

Swing

Threads

Overview

- event dispatching and GUI painting
 - **single** thread (*event-dispatching thread*)
 - ensures sequential event processing
 - each event is processed after the previous one is finished
 - events do not interrupt painting
- `SwingUtilities.invokeLater(Runnable doRun)`
- `SwingUtilities.invokeAndWait(Runnable doRun)`
- `SwingUtilities.isEventDispatchingThread()`
 - tests whether the current thread is *event-dispatching thread*
- event processing
 - must finish quickly!
 - for long ones → move it to a special thread

SwingWorker<T, V>

- for lengthy GUI-interaction tasks
- part of JDK since 6
 - for older JDK must be downloaded separately
- abstract class
 - necessary to implement the method
protected abstract T doInBackground()
 - performs the lengthy task
 - the method execute() launches a new thread and runs the doInBackground() method in it

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()` returns a value
 - can be obtained by the method `get()`
 - it blocks until `doInBackground()` terminates
- metoda `done()`
 - called after `doInBackground()` terminates
 - run in the event-dispatching thread (!)

SwingWorker<T, V>

- type parameters
 - T
 - the type of the worker's returning value
 - V
 - the type for intermediate results
 - protected void publish(V... chunks)
 - „**sends**“ data
 - called from doInBackground()
 - protected void process(List<V> chunks)
 - processes the published data
 - intended for overriding
 - run in the event-dispatching thread (!)
- worker's state
 - public SwingWorker.StateValue getState()
 - values PENDING, STARTED, DONE

SwingWorker<T, V>

- current progress
 - int getProgress()
 - void setProgress(int progress)
 - not set automatically
 - has to be called explicitly from doInBackground()
 - but it is not necessary
- addChangeListener(PropertyChangeListener listener)
 - a listener for state and progress changes
- canceling the worker
 - the metoda cancel()
 - doInBackground() must cooperate using the method *isCancel()*;

Swing Timer

- the class `javax.swing.Timer`
 - planning a task for future (repeated) execution
- it is timer for cooperation with GUI
 - intended for tasks that manipulate GUI – there is a special thread that cooperates with the event-dispatching thread
 - "regular" `Timer` should not be used for GUI manipulations
- creation
 - `Timer(int delay, ActionListener listener)`
- **Action listener** – its method is run in the event-dispatching thread (!)
- methods
 - `start()`, `stop()`
 - `setRepeats(boolean b)` – by default `true`

Swing

Own painting

Overview

- redefining the following method of GUI components

```
public void paintComponent(java.awt.Graphics g)
```

- Graphics
 - raphics2D
 - offers methods for painting
 - usually an instance of the child Graphics2D

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

Overview

- can be redefined for any component
 - typically JPanel is used
 - e.g. for games
 - but other component can be used too
 - e.g. buttons
 - JComponent can be extended directly too
- the method **paintComponent()** is called automatically if needed
- explicit repainting request by calling **repaint()**
 - does not call **paintComponent()** directly but
 - puts a repaint request to a queue of events
 - several subsequent requests → single painting

Overview

- `repaint()` exists in several variants
 - without parameters
 - repainting a complete component
 - with parameters
 - repainting a given rectangle only
- note
 - painting is taken (and modified) from AWT
 - in AWT – own painting via the methods `paint()` and `update()`
 - default implementation – `update()` calls `paint()`
 - in Swing – from `paint()`, `paintComponent()` is called
 - plus the methods `paintBorder()` and `paintChildren()`
 - typically no need to override

Swing

Images

Overview

- the core class (from AWT)
`java.awt.Image`
- assumption (from JDK 1.0) – images are loaded over the network
- obtaining an image
 - an applet
 - the method `getImage()`
 - an application
 - `Toolkit.getDefaultToolkit().getImage()`
- drawing
 - `g.drawImage()` `// Graphics g;`
- supports GIF, PNG, JPG

Example

```
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);
    }
}
```

- an issue
 - getImage() does not load the image, just allocates memory
 - the image is loaded in drawImage() during drawing

Drawing

- Graphics.drawImage(Image img, int x, int y, ImageObserver observer)
 - ImageObserver
 - monitors loading the image
 - periodically calls imageUpdate()
 - by default it calls repaint()
 - JApplet and JFrame implements ImageObserver
- MediaTracker class
 - “pre-loading” images

```
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");  
    }  
}
```

Imagelcon

- merge of Image and ImageTracker
`im = new Imagelcon( getDocumentBase()+"ball.gif").getImage();`
- can be used for any image
 - not only icons (small images)
- typical usage in applications
`im = new Imagelcon( getClass().getResource("ball.gif") ).getImage();`

Java 2D API

- added in latter versions
- extension of graphic operations
- the core class

`java.awt.Graphics2D`

- extends `java.awt.Graphics`
- the method `paintComponent()` still has “only” the type `Graphics`
 - => must be explicitly casted
 - can be done in fact always
- in active painting (will be later)
 - the return value of `getGraphics()` can be also cast to `Graphics2D`
- offers more operations than `Graphics`
- easier to use

BufferedImage

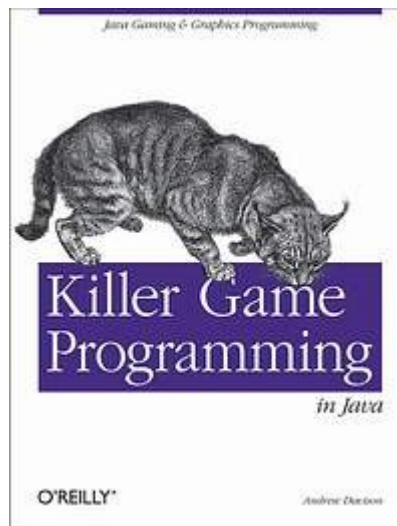
- extends Image
 - the package `java.awt.image`
- easy access to data of images
- automated conversion to *managed image*, which allow for usage of HW acceleration
- loading via `javax.imageio.ImageIO.read()`
 - should be faster than `ImageIcon`
- operations with `BufferedImage`
 - classes implementing `java.awt.image.BufferedImageOp`
 - transformations
 - `AffineTransformOp`, `ColorConvertOp`,...

Swing

Drawing in games

Overview

- examples taken from the book
A. Dawison: ***Killer Game Programming in Java***
 - the book can be downloaded at
<http://fivedots.coe.psu.ac.th/~ad/jg/>
 - not a final version of the book
 - also there are some additional chapters
 - the book exists in Czech also
 - Programování dokonalých her v Javě



Example 1

1/2

```
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();
        }
    }
}
```

Example 1

2/2

```
...
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)
        ...
}
...
}
```

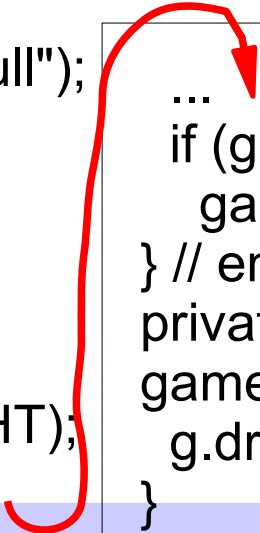
Example 1

Rendering

- usage of „double buffering“
 - drawing to an off-screen buffer
 - copying the buffer to the screen

```
private Graphics dbg;  
private Image dblImage = null;  
:
```

```
private void gameRender() {  
    if (dblImage == null){  
        dblImage = createImage(PWIDTH, PHEIGHT);  
        if (dblImage == null) {  
            System.out.println("dblImage is null");  
            return;  
        } else  
            dbg = dblImage.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);  
}
```

- copying the buffer in paintComponent()

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


Example 1

Input

- adding reactions to user input

```
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()); }  
    });  
}
```

Example 1

Input

```
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) {
        ...
    }
}
```

- the variables `running` and `gameOver` must be `volatile`
 - there are several threads – each of them can have a local copy of the variables (because of performance)
 - if they are `volatile`, they cannot be in a local copy
- `repaint()` only request for repainting
 - no guarantee when executed; its execution time cannot be obtained
 - amount of time for `sleep()` cannot be estimated
 - `sleep` is necessary
 - releasing CPU
 - `repaint()` can be executed

Example 2

- active rendering

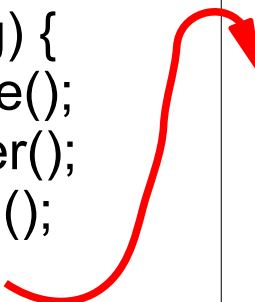
```
public void run() {
    running = true;
    while(running) {
        gameUpdate();
        gameRender();
        paintScreen();
        try {
            Thread.sleep(20);
        } catch (InterruptedException ex){}
    }
    System.exit(0);
}
```

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

Example 3

- painting fully controlled
=> can be measured
=> time for sleep() can be set based on requested 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);  
}
```



Example 3

- the period variable contains requested FPS in milliseconds
 - example FPS 100
 $1000/100 = 10 \text{ ms}$
- possible problems
 - imprecise timer
 - different precision on different platforms
- better to use `System.nanoTime()`
- further possibilities for enhancements
 - counting imprecision of the timer
 - separation of rendering period and game state update period

Full-Screen Exclusive Mode

- since JDK 1.4
- direct access to video RAM
 - bypasses most of Swing and AWT
- the class `VolatileImage`
 - accelerated images
 - no need to use directly
 - Swing decides when possible

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
 - drawing to several buffers
 - no copying
 - only switching of video RAM pointer
- setting a number of buffers

```
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

- end

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

Others...

- JOGL
 - <http://jogamp.org/jogl/>
 - usage of OpenGL
- ...

GUI

System integration for desktop applications

java.awt.Desktop

- system integration for desktop applications
- **static boolean isDesktopSupported()**
 - whether the desktop integration is supported
- **static Desktop getDesktop()**
 - returns an instance of the desktop
- **boolean isSupported(Desktop.Action action)**
 - what is supported
 - Desktop.Action
 - enum

Desktop.Actions

- 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

- methods corresponds with values in Desktop.Action
-
- **void browse (URI uri)**
 - opens an uri in the default browser
- **void edit (File file)**
 - opens the file in the default editor for the given file type
- **void mail (URI mailtoURI)**
 - opens the default mail client
- **void open (File file)**
 - opens the file in the default program for the given file type
- **void print (File file)**
 - prints file
- ...

java.awt.SystemTray

- represents the system “tray”
- example

```
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

- right-click on the icon
 - shows menu
- left-click
 - generates the action event
- a single application can add any number of icons
- methods
 - **static boolean isSupported()**
 - **void add(TrayIcon icon)**
 - **void remove(TrayIcon icon)**
 - removes the icon from the tray
 - when the application terminates the icons are moved automatically
 - **TrayIcon[] getTrayIcons()**
 - returns all tray icons of the application



Slides version AJ06.en.2020.01

This slides are licensed under a [Creative Commons Attribution-NonCommercial 4.0 International License](https://creativecommons.org/licenses/by-nc/4.0/).