Recitation 09. Applications and Applets

Java applications

Those who took CS100J and CS100M in fall 2003 may not have had much experience with Java *applications*. A Java application is simply a collection of classes one of which contains one method with this header:

public static void main(String[] pars)

In a terminal or DOS window, one executes an application whose method main is in class C (file C.java) by navigating to the directory containing the program files and executing

java C

You can type this in the DrJava Interactions pane, too.

The argument to main can be an array of Strings, and method main can use these as input to it. We don't handle that here.

jar files (Java Archive files)

A program may consist of many classes, generally in the same directory. If you want to give someone your program, you have to give them all the .class files —zip them up into a .zip file— and the someone must then unzip them. That's messy. To make things easier, you can make a jar file of the classes. Jar stands for Java ARchive; after tar files (TapeARchives) on Unix systems.

To make a jar file, get into a DOS or command-line window and navigate to the directory where the .java and .class files are. Then, type in this command:

jar -cf file-name.jar *.class

The c is for create. The f is for file and indicates that the name of the file to create follows: file-name.jar. The *.class is expanded to name all the .class files in the directory. So, this command makes up a jar file named file-name.jar that contains all the .class files in the directory.

You still have to insert into the jar file something that tells it which class has method main. Suppose it is class MainClass. Then do the following:

(a)Make up a file x.mf that contains one line of text in it:

Main-class: MainClass

The suffix on x.mf stands for manifest. You actually don't have to use the suffix. Make sure you hit the enter key after typing the text; there must be a carriage-return or line-feed in it. You can create this file in wordpad or notepad or DrJava or any editor you want. Make sure the file is in the same directory as file file-name.jar.

(b) Type this command in the window:

jar -umf x.mf file-name.jar

The u stands for update, the m for manifest, and the f for file. Since the m comes before the f, the manifest file name, x.mf, comes before the file name. This command inserts into jar file file-name.jar the fact that method main appears in class MainClass.

You can do both steps together —insert the classes and the main-class indication— by using the command

jar -cmf x.mf file-name.jar *.class

You can now email file file-name.jar to anyone, and they can run it on their computer, whether it has a Unix, Macintosh, or Windows system, as long as their system has Java in it. To execute the program in the jar file, type this line (yes, include the extension .jar):

java -jar file-name.jar

In some systems, you will also be able to run the program just by double-clicking on the jar file.
If you want to see what is in jar file file-name.jar, then type this:

jar tvf file-name.jar

You can find out more about the jar command by typing simply jar and hitting the return/enter key.

Recitation 09. Applications and Applets

Applets. An *applet* is a Java program that is started by a browser (e.g. netscape or internet explorer) when an html file has a suitable applet tag. Here, we explore the difference between applications (the kind of programs we have been writing) and applets.

Classes Applet and JApplet. Any Java program that is to be executed by a browser has to be a subclass of either Applet (in package java.awt) or JApplet (in the newer Swing package, javax.swing).(Class JApplet is a subclass of Applet).

Here is a tutorial on writing Japplets:

http://java.sun.com/j2se/1.3/docs/api/index.html

An application is started by calling static method main, right?

An applet, instead, has five methods, all of which are nonstatic, that are called by the browser at various times. Som of these should be overridden for the applet to do its job. The inherited versions of these method do nothing (but very fast).

• Method *init()* is called to tell the Java program that it has been loaded into the system. It is always called before the first time that method *start* is called. Override this to perform initialization (e.g. start a thread).
.
• Method *start()* is called by the browser to inform this applet that it should start its execution. It is called after method *init* and each time the applet is revisited in a Web page. (E.g. if the applet window is hidden, method *stop* is called, which could stop anything that the applet is doing, like providing an animation. When the window becomes visible again, method *start* is called, and it can continue execution).

• Method *stop()* is called when the applet should stop whatever it is doing because the applet is no longer visible --see the discussion of method *start*.

• Method *destroy* is called just before the applet is terminated by the browser. In method destroy, the applet can kill its resources --e.g. threads that were started in method *init*.

• Method *paint(Graphics g)*. An applet occupies a space in the window for the html page. It can be painted, just like any Panel.

Our applets will not use start, stop, and destroy.

An html file resides on one computer and is sent to your computer, where the browser loads it and shows it in a window. The applet typically is on the same computer (in the same folder) as the html page. It, too, is sent to your computer, where your browser executes it.

You know about tags like <p> and on html pages. Consider these two tags

```
<applet codebase="Java Classes"
    code="TrivialApplet.class"
    width=200 height=200>
</applet>
```

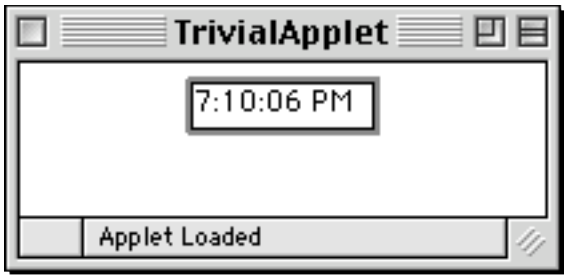
These tell the browser to start a Java applet, which is in TrivialApplet.class. The applet is a 200x200 pixel square.

The argument codebase="Java Classes" tells the browser that the .class files for this program are in folder "Java Classes", which must be in the same folder as the html page that contains the tag.

If there are a lot of class files, the above can be inefficient, because each file is individually retrieved from the place where the html file resides and brought to your compueter. You can have CodeWarrior store all the .class files in compressed form in a single file called a "jar" file, so that only one file has to be retrieved. To do this, in Codewarrior, click on the target setting icon, select "Java output", and then set the "Output type" to "Jar File". Then, Codewarrior will produce a jaf files instead of a folder of .class files.

Then, use these tags instead of the ones above:

```
<applet archive="JavaClasses.jar"
code="TrivialApplet.class" width=200 height=200>
</applet>
```



This is the clock applet, viewed in appletviewer

Here's a simple applet that puts some text and a horizontal line on the web page.

```
public class Apple extends JApplet {
    // Initialize the applet
   public void init() {   }

   public void paint(Graphics g) {
     g.drawString( "Hello World!", 30, 30 );
     g.drawLine(30-2, 30+2, 30+70, 30+2);
   }
}
```

This applet puts a clock in the window.

```
/* An applet that puts a clock in the window */
import java.awt.*;
import java.applet.Applet;
import java.util.*;
import java.text.*;

public class TrivialApplet extends Applet
        implements Runnable {
   private final static int SIZE= 8;  // No chars in clock
   private final static int DELAY= 1000;    // sleep
                  // time: 1 second
   private TextField clock; // The clock itself
   private Thread timer;     // Thread that runs the clock

    public void start();
      {  timer= new Thread(this);   timer.start(); }

    public void run() {
      Date time;          // The current time
      SimpleDateFormat f=
            new SimpleDateFormat("h:mm:ss a");
     Thread thisThread = Thread.currentThread();
      while (timer != null) {
         time= new Date();
         clock.setText(f.format(time));
         try {
            Thread.currentThread().sleep(DELAY);
         }
         catch (InterruptedException e) {}
      }
    }

    public void stop() { timer= null; }

    public void init()
      {  clock= new TextField(SIZE); add(clock); }
}
```

As discussed in lecture, a class that implements Runnable has to have method run().

Every time start is called by the browser, it creates a Thread that is attached to this instance. Then, calling timer.start results in method run being called.

An instance of class Date contains the time at which it was created. An instance of class SimpleDateFormat describes how to present the date as a String. Here, "h stands for hour, "m" for minute, "s" for second, and "a" for AM-PM.

Method run, and thus the thread, continues until timer becomes null, which means that stop was called. Then, method run, and thus the thread. stops.

The browser calls method init, and then start, which starts the thread going. When the applet window becomes hidden, the browser calls method stop, which set timer to null.

Method init is called by the system. Init creates a TextField, which will contain the time on the clock, and adds it to the applet (much like one adds labels and buttons, etc., to a Frame).

HyperTextMarkup Language (HMTL)

```
<html>
   <head>
      <title>FacultyApplet</title>
  </head>
  <body>
      <p align="center"><B>This</B> is
                an <i>Applet!</i>
     </p>
     <br><br>

      <p><applet archive="AppletClasses.jar”
            code="FacultyApplet.class”
           width=800    height=550>
          </applet>
     </p>

  </body>
</html>
```

tags

<html> start an html page

<head> start the “heading”

<title> the title for the page

<body> start the body, content of the page

<p> begin a paragraph

<b> begin boldface

<i> begin italics

<applet> start a Java applet

**
** line break (no end tag)