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
- addPropertyChangeListener(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

JTree

Overview

- javax.swing.JTree
- displaying hierarchical data
- JTree does not hold data directly
  - only displays data
  - data are hold by a *model (model-view concept)*
- in general
  - all more complex components have a model
    - JTree, JTable, JList, JButton, ...
  - the model determines how the data are stored and retrieved
  - a single component can have multiple models
    - e.g. JList
      - ListModel – holds a content of the list
      - ListSelectionModel – manages current selection

JTree: static content

```
DefaultMutableTreeNode top =
        new DefaultMutableTreeNode("Root");
createNodes(top);
tree = new JTree(top);
...
private void createNodes(DefaultMutableTreeNode top) {
  DefaultMutableTreeNode node = null;
  DefaultMutableTreeNode leaf = null;

  node = new DefaultMutableTreeNode("Node1");
  top.add(node);

  leaf = new DefaultMutableTreeNode("Leaf1");
  node.add(leaf);
  leaf = new DefaultMutableTreeNode("Leaf2");
  node.add(leaf);

  node = new DefaultMutableTreeNode("Node2");
  top.add(node);
```

JTree: dynamic changes

```
rootNode = new DefaultMutableTreeNode("Root Node");
treeModel = new DefaultTreeModel(rootNode);
treeModel.addTreeModelListener(new MyTreeModelListener());
tree = new JTree(treeModel);
tree.setEditable(true);
tree.getSelectionModel().setSelectionMode
        (TreeSelectionModel.SINGLE_TREE_SELECTION);
...
class MyTreeModelListener implements TreeModelListener {
  public void treeNodesChanged(TreeModelEvent e) {
  }
  public void treeNodesInserted(TreeModelEvent e) {
  }
  public void treeNodesRemoved(TreeModelEvent e) {
  }
  public void treeStructureChanged(TreeModelEvent e) {
  }
}
```

JTree: dynamic changes

```
public DefaultMutableTreeNode addObject(DefaultMutableTreeNode
                parent, Object child, boolean shouldBeVisible) {

  DefaultMutableTreeNode childNode =
           new DefaultMutableTreeNode(child);
  ...
  treeModel.insertNodeInto(childNode, parent,
                           parent.getChildCount());

  if (shouldBeVisible) {
    tree.scrollPathToVisible(new TreePath(childNode.getPath()));
  }
  return childNode;
}
```

JTree: own model

- *model-view*
  - Model
    - describes data (e.g. DefaultTreeModel)
  - View
    - defines how to display data (JTree)
- default model – DefaultTreeModel
- if not suitable → own model
  - e.g., by default, nodes in the tree are DefaultMutableTreeNode and implements the TreeNode interface
    - own model can have nodes of a completely different type
- the model must implement TreeModel interface

TreeModel

```
void addTreeModelListener(TreeModelListener l);

Object getChild(Object parent, int index);

int getChildCount(Object parent);

int getIndexOfChild(Object parent, Object child);

Object getRoot();

boolean isLeaf(Object node);

void removeTreeModelListener(TreeModelListener l);

void valueForPathChanged(TreePath path, Object;
  newValue);
```

Icons in JTree

- TreeCellRenderer
  - interface
- setCellRenderer(TreeCellRenderer r)
  - method of JTree

```
class MyRenderer extends DefaultTreeCellRenderer {
  public Component
  getTreeCellRendererComponent(JTree
     tree,Object value,boolean sel,boolean expanded,
     boolean leaf,int row,boolean hasFocus) {

    super.getTreeCellRendererComponent(tree, value,
        sel, expanded, leaf, row, hasFocus);
    if (....) {
      setIcon(someIcon);
      setToolTipText("....");
    } else {.....}
    return this;
```

Icons in JTree

```
ImageIcon leafIcon = createImageIcon("..");

if (leafIcon != null) {
    DefaultTreeCellRenderer renderer =
 new DefaultTreeCellRenderer();

    renderer.setLeafIcon(leafIcon);
    tree.setCellRenderer(renderer);
}
```

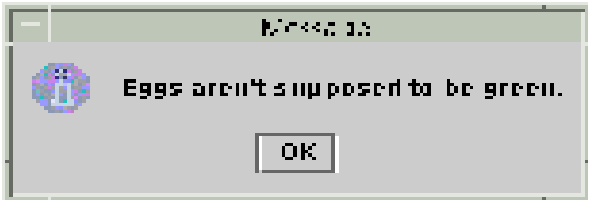
Swing

Dialogs

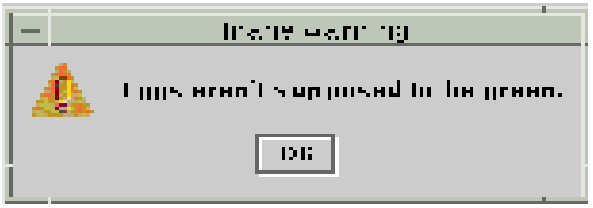
Overview

- *JDialog*
- *a dialog* = a window similar to the *frame*
- *dialogs* depend on a *frame*
- a dialog is modal
  - if it is displayed, input to other windows of an application is blocked
  - non-modal dialogs can be created also
- managing the dialog – almost the same as for frame
- JOptionPane
  - a component simplifying creation of standard dialogs
  - predefined dialogs

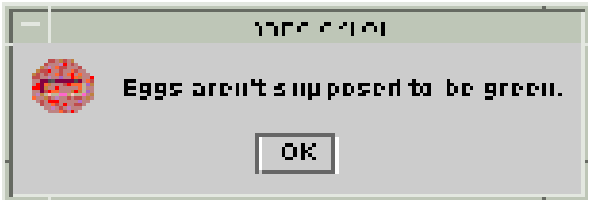
JOptionPane



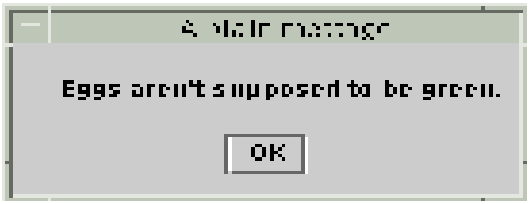

```
//default title and icon
JOptionPane.showMessageDialog(frame,
    "Eggs aren't supposed to be green.");
```



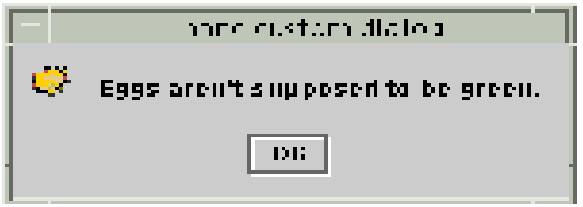

```
//custom title, warning icon
JOptionPane.showMessageDialog(frame,
    "Eggs aren't supposed to be green.",
    "Inane warning",
    JOptionPane.WARNING_MESSAGE);
```




```
//custom title, error icon
JOptionPane.showMessageDialog(frame,
    "Eggs aren't supposed to be green.",
    "Inane error",
    JOptionPane.ERROR_MESSAGE);
```




```
//custom title, no icon
JOptionPane.showMessageDialog(frame,
    "Eggs aren't supposed to be green.",
    "A plain message",
    JOptionPane.PLAIN_MESSAGE);
```




```
//custom title, custom icon
JOptionPane.showMessageDialog(frame,
    "Eggs aren't supposed to be green.",
    "Inane custom dialog",
    JOptionPane.INFORMATION_MESSAGE,
    icon);
```

JOptionPane

- predefined dialogs
  - but can be configured
- a set of static methods creating dialogs (always several variants of the single method)
  - showMessageDialog()
    - a dialog with message
  - showInputDialog()
    - a dialog with an input line
    - returns String
  - showConfirmDialog()
    - a dialog with a question (Yes/No/Cancel)
    - returns int
  - showOptionDialog()
    - selection of several choices (Yes-No-Maybe-Cancel)

JOptionPane

- can be also used directly
  - by creating an instance of JOptionPane
    - several constructors
  - the created object can inserted to a dialog

JFileChooser

- a standard dialog for file selection

```
JFileChooser chooser = new JFileChooser();
chooser.setDialogType(JFileChooser.OPEN_DIALOG)
FileNameExtensionFilter filter =
                    new FileNameExtensionFilter(
                                "Images", "jpg", "gif");
chooser.setFileFilter(filter);
int returnVal = chooser.showOpenDialog(parent);
if (returnVal == JFileChooser.APPROVE_OPTION) {
  System.out.println("Selected file: " +
  chooser.getSelectedFile().getName());
}
```

JAVA

Applet

Overview

- applet = a "small" program running in a browser
- security limitations
  - in fact no access to the computer
    - limited access to a disk
    - limited access to a net
    - ...
- "signing" applets
- "slower" launching
  - the code is downloaded over the network
  - the browser can have the code in its cache
    - but not always

Implementation

- extending the class **javax.swing.JApplet**
  - applets using the Swing components
  - extending **java.applet.Applet**
    - AWT components
- methods
  - init()
    - called during initialization
    - contains e.g. GUI creation
    - WARNING – it is necessary to use *SwingUtilities.invokeAndWait(Runnable r)*
  - start()
    - called always when the applet is shown
  - stop()
    - called always when the applet is hidden
  - destroy()
    - called during termination

Implementation

- the applet is a container
- class hierarchy
  - java.lang.Object
    java.awt.Component
    java.awt.Conteiner
    java.awt.Panel
    java.awt.Applet
    javax.swing.JApplet
- GUI is created the same way as in “regular” application

Deploying

- a browser runs the applet
  - below old and not recommended approach

```
<applet code="Example1.class"
        codebase="example"
        archive="Example.jar, Utilities.jar"
        width="100" height="80">
    <param name="maxwidth" value="120">
    <param name="nimgs" value="17">
    <param name="offset" value="-57">
    <param name="img" value="images/tumble">
Your browser is completely ignoring the &lt;APPLET&gt;
  tag!
</applet>
```

- appletviewer
  - a part of JDK
  - shows only the applet
    - without the html page around

Example

```
public class Applet1 extends JApplet {
  private void createGUI() {
    getContentPane().add(new JLabel("Applet!"));
  }

  public void init() {
    try {
      SwingUtilities.invokeAndWait(new Runnable() {
        public void run() {
          createGUI();
        }
      });
    } catch (Exception e) {
      System.exit(0);
    }
  }
}
```

JNLP
Java Web Start

Overview

- Java Network Launch Protocol (JNLP)
- advantages of both regular applications and applets
- an application is downloaded over the network
  - a regular application
- it is saved to the disk and launched in a *sandbox*
- in later launches, first a version is checked
  - if it is the same, the application is run from the disk
  - if it changed, the application is downloaded
- similar security limitations like for applets
  - applications can be signed
  - even non-signed application can have access to the computer
    - via JNLP API
- **JNLP** is only a specification
  - implementation – Java Web Start

Creating a JNLP application

- a regular application packed in a JAR file
- plus a descriptor file
  - xml, the jnlp filename extension
- Web server – mime.conf
  - *application/x-java-jnlp-file JNLP*

```
<?xml version="1.0" encoding="UTF-8"?>
   <jnlp codebase="http://somewhere/" href="hello.jnlp">
     <information>
         <title>Example</title>
         <vendor>Vendor</vendor>
         <description>Description</description>
         <icon href="icon.png"/>
         <offline-allowed/>
     </information>
     <resources>
       <j2se version="1.4+"/>
       <jar href="hello.jar"/>
     </resources>
     <application-desc main-class="MainClass" />
</jnlp>
```

JNLP API

- for JDK <= 1.4
  - must be downloaded
- API defines services for accessing resources of a computer
  - files, network, clipboard,...
- a dialog is shown and a user has to allow access
- direct access does not work
  - an exception is thrown

- services are loaded via javax.jnlp.ServiceManager.lookup(String s)
- access to all resources can be allowed for signed JARs using the JNLP file:

```
<security>
  <all-permissions/>
</security>
```

FileOpenService

```
FileOpenService fs = (FileOpenService)
     ServiceManager.lookup("javax.jnlp.FileOpenService");

FileContents fileContents = fs.openFileDialog(".",
            new String[]{"txt", "*"});

if (fileContents != null) {

  String fname = fileContents.getName();
  InputStream is = fileContents.getInputStream();

  ...
}
```

FileSaveService

```
FileSaveService fs = (FileSaveService)
     ServiceManager.lookup("javax.jnlp.FileSaveService");

FileContents fileContents = fs.saveFileDialog(".",
     new String[]{"txt"},
     new ByteArrayInputStream("data".getBytes()),
     "soubor.txt");

if (fileContents != null) {

  fileContents.getName();
  ...
}
```

FileContents

- methods
  - boolean canRead()
  - boolean canWrite()
  - long getLength()
  - String getName()
  - InputStream getInputStream()
  - OutputStream getOutputStream()
  - JNLPRandomAccessFile getRandomAccessFile(String mode)

JNLP & Applets

- applets can be also deployed using JNLP
  - currently recommended approach
  - necessary to have JDK 6 and higher

```
<?xml version="1.0" encoding="UTF-8"?>
<jnlp spec="1.0+" codebase="" href="">
    <information>
        <title>Dynamic Tree Demo</title>
        <vendor>Dynamic Team</vendor>
    </information>
    <resources>
        <j2se version="1.6+" href="http://java.sun.com/products/autodl/j2se" />
        <jar href="DynamicTreeDemo.jar" main="true" />
    </resources>
    <applet-desc  name="Dynamic Tree Demo Applet"
         main-class="components.DynamicTreeApplet"
         width="300"  height="300">
     </applet-desc>
     <update check="background"/>
</jnlp>
```

JNLP file

```
<script src="http://www.java.com/js/deployJava.js"></script>
<script>
    var attributes = { code:'components.DynamicTreeApplet',  width:300, height:300} ;
    var parameters = {jnlp_href: 'dynamictree-applet.jnlp'} ;
    deployJava.runApplet(attributes, parameters, '1.6');
</script>
```

HTML page