponenty předefinovat metodu
  `public void paintComponent(java.awt.Graphics g)`

- Graphics
  - poskytuje metody pro kreslení
  - obvykle instance potomka Graphics2D

```
class MyPanel extends JPanel {

  public void paintComponent(Graphics g) {
    super.paintComponent(g);
    g.drawString("This is my custom Panel!",10,20);
  }
}
```

Přehled

- lze předefinovat u jakékoliv komponenty
  - obvykle se používá JPanel
    - např. pro vytváření her
  - lze i ostatní komponenty
    - např. tlačítka apod.
  - lze dědit i přímo od JComponent
- metoda **`paintComponent()`** se volá automaticky, když je potřeba
- explicitně lze požádat o překreslení zavoláním metody **`repaint()`**
  - nezavolá **`paintComponent()`** přímo, ale
  - dá fronty událostí požadavek na překreslení
    - při více požadavcích za sebou se překreslí jen jednou

Přehled

- repaint() existuje v několika verzích
  - bez parametrů
    - překreslení celé komponenty
  - s parametry
    - překreslení jen daného obdélníku
- „odbočka“
  - mechanizmus vykreslování převzat (a upraven) z AWT
  - v AWT – vlastní vykreslovaní přes metody paint() a update()
    - defaultní implementace – update() volá paint()
  - ve Swingu – z paint() se zavolá paintComponent()
    - a ještě metody paintBorder() a paintChildren()
      - ty ale není obvykle potřeba předefinovat

Swing

Práce s obrázky

Přehled

- základní třída (ještě z AWT)
  java.awt.Image
- předpoklad (z dob JDK 1.0) – obrázky se stahují po síti
- získání obrázku
  - applet
    - metoda getImage()
  - aplikace
    - Toolkit.getDefaultToolkit().getImage()
- vykreslení
  - g.drawImage() // Graphics g;

- podpora GIF, PNG, JPG

Příklad

```
import javax.swing.*;
import java.awt.*;
public class ShowImage extends JApplet {
  private Image im;
  public void init()  {
    im = getImage( getDocumentBase(), "ball.gif");
  }
  public void paint(Graphics g)  {
    g.drawImage(im, 0, 0, this);
  }
}
```

- problém
  - getImage() obrázek nenatahuje, pouze alokuje paměť
  - obrázek natahuje až drawImage() v průběhu vykreslování

Vykreslování

- Graphics.drawImage(Image img, int x, int y, ImageObserver observer)
  - ImageObserver
    - monitoruje nahrávání obrázku
    - opakovaně se volá imageUpdate()
      - implicitní chování je zavolání repaint()
    - JApplet i JFrame implementují ImageObserver
- MediaTracker třída
  - „přednatažení“ obrázků

```
public void init() {
   im = getImage(getDocumentBase(), "ball.gif");
   MediaTracker tracker = new MediaTracker(this);
   tracker.addImage(im, 0);
   try {
     tracker.waitForID(0);
   } catch (InterruptedException e)  {
     System.out.println("Download Error");
   }
}
```

ImageIcon

- kombinace Image a ImageTracker
  im = new ImageIcon(getDocumentBase()+"ball.gif").getImage();
- lze použít pro jakékoliv obrázky
  - ne nutně jen ikony (malé obrázky)
- typické použití v aplikacích
  im = new ImageIcon(getClass().getResource("ball.gif")).getImage();

Java 2D API

- přidáno v pozdějších verzích
- rozšíření grafických operací
- základní třída
  java.awt.Graphics2D
  - potomek java.awt.Graphics
  - metoda paintComponent() má stále parametr „jen“ typu Graphics
    => musí se explicitně přetypovat
    - lze v podstatě vždy
  - u aktivního vykreslování (viz dále ve slidech)
    - návratovou hodnotu getGraphics() také přetypovat na Graphics2D
  - poskytuje více operací než Graphics
  - lépe se používá

BufferedImage

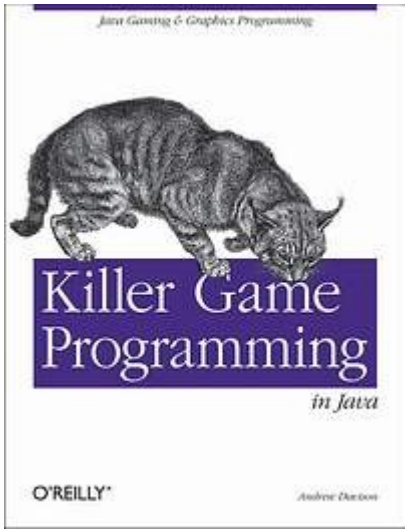
- potomek Image
  - balíček java.awt.image
- snadný přístup k datům obrázků
- automaticky se převádějí na *managed image*, které umožňují používat HW akceleraci
- nahrávání pomocí javax.imageio.ImageIO.read()
  - mělo by být rychlejší než ImageIcon
- operace nad BufferedImage
  - třídy implementující java.awt.image.BufferedImageOp
  - různé transformace
    - AffineTransformOp, ColorConvertOp,...

Swing

Vykreslování ve hrách

Přehled

- příklady převzaty z knihy
  A. Dawison: ***Killer Game Programming in Java***
  - kniha volně ke stažení na
    http://fivedots.coe.psu.ac.th/~ad/jg/
    - není to finální verze knihy
    - jsou zde ale i některé kapitoly navíc
  - kniha existuje i v českém překladu
    - Programování dokonalých her v Javě



Příklad 1


```
public class GamePanel extends JPanel implements Runnable {
  private static final int PWIDTH = 500;
  private static final int PHEIGHT = 400;
  private Thread animator;
  private boolean running = false;
  private boolean gameOver = false;
        :
  public GamePanel()  {
    setBackground(Color.white);
    setPreferredSize( new Dimension(PWIDTH, PHEIGHT));
    ...
  }
  public void addNotify()  {
    super.addNotify();
    startGame();
  }
  private void startGame()  {
    if (animator == null || !running) {
      animator = new Thread(this);
      animator.start();
    }
  }
```

Příklad 1


```
...
public void stopGame() { running = false;    }

public void run() {
  running = true;
  while(running) {
    gameUpdate();
    gameRender();
    repaint();
    try {
      Thread.sleep(20);
    } catch(InterruptedException ex) {}
  }
  System.exit(0);
}
private void gameUpdate() {
  if (!gameOver)
    ...
}
...
}
```

Příklad 1 Vykreslování

- použití „double buffering“
  - kreslení do bufferu mimo obrazovku
  - překopírování bufferu naráz na obrazovku

```
private Graphics dbg;
private Image dbImage = null;
     :
private void gameRender() {
  if (dbImage == null){
    dbImage = createImage(PWIDTH, PHEIGHT);
    if (dbImage == null) {
      System.out.println("dbImage is null");
      return;
    } else
      dbg = dbImage.getGraphics();
  }
  dbg.setColor(Color.white);
  dbg.fillRect (0, 0, PWIDTH, PHEIGHT);
    ...
    ...
    if (gameOver)
      gameOverMessage(dbg);
} // end of gameRender()
private void
gameOverMessage(Graphics g) {
  g.drawString(msg, x, y);
}
```

- překopírování bufferu v paintComponent()

```
public void paintComponent(Graphics g) {
  super.paintComponent(g);
  if (dbImage != null) {
    g.drawImage(dbImage, 0, 0, null);
  }
}
```

Příklad 1 Vstup

- přidání reakce na vstup od uživatele

```
public GamePanel() {
  setBackground(Color.white);
  setPreferredSize( new Dimension(PWIDTH, PHEIGHT));

  setFocusable(true);
  requestFocus();
  readyForTermination();
  ...
  addMouseListener( new MouseAdapter() {
    public void mousePressed(MouseEvent e) {
      testPress(e.getX(), e.getY()); }
  });
}
```

Příklad 1 Vstup

```
private void readyForTermination()  {
    addKeyListener( new KeyAdapter() {
     public void keyPressed(KeyEvent e)  {
       int keyCode = e.getKeyCode();
       if ((keyCode == KeyEvent.VK_ESCAPE) ||
           (keyCode == KeyEvent.VK_Q) ||
           (keyCode == KeyEvent.VK_END) ||
           ((keyCode == KeyEvent.VK_C) && e.isControlDown()) ) {
         running = false;
       }
     }
   });
}

private void testPress(int x, int y) {
  if (!gameOver) {
    ...
  }
}
```

Příklad 1 Problémy

- proměnné `running` a `gameOver` by měly být `volatile`
  - v aplikaci je více vláken, každé může mít lokální kopii proměnných (kvůli rychlosti)
  - pokud budou `volatile`, nebudou v lokální kopii
- `repaint()` je pouze požadavek na překreslení
  - nelze zajistit kdy se provede ani zjistit jak dlouho trvá
  - nelze odhadnout, jak dlouhý čas dát do `sleep()`
  - `sleep` je nutný
    - uvolnění procesoru
    - může se provést `repaint()`

Příklad 2

- aktivní vykreslování

```
public void run() {
  running = true;
  while(running) {
    gameUpdate();
    gameRender();
    paintScreen();
    try {
      Thread.sleep(20);
    } catch(InterruptedException ex){}
  }
  System.exit(0);
}
```

```
private void paintScreen() {
  Graphics g;
  try {
    g = this.getGraphics();
    if ((g != null) && (dbImage != null))
      g.drawImage(dbImage, 0, 0, null);
    g.dispose();
    Toolkit.getDefaultToolkit().sync();
  } catch (Exception e)  {
    System.out.
        println("Graphics context error: " + e);
  }
}
```

Příklad 3

- vykreslování je plně pod kontrolou
  => lze měřit, jak dlouho trvá
  => lze nastavit čas do sleep() podle požadovaných FPS

```
public void run() {
  long beforeTime, timeDiff, sleepTime;
  beforeTime = System.currentTimeMillis();
  running = true;
  while(running) {
    gameUpdate();
    gameRender();
    paintScreen();
    timeDiff = System.currentTimeMillis() - beforeTime;
      sleepTime = period - timeDiff;
      if (sleepTime <= 0)
        sleepTime = 5;
      try {
        Thread.sleep(sleepTime);
      } catch(InterruptedException ex){}
      beforeTime = System.currentTimeMillis();
    }
    System.exit(0);
  }
```

Příklad 3

- proměnné period obsahuje požadované FPS v milisekundách
  - př. FPS 100
    1000/100 = 10 ms

- možné problémy
  - nepřesnost časovače
  - různá přesnost na různých systémech
- lépe použít System.nanoTime()
- další možnosti vylepšení
  - počítat nepřesnosti časovače
  - oddělit periodu vykreslování a aktualizace hry

Full-Screen Exclusive Mode

- od JDK 1.4
- přístup přímo do video paměti
  - obchází většinu Swingu a AWT
- třída VolatileImage
  - akcelerované obrázky
  - obvykle není potřeba používat přímo
    - Swing je použije pokud to lze

Full-Screen Exclusive Mode

```
private GraphicsDevice gd;
private Graphics gScr;
private BufferStrategy bufferStrategy;
     :
private void initFullScreen() {
  GraphicsEnvironment ge =
     GraphicsEnvironment.getLocalGraphicsEnvironment();
  gd = ge.getDefaultScreenDevice();
  setUndecorated(true);
  setIgnoreRepaint(true);
  setResizable(false);
  if (!gd.isFullScreenSupported()) {
    System.out.println("Full-screen exclusive mode not supported");
    System.exit(0);
  }
  gd.setFullScreenWindow(this);
  // setDisplayMode(800, 600, 8);
  // setDisplayMode(1280, 1024, 32);
  }
```

Full-Screen Exclusive Mode

- page flipping
  - vykreslovaní do více bufferů
  - nekopírují se jako u double bufferingu
  - pouze se „přehazuje“ ukazatel ve video RAM, co se má zobrazit
- nastavení počtu bufferů

```
try {
  EventQueue.invokeAndWait( new Runnable() {
    public void run()
    { createBufferStrategy(NUM_BUFFERS); }
  });
} catch (Exception e) {
  System.exit(0);
}
try {
  Thread.sleep(500);
}  catch(InterruptedException ex) {}
bufferStrategy = getBufferStrategy();
```

Full-Screen Exclusive Mode

```
private void screenUpdate() {
  try {
    gScr = bufferStrategy.getDrawGraphics();
    gameRender(gScr);
    gScr.dispose();
    if (!bufferStrategy.contentsLost())
      bufferStrategy.show();
    else
      System.out.println("Contents Lost");
  } catch (Exception e)  {
    e.printStackTrace();
    running = false;
  }
}
```

```
private void gameRender(Graphics gScr) {
  gScr.setColor(Color.white);
  gScr.fillRect (0, 0, pWidth, pHeight);
  …
}
```

Full-Screen Exclusive Mode

- ukončení

```
private void restoreScreen() {
  Window w = gd.getFullScreenWindow();
  if (w != null)
    w.dispose();
  gd.setFullScreenWindow(null);
}
```

Další...

- JOGL
  - http://jogamp.org/jogl/
  - používání OpenGL z Javy
- ...

GUI

Systémová integrace pro desktopové aplikace

java.awt.Desktop

- systémová integrace desktopových aplikací
- **`static boolean isDesktopSupported()`**
  - test zda je integrace k dispozici
- **`static Desktop getDesktop()`**
  - vrátí instanci
- **`boolean isSupported(Desktop.Action action)`**
  - test, co je vše je podporováno
  - Desktop.Action
    - enum

Desktop.Action

- APP_ABOUT
- APP_EVENT_FOREGROUND
- APP_EVENT_HIDDEN
- APP_EVENT_REOPENED
- APP_EVENT_SCREEN_SLEEP
- APP_EVENT_SYSTEM_SLEEP
- APP_EVENT_USER_SESSION
- APP_HELP_VIEWER
- APP_MENU_BAR
- APP_OPEN_FILE
- APP_OPEN_URI
- APP_PREFERENCES
- APP_PRINT_FILE
- APP_QUIT_HANDLER
- APP_QUIT_STRATEGY
- APP_REQUEST_FOREGROUND
- APP_SUDDEN_TERMINATION
- BROWSE
- BROWSE_FILE_DIR
- EDIT
- MAIL
- MOVE_TO_TRASH
- OPEN
- PRINT

java.awt.Desktop

- metody odpovídají hodnotám na Desktop.Action

- **`void browse(URI uri)`**
  - otevře uri v defaultním prohlížeči
- **`void edit(File file)`**
  - otevře soubor v defaultním editoru pro daný typ souboru
- **`void mail(URI mailtoURI)`**
  - otevře defaultní mailový klient
- **`void open(File file)`**
  - otevře soubor v defaultní aplikaci pro daný typ
- **`void print(File file)`**
  - tisk souboru
- ...

java.awt.SystemTray

- reprezentuje systémový „tray“
- př.

```
TrayIcon trayIcon = null;
if (SystemTray.isSupported()) {
  SystemTray tray = SystemTray.getSystemTray();
  Image image = ...
  ActionListener listener = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      ...
    }
  };
  PopupMenu popup = new PopupMenu();
  popup.add(...);
  trayIcon = new TrayIcon(image, "Tray Demo", popup);
  trayIcon.addActionListener(listener);
  tray.add(trayIcon);
}
```

java.awt.SystemTray

- pravý stisk tlačítka na ikonu
  - zobrazí menu
- levý stisk
  - generuje action event
- jedna aplikace může přidat libovolné množství ikon
- metody
  - **`static boolean isSupported()`**
  - **`void add(TrayIcon icon)`**
  - **`void remove(TrayIcon icon)`**
    - odebere ikonu z traye
      - při ukončení aplikace jsou ikony odebrány automaticky
  - **`TrayIcon[] getTrayIcons()`**
    - vrací všechny tray ikony aplikace