ISCTE IUL
Instituto Universitário de Lisboa

Concurrency and Swing

Programação Concorrente e Distribuída

Parallel and Distributed Programming

2012-2013

After this class you will be able to…

- Understand how threads can interact with Swing
- Will have gotten a complete overview of Swing architecture
- Know what are:
  - Initial Thread
  - Event Dispatcher Thread
  - Worker Thread
- Understand what are Java Applets
  - Life cycle
  - Advantages
  - Restrictions

Threads in Swing

- Careful use of concurrency is particularly important to the Swing programmer.

- In the Swing part of a program we should **only** have the following threads:
  - ***Initial threads (the thread that realizes the GUI);***
  - **The *event dispatch thread* (EDT);**
  - ***Worker threads*;**

- The programmer does not need to provide code that explicitly creates these threads.
- The programmer's job is to utilize these threads to create a responsive, maintainable Swing program.

Initial Thread

- Initial Thread should create a Runnable object that initializes the GUI and schedule that object for execution on the event dispatch thread;
- Once the GUI is created, the program is primarily driven by GUI events;
- An initial thread schedules the GUI creation task by invoking javax.swing.SwingUtilities.invokeLater or javax.swing.SwingUtilities.invokeAndWait .

```
SwingUtilities.invokeLater(new Runnable() {
    public void run() {
        createAndShowGUI();
    }
});
```

Event Dispatch Thread

- Thread that is responsible to handle the Swing events;
- Most code that invokes Swing methods also runs on this thread.
- This is necessary because most Swing object methods are not "thread safe";
- Some Swing component methods are labelled "thread safe" in the API specification; these can be safely invoked from any thread.
- All other Swing component methods must be invoked from the event dispatch thread.
- We can see the code running on the event dispatch thread as a series of short tasks.
- Tasks on the event dispatch thread must finish quickly.

Worker Thread (from v1.6)

- If a Swing component needs to execute a long-running task, it should uses a worker threads (background thread);
- The tasks that will run on a worker thread have to be instances of javax.swing.SwingWorker;
- SwingWorker provides a number of communication and control features:
    - done(), invoked on the event dispatch thread when the background task is finished;
    - publish(), causes SwingWorker.process to be invoked from the event dispatch thread;
    - the background task can define bound properties;
    - SwingWorker implements java.util.concurrent.Future.

Worker Thread Workflow

Three threads are involved in the life cycle of a SwingWorker :

- **Current thread**: execute() schedules the SwingWorker for the execution on a worker thread and returns immediately. The get method only returns when the SwingWorker is complete.
- **Worker thread**: runs the doInBackground() where all background activities should happen.
- **Event Dispatch Thread**: invokes the process() and done() methods and notifies any PropertyChangeListeners on this thread.

javax.swing.SwingWorker

Class SwingWorker<T,V>

Type Parameters:

`T` - the result type returned by this `SwingWorker's` `doInBackground` and `get` methods

`V` - the type used for carrying out intermediate results by this `SwingWorker's` `publish` and `process` methods

- doInBackground() - this method is executed in a background thread.
- process() - receives data chunks from the publish method asynchronously on the *Event Dispatch Thread*.
- done() - executed on the *Event Dispatch Thread* after the doInBackground method is finished.

Cancel a Worker Thread

- To cancel a running background task, invoke SwingWorker.cancel
- The task must cooperate with its own cancellation. There are two ways it can do this:
  - By terminating when it receives an **interrupt**.
  - By invoking SwingWorker.isCanceled at short intervals. This method returns true if cancel has been invoked for this SwingWorker.

Bound Properties of the SwingWorker

- SwingWorker supports bound properties, which are useful for communicating with other threads.
- Two bound properties are predefined: progress and state.
- As with all bound properties, progress and state can be used to trigger event-handling tasks on the event dispatch thread.

Bound Properties of the SwingWorker

- The **progress** bound variable is an int value that can range from 0 to 100. (SwingWorker.setProgress and SwingWorker.getProgress).
- The **state** bound variable indicates where the SwingWorker object is in its lifecycle. (SwingWorker.getState). Possible values are:
  - PENDING - from the construction until just before doInBackground is invoked.
  - STARTED - from shortly before doInBackground until shortly before done is invoked.
  - DONE - remainder of the existence of the object.

Bound Properties of the SwingWorker

```
task = new SwingWorker<List<Integer>, Integer>() {
   @Override
   public List<Integer> doInBackground() {
     while (! enough && ! isCancelled()) {
       number = nextPrimeNumber();
       publish(number);
       setProgress(100 * numbers.size() / numbersToFind);
     }
     return numbers;
   }

   @Override
   protected void process(List<Integer> chunks) {
         for (int number : chunks) {
             textArea.append(number + "\n");
         }
   }
}
```

Bound Properties of the SwingWorker

```
task.addPropertyChangeListener(new
                PropertyChangeListener() {
   public  void propertyChange(PropertyChangeEvent evt) {
       if ("progress".equals(evt.getPropertyName())) {
            label.setValue((Integer)evt.getNewValue());
       }
   }
});

task.execute();
System.out.println(task.get()); //waits until task is done
                               //and prints all prime
                               //numbers we have got
```

Applets

Applets

- An Applet is a special Java application that runs in a browser enabled with Java technology.
- Applets can be downloaded from the internet and run in a browser.
- Applets are inserted into web pages as html tags.
- Swing provides a special subclass of the Applet class called JApplet.

Webpages and HTML

- Webpages use a language called *Hypertext Markup Language* (HTML)
  - HTML defines a set of *tags.* The tags specify the rules for the layout and (limited) interaction of on a webpage.
- Webpages are usually divided into the follow sections:

```
<HTML>
<HEAD>
    ....
</HEAD>
<BODY>
    ....
</BODY>
</HTML>
```

- Several technologies exist for making webpages more sophisticated: JavaScript and AJAX, HTML 5, Flash, applets and so on...

Running Applets

- A Java Applet can be embed on a wepage using the <applet ...> tag. For instance:

```
<applet code=AppletWorld.class width="200" height="200">
</applet>
```

- In Eclipse, applets can be tested and run in an *Applet Viewer.* Right-click on the applet Java file, select “Run as...” and select “Java Applet”

HelloWorld

HelloWorld.java

```
public class HelloWorld extends JApplet {
    public void paint(Graphics g) {
    g.drawRect(0, 0,
          getSize().width - 1,
          getSize().height - 1);
       g.drawString("Hello world!", 5, 15);
    }
}
```

HelloWorld.htm

```
<html>
    <head>
        <title> HelloWorld Applet</title>
    </head>
    <body>
        <applet code=HelloWorld.class width="200" height="200">
        </applet>
    </body>
</html>
```

The Life-Cycle of Applets

There exist four basic method in the applet class:

- **init:** Used for any type of initialization. The init method is called after the definition of the param attributes in the applet tag. Called exactly once.
- **start:** Called at least once when the applet is started or restarted.
- **stop:** Called at least once in an applet's life, when the browser leaves the page in which the applet is embedded.
- **destroy:** Called exactly once in an applet's life, just before the browser unloads the applet.

Life Cycle Example 1/2

```
public class Simple extends JApplet {
    StringBuffer buffer;
    public void init() {
        buffer = new StringBuffer();
        addItem("initializing... ");
    }

    public void start() {
        addItem("starting... ");
    }

    public void stop() {
        addItem("stopping... ");
    }
    ….
```

Life Cycle Example 2/2

```
   public void destroy() {
          addItem("preparing for unloading...");
   }

   private void addItem(String newWord) {
          System.out.println(newWord);
          buffer.append(newWord);
          repaint();
   }

       public void paint(Graphics g) {
      //Draw a Rectangle around the applet's display area.
          g.drawRect(0, 0, getWidth() - 1, getHeight() - 1);
      //Draw the current string inside the rectangle.
          g.drawString(buffer.toString(), 5, 15);
      }
}
```

Threads in applets

- Any applet can create and start one or more threads. The applet's GUI is drawn by the *event dispatching thread (EDT)*.
- The thread(s) calling the four methods — *init, start, stop,* and *destroy* — depend on the application in which the applet is running.

Drawing and event-handling example

```
public class SimpleClick extends JApplet{
    StringBuffer buffer;

       public void init() {
        addMouseListener(new MouseListener(){
            public void mouseEntered(MouseEvent event) {    }
            public void mouseExited(MouseEvent event) {    }
            public void mousePressed(MouseEvent event) {    }
            public void mouseReleased(MouseEvent event) {    }
            public void mouseClicked(MouseEvent event) {
                         addItem("click!... ");
            }
        });
        buffer = new StringBuffer();
            addItem("initializing... ");
    }

    public void start() {   addItem("starting... ");    }
    public void stop() {    addItem("stopping... ");    }
    public void destroy() { addItem("preparing for unloading...");  }

    void addItem(String newWord) {
            buffer.append(newWord);
            repaint();
    }

    public void paint(Graphics g) {
            g.drawRect(0, 0, getWidth() - 1, getHeight() - 1);
            g.drawString(buffer.toString(), 5, 15);
        }
}
```

Specializing JApplet

- An applet has to extend from JApplet.
- In this way, the specialization inherits various functionality such as communication with the browser and the ability to display a GUI to the user.
- The class JApplet offers high-level components, such as the classes JFrame and JDialog in Swing.

The key methods of the classes JApplet

- Methods for the different phases of an applet's life `(init, start, stop, destroy)`
- Methods for adding graphical components to the applet's GUI (identical to the usual GUI methods)
- Methods for registering event listeners (identical to the usual GUI methods)

Applets *vs*. Java applications

- The fundamental difference between applets and Java applications is that applets are meant to be executed in the context of a webpage in a browser.
- A Java application, on the other hand, is executed independently outside of a browser.

Applets *vs*. Java applications

When comparing an applet and a Java application, we see that:

- An applet is declared public so that it can be accessed by an *appletviewer*
- Applets inheret from JApplet while an application uses the `JFrame` to create a GUI.
- An applet does not have a *main* method.
- The constructor of an applet is replaced by the methods *init* and *start*.
- GUI components are added directly to an applet while in an application they are added to, for instance, the *content pane* of an object of the class `JFrame`.

Convert Java Applications into Applets

Key changes necessary to turn a Java application into an applet:

- It is necessary to extend from JApplet in which the *init* method is used to initialize the applet resources in the same way the main method initializes the application resources.
- The top element Panel has to be added to the apple in the *init* method; usually it would be added to a JFrame in the main method.

Advantages and restrictions

- Applets can easily be included in a webpage and thereby be distributed to end-users. Applications, on the other hand, need to be downloaded and usually installed.
- Applets are executed in a restricted environment (“sandbox”). An applet is therefore safe to run from a user's standpoint since it cannot perform destructive operations or compromise the user's privacy (such as read or write files).

Restrictions

Browsers impose the following restrictions on applets that have been downloaded over the net (i.e. those on normal webpages) :

- An applet cannot call libraries that contain native (non-Java) code.
- An applet cannot read or write files on the computer on which they are executed.
- An applet cannot connect to other hosts than the one from which the applet was loaded.
- An applet cannot start programs on the computer on which it is running.

Restrictions

- An applet can execute public methods on other applets on the same page.

- Applets that are that are loaded from the local machine do not necessarily have the same restrictions as applets downloaded as part of a page on the internet. In Eclipse's *appletviewer,* it is possible to set the class access to either *restricted* or *unrestricted*.

To explore

Taking Advantage of the Applet API

- http://download.oracle.com/javase/tutorial/deployment/applet/
- http://download.oracle.com/javase/tutorial/uiswing/concurrency/dispatch.html

Base bibliography

Concurrency in Swing

http://docs.oracle.com/javase/tutorial/uiswing/concurrency/index.html

JAVA Applet Tutorial

http://docs.oracle.com/javase/tutorial/deployment/applet/index.html

Summary

- Threads and Swing
- Swing architecture
- Worker Threads and Swing Workers
- Web and Applets
  - The life cycle of Applets
  - Threads in applets and the EDT
  - The `Japplet` class
  - Applets vs Java applications
  - Advantages and restrictions