

Threads and Timers

Chapter 13 - Lecture Slides

Threads

- Multi-processing:
- When we run a program, a single thread automatically starts to run the program.
- To run two or more things simultaneously, we need to create a second (or more) thread(s).
- Ideal for
 - For example, if we want to perform a slideshow **while** the user is typing inside a, we need to use a separate thread for the slideshow. `JTextArea`
 - If we didn't use a second thread, the applet would hang while we ran through the code for the slideshow.

Threads – steps to make it work

- There are four steps to get Threads to work:

1. The class needs to implement the `Runnable` interface.

```
implements Runnable
```

2. Create a Thread

```
Thread runner;  
runner = new Thread( this );
```

3. Start the Thread

```
runner.start( );
```

4. Create a method named `run`

```
public void run( )  
{  
}
```

`run()` method

- Code goes inside the `run` method.
Usually we want the code to loop. We can set up the loop as an infinite loop.

```
public void run( )  
{  
    while( true )  
    {  
        // code  
    }  
}
```

- Use the method `sleep` in the `Thread` class to simulate a delay:
(This is the proper way to delay, unlike the for loop we used earlier in the book)

```
Thread.sleep( delayInMilliseconds );  
OR  
thread.sleep( delayInMilliseconds );    // thread variable
```

- To use the `sleep` method, we need to put it within a `try ... catch` block to catch any exceptions (errors) that may occur
(usually put the `try...catch` around the `while(true)` loop):

```
try {  
    Thread.sleep( delayInMilliseconds );  
} catch( Exception exc ) { }
```

Applet methods

Method Header	Description
---------------	-------------

```
public void init( )
```

```
public void start( )
```

```
public void stop( )
```

Slideshow Example

- Slideshow inside main applet class
 - see Slideshow.java
- Slideshow in a separate class
 - see Slides.java and SlidesApplet.java

Animation Example

- Animation is done by redrawing
 - see Wink.java and Smile.java
 - See AnimateThreads.java

Timers

Timers

- Timers are useful for repeating steps at particular intervals.
 - Examples:
- Timers are based on
 - Important that the code to be run from a `Timer` event can be executed **quickly** to enable the system to handle the next event. (*Threads do not have this limitation and therefore are ideal for more time-intensive code processing*).
- Create a `Timer` object by specifying the delay count in milliseconds and the listener for the :

```
Timer timer;  
timer = new Timer( delay, this );
```

Timers

- Similar to threads, we then need to start the timer:
- After the delay in milliseconds, the `actionPerformed` method is called. Therefore, just like when we listen for events on buttons, we need to implement the `actionPerformed` method.

```
public void actionPerformed( ActionEvent ae )
{
    Object src = ae.getSource( );
    if ( src instanceof Timer )
        // do something since Timer expired
    else if ( src instanceof JButton )
        // do something based on a button click
}
```

Timers

- There are five steps to get Timers to work:
 1. Import the package to handle ActionEvent events:

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

2. Specify that we're listening for events:

```
public class xyz extends JApplet implements ActionListener
```

3. Create a Timer object

```
Timer timer;
timer = new Timer( delay, this );
```

4. Start the timer:

```
timer.start( );
```

5. Add an actionPerformed method:

```
public void actionPerformed( ActionEvent ae )
{
    Object src = ae.getSource( );
    if ( src instanceof Timer )
        ...
    else if ( src instanceof JButton )
        ...
}
```

6. (Optional) Stop the Timer

```
timer.stop( );
```

Timers - Example

- TimerEx.java
- ClockTimer.java



Summary

- Threads
- Timers