3. Introduction to threads

In this chapter

- What is multithreaded programming?
- Constructing concurrently running Threads

Modern computers have the ability to perform multiple tasks seemingly, i.e. running multiple programs or processes, simultaneously. Strictly speaking, a single CPU can perform a single computation at a time. But the high speeds of the processors and the ability of modern operating systems to schedule which computations are going to be performed at each moment make the *multitaksing* possible.

Multithreading, on the other hand, extends the idea of multitasking and allows a single program to perform multiple taks simultaneously (or strictly speaking, concurrently on a single CPU). Each of these tasks is called a *thread*. The programs which can run more than one task concurrently are called *multithreaded*. This technique of programming is often named as *parallel programming*.

Multithreading is extremely useful for a psychophysics experiment. Suppose that you want to present your observers an animation and ask them to adjust the luminance of a test patch in the scene. In order to allow a smooth flow of the animation, you can construct three threads which perform the following three tasks: One for displaying the frames of animation, a second one for getting the observer response, and the third one to re-render the stimulus and change the luminance of the test patch depending on the observer's response.

3.1. Constructing concurrently running threads

Although extremely useful, multithreaded programming can get very complex. In this chapter I only introduce how to construct a new thread and run an *experiment* in that new thread. Later in Chapter XX, we will work on a more complex example of multithreaded programming.

The creation of a thread is actually quite straightforward. You place the code that performs the task in the run() method of a class which *implements* the Runnable interface (see the references listed in Chapter XXX for more on object oriented programming and interfaces)

```
class RunnableTest implements Runnable {

  public void run(){

    // code to perform your task
  }
}
```

next you construct an object of that class, then construct a Thread with that object and finally start the Thread. Here are those three steps

```
Runnable test = new RunnableTest();
Thread experiment = new Thread(test);
experiment.start();
```

I will use the same example as in previous chapter to emphasize how to construct a new thread. Here is the multithreaded version of HelloPsychophysicist example from Chapter 2.

```
/*
 * chapter 3: HPThreaded.java
 *
 * Multithreaded version of HelloPsychophysicist of Chapter 2
 *
 * displays the text "Hello Psychophysicist (Threaded)"
 * and two images on an otherwise entirely blank screen
 *
 */
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
public class HPThreaded extends FullScreen1 implements Runnable {

  public static void main(String[] args) {

    HPThreaded fs = new HPThreaded();
    fs.setNBuffers(2);
    Thread experiment = new Thread(fs);
    experiment.start();
  }

  public void run() {

    try {

      displayText("Hello Psychophysicist (Threaded)");
      updateScreen();
      Thread.sleep(2000);
      blankScreen();
      hideCursor();
      BufferedImage bi1 = ImageIO.read(new File("psychophysik.png"));
      displayImage(bi1);
      updateScreen();
      Thread.sleep(2000);
      blankScreen();
      BufferedImage bi2 = ImageIO.read(new File("fechner.png"));
      displayImage(0,0,bi2);
      updateScreen();
      Thread.sleep(2000);
```

```
      } catch (IOException e) {
        System.err.println("File not found");
        e.printStackTrace();
      } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
      }
      finally {
        closeScreen();
      }
    }
  }
```

The first difference from the previous version is that HPThreaded inherits from the FullScreen1 class

```
public class HPThreaded extends FullScreen1
```

This means that HPThreaded itself is-a FullScreen1. We can construct an object of HPThreaded as we constructed an object of FullScreen1 in the previous chapter. All the methods of the FullScreen1 class will be available for the object of the class HPThreaded as well. This approach is going to save us quite a bit of bookkeeping and I will use it throughout this guide.

The other difference is that HPThreaded implements the Runnable interface

```
public class HPThreaded extends FullScreen1 implements Runnable
```

This way we can create a HPThreaded object with the convinience of having all the methods of FullScreen1 class accessible, moreover we can also create a new Thread using that object and put the experimental code inside its own run() method (*multiple inheritance*). This results in a clearer and a simpler code.

As explained above, we first create an object of a class which implements the Runnable interface

```
HPThreaded fs = new HPThreaded();
```

Note that the HPThreaded class must implement the run() method (see below). Because HPTHreaded is a Runnable, we can create a Thread object using a HPTHreaded object

```
Thread experiment = new Thread(fs);
```

At last, to initiate the execution of the run() method, we invoke the start() method of the Thread class

```
experiment.start();
```

Note that we don't directly invoke the run() method of the Runnable class, instead invoke the start() method of Thread class.

Next, let's inspect the run() method

```
  public void run() {

    try {

      displayText("Hello Psychophysicist (Threaded)");
      updateScreen();
```

```
      Thread.sleep(2000);
      blankScreen();
      hideCursor();
      BufferedImage bi1 = ImageIO.read(new File("psychophysik.png"));
      displayImage(bi1);
      updateScreen();
      Thread.sleep(2000);
      blankScreen();
      BufferedImage bi2 = ImageIO.read(new File("fechner.png"));
      displayImage(0,0,bi2);
      updateScreen();
      Thread.sleep(2000);
    } catch (IOException e) {
      System.err.println("File not found");
      e.printStackTrace();
    } catch (InterruptedException e) {
      Thread.currentThread().interrupt();
    }
    finally {
      closeScreen();
    }
  }
```

This portion of the code is the same as the corresponding portion of the HelloPsychophysicist class from the previous Chapter. With only one difference: notice that here we directly invoke the methods of FullScreen1 class, for example

```
displayText("Hello Psychophysicist (Threaded)");
updateScreen();
```

instead of

```
fs.displayText("Hello Psychophysicist");
fs.updateScreen();
```

We are able to this, because the run() method is in the HPThreaded class, which inherits from the FullScreen1. Just as the methods in the FullScreen1 class doesn't need an explicit object reference to invoke each other, the calls to FullScreen1 methods from within the run() method of HPThreaded also doesn't need an explicit object reference. (Nevertheless there is a special keyword **this,** which could be used to make an explicit reference to an object, for example **this.updateScreen();**)

3.2. Summary

Here are the steps to take to write a Threaded program

1. Prepare a class which *implements* the Runnable interface (say RunnableTest)
2. Place the code which performs the task in the run() method of RunnableTest class

3. Construct an object of RunnableTest class:

```
RunnableTest rt = new RunnableTest();
```

4. Construct a Thread with that RunnableTest object:

```
Thread experiment = new Thread(rt);
```

   This allocates a new Thread and the argument is the object whose run method is going to be invoked. In this case the argument is conveniently an HPThreaded object.

5. Finally start the Thread:

```
experiment.start();
```

This was a very brief introduction to multithreading, see Chapter XXX for more complex examples. In this Chapter we also established a more convinient coding style. This style saves us some bookkeeping and results in clearer code. In the remaining of the Guide I will follow this convention of style by performing the 5 steps mentioned above.