

CIS 27P

Class 8

Applets

- Architecture
- Security
- Event Driven
- Extends `java.applet.Applet`
- `paint(Graphics g)` must be implemented
- Defined by an applet tag
- Run using browser or appletviewer

Applets (cont)

- No need for a main()
- Applet console
- upon startup
 - init()
 - start()
 - paint(Graphics g)
- upon termination
 - stop()
 - destroy()

Applet Methods

- `stop()` also called when user leaves web page
- `start()` also called when user re-enters web page
- `update()` can be overridden
- `drawString(String msg, int x, int y)`
- `setBackground(Color c)`
- `setForeground(Color c)`
- `showStatus(String msg)`

repaint()

- Calls update()
- Used by applet code when painting is necessary
- forms:
 - repaint()
 - repaint(int x, int y, int w, int h)
 - repaint(long maxDelay)
 - repaint(long maxTelay, int x, int y, int w, int h)

Applets with Threads

- implement Runnable
- start a thread in start()
- allow for stop()
- implement run()
- call repaint()

in start()

// Start thread

```
public void start() {  
    t = new Thread(this);  
    stopFlag = false;  
    t.start();  
}
```

in stop()

```
// Pause the banner.  
public void stop() {  
    stopFlag = true;  
    t = null;  
}
```

in run()

```
public void run() {  
    char ch;  
    for( ; ; ) {  
        try {  
            repaint();  
            Thread.sleep(250);  
            ch = msg.charAt(0);  
            msg = msg.substring(1, msg.length());  
            msg += ch;  
            if(stopFlag) break;  
        } catch(InterruptedException e) {}  
    }  
}
```

Parameters

- Add the following to `<applet>` tag:
 - `<param name=param1 value=val>`
- Access using:
 - `getParameter("param1")`

Exercise

Write an applet that counts down from 10 to 0 in one second increments and then displays **LIFTOFF!!!**

```
import java.awt.*;  
import java.applet.*;
```

```
public class LiftOff extends Applet implements  
    Runnable {
```

```
    Thread t = null;  
    int counter;  
    boolean stopFlag;
```

```
    public void init() {  
        counter=10;  
    }
```

```
    public void start() {  
        t = new Thread(this);  
        stopFlag = false;  
        t.start();  
    }
```

```
public void run() {  
    for( ; counter >= 0 ; counter-- ) {  
        try {  
            repaint();  
            Thread.sleep(1000);  
            if(stopFlag) {break;}  
        } catch(InterruptedException e) {}  
    }  
}
```

```
public void stop() {  
    stopFlag = true;  
    t = null;  
}
```

```
public void paint(Graphics g) {  
    String msg= (counter!=0)?String.valueOf(counter)  
        : “Liftoff!!!”;  
    g.drawString(msg, 10, 30);  
}  
}
```

Streams

- InputStream and OutputStream abstract classes implement a Byte Stream
- Reader and Writer abstract classes implement a Character Stream
- All concrete stream classes implement the abstract read() and write() methods.

Implicit Streams

- `System.in` is an `InputStream` reference
- `System.out` is `PrintStream`
`System.err` is `PrintStream`

Reading Characters

// Use a BufferedReader to read characters from the console.

```
import java.io.*;
```

```
class BRRead {  
    public static void main(String args[])  
        throws IOException  
    {  
        char c;  
        BufferedReader br = new  
            BufferedReader(new InputStreamReader(System.in));  
        System.out.println("Enter characters, 'q' to quit.");  
  
        // read characters  
        do {  
            c = (char) br.read();  
            System.out.println(c);  
        } while(c != 'q');  
    }  
}
```

Reading Strings

```
// Read a string from console using a BufferedReader.
```

```
import java.io.*;
```

```
class BRReadLines {
```

```
    public static void main(String args[])
```

```
        throws IOException
```

```
{
```

```
    // create a BufferedReader using System.in
```

```
    BufferedReader br = new BufferedReader(new  
        InputStreamReader(System.in));
```

```
    String str;
```

```
    System.out.println("Enter lines of text.");
```

```
    System.out.println("Enter 'stop' to quit.");
```

```
    do {
```

```
        str = br.readLine();
```

```
        System.out.println(str);
```

```
    } while(!str.equals("stop"));
```

```
}
```

```
}
```

Writing to Console

```
import java.io.*;
public class PrintWriterDemo {
public static void main(String args[])
{
    PrintWriter pw = new
        PrintWriter(System.out, true);
    pw.println("This is a string");
    int i = -7;
    pw.println(i);
    double d = 4.5e-7;
    pw.println(d);
}
}
```

Write to a File

```
import java.io.*;

class WriteFile {
    public static void main(String args[]) throws
        IOException
    {
        String fileName = args[0];
        FileWriter fw=null;
        try
        {
            fw = new FileWriter(fileName, true);
            fw.write("This will be appended to the file.\n");
        }
        catch ( IOException ioe )
        {
            System.out.println("Exception while writing.");
            ioe.printStackTrace();
        }
        finally { fw.close(); }
    }
}
```

Reading From a File

```
import java.io.*;
class ReadFile {
    public static void main(String args[]) throws
        IOException
    {
        String fileName = args[0];
        FileReader fr=null;
        try
        {
            fr = new FileReader(fileName);
            for ( int i=0;( i=fr.read()) != -1; )
            {
                System.out.print( (char)i );
            }
        }
        catch ( IOException ioe )
        {
            System.out.println("Exception while reading.");
            ioe.printStackTrace();
        }
        finally {fr.close(); }
    }
}
```

Redirect Stack Trace

```
import java.io.*;
class RedirectStackTrace {
    public static void main(String args[]) throws IOException
    {
        String fileName = args[0];
        FileWriter fw = null;
        PrintWriter pw = null;
        try
        {
            fw = new FileWriter( fileName, true);
            pw = new PrintWriter( fw );
            int a = 5 / 0;
        }
        catch ( Exception e )
        {
            e.printStackTrace( pw );
        }
        finally
        {
            fw.close();
            pw.close();
        }
    }
}
```

Miscellaneous

- variable modifiers
 - transient: the variable is not part of the persistent state of the object
 - volatile: the variable may be accessed by unsynchronized threads
- instanceof
 - if (sportsCar instanceof Vehicle)