

CS1Bh Lecture Note 14

Applets

The Java language provides a restricted subclass of programs called *applets*. The name is intended to be suggestive of “a small application”¹. Applets are used to provide executable program code which can be embedded in a Web page, allowing the readers of the page to interact with it, making a Web page more than just text on a screen.

Many different programming languages can be used to provide embedded program content in a Web page but Java was the first language which could be used for this purpose. However, its importance in this area is more than merely historical. Java’s applets provide well-understood guarantees about restrictions which are placed on the effects which an applet can have. Placing these restrictions on applet code is called *sandboxing*². One of the sandboxing restrictions is that an applet cannot read or write files on the local file system, thereby preventing a malicious author of an applet from inflicting damage on the local file system of the computer onto which the applet was downloaded. Sandboxing also prevents an applet from sending email messages or opening network connections to computers other than the one from which it was originally downloaded. There are many other similar sandboxing restrictions.

14.1 Applets and security

Sandboxing prevents applets from being able to provide many useful functions. For example, one cannot program a text editor as an applet (because one would be unable to save the text to a file). Similarly, one cannot program email clients or interactive ‘chat’ applications as applets. Why then is sandboxing useful? In order to answer this question, we could compare Java with a language which has no equivalent notion of sandboxing such as Microsoft’s VBScript. It can be used to program embedded functionality in Web pages and can also be sent as *attachments* to email messages.

On Thursday, 4th May 2000, a young man in Manila in the Philippines sent a computer virus in the form of a VBScript program attached to a brief email message which said “Kindly check the attached love letter coming from me”. On the 7th of May he was arrested. His “love bug” virus spread by re-sending itself to everyone in the recipient’s email address book, causing email systems to become overloaded. It is

¹ The -let suffix suggests the small part, for example, a piglet is a small pig.

² The analogy is with a sand-filled box in a children’s playground. Its function is to provide a safe place in which to play.

estimated to have crashed one tenth of the world's mail servers. In the UK this included those operated by the BBC, News International, a number of FTSE-100 companies, the Houses of Parliament and, rather poignantly, also Microsoft's mail server. The virus downloads more harmful software from a remote Web site, renames files and redirects Web browsers. It installs itself in the operating system of the machine so that it can persist even if the machine is rebooted in an attempt to clear it. Note that none of this harm could have been done by an applet.

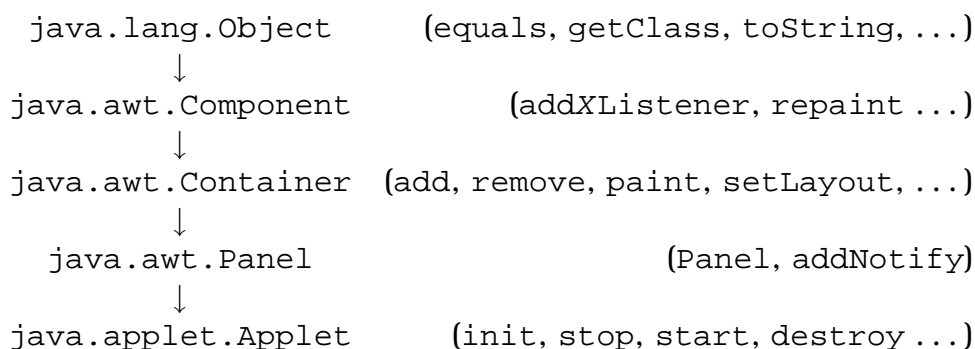
14.2 The security manager

The responsibility for policing the sandbox restrictions in the Java run-time system resides with the `java.lang.SecurityManager` class. The sandbox model works because all of the core Java classes which perform security-sensitive operations first ask permission of the resident `SecurityManager`. If the sensitive operation is being invoked either directly or indirectly from untrusted code then the `SecurityManager` throws an exception to prevent the operation taking place. Applet viewers and Web browsers which support Java create a subclass of `SecurityManager` to enforce their desired level of control. When an applet is loaded from the local file system, many of the sandbox restrictions may be lifted because the assumption is that local code is more trustworthy.

One of the few possible ways in which an applet can attack its host computer is a *denial of service* attack. In this, an applet needlessly consumes memory or wastes CPU cycles in order to degrade the quality of service which is provided by the host computer. This type of attack is very difficult to prevent because it is impossible for the Java run-time system to distinguish between a program which needlessly wastes resources and one which genuinely requires a lot of resources.

14.3 Applets and the class hierarchy

The `Applet` class is four levels deep in the class hierarchy, extending the AWT class `Panel`, which in turn extends the abstract class `Container`, which extends `Component`, which extends `Object`.



Method `init` is invoked when the applet is first downloaded by a browser, while `start` is invoked every time the applet is executed.

14.4 An Example: Counting Mouse Clicks

The following applet counts mouse clicks. It has a private integer field `clicks` which is used to record the number of clicks. (All variables in Java are initialised, and integers are initialised to zero.) The invocation of the `repaint()` method below will cause the `paint()` method to be invoked later.

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

public class Clicker extends Applet {
    private int clicks; // Click counter

    // Print the number of clicks on the screen
    public void paint(Graphics g) {
        g.setColor(Color.white);
        g.fillRect(0, 0, 400, 400);
        g.setColor(Color.black);
        g.setFont(new Font("TimesRoman", Font.BOLD, 64));
        g.drawString("Clicks: " + clicks, 60, 240);
    }

    public void init() {
        addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                // We register mouse click and repaint
                clicks++;
                repaint();
            }
        });
    }
}
```

In order for an applet to be displayed within a Web page we also need to compose the text of the Web page. The language HTML (HyperText Markup Language) is used for this. The size of the applet display can now be set within the HTML, not inside the Java code. It is necessary for this information to be known in the HTML page so that a Web browser can leave the required amount of space for the applet's display to be painted. The HTML page used to present this applet is shown below.

We can view this file in one of two ways. Either we use the `appletviewer` command from SUN's Java Development Kit:

```
appletviewer Clicker.html
```

or we can visit the page with a Java-enabled Web browser in the conventional way.

```
<html>
<head>
```

```

    <title>The Clicker Applet</title>
</head>

<body>
    <h1>The Clicker Applet</h1>

    <applet code="Clicker.class" width="400" height="400">
    </applet>

    <hr>
</body>
</html>

```

14.5 A Case Study: Noughts & Crosses

We now develop an applet which plays noughts and crosses. In this simple version, the player is always “X” and plays first. The computer is always “O”. This requires us to

- design and implement an internal representation of the game and the moves made by the computer;
- design and implement a GUI which allows us to interact with the internal representation.

Internal Representation

A noughts & crosses grid has three rows of three squares. We will think of these as being indexed from 0 to 8, starting at the top left and working row by row to the bottom right. We will then represent the moves already made by a player as an array of nine booleans, in which the value `false` indicates that the given square has not been used by this player, and `true` indicating that it has been used. We have one such array for the computer and one for the human player. For convenience, we also store a count of the total number of moves made so far.

```

public class OXO extends Applet {
    int movesmade;

    boolean player[] = new boolean[9]; // Player's positions
    boolean computer[] = new boolean[9]; // Computer's positions

```

We then write methods which check whether a proposed move, represented by the integer number of its square, is allowable (in other words, we check that this square hasn't been used already) and another method which checks if a given player (represented by his/her/its moves so far) has won the game. There are seven ways to win.

```

boolean MoveOK (int m) {
    return !(computer[m] || player[m]);
}

// Check all the possible ways of winning
boolean won (boolean []moves) {
    return (moves[0] && moves[1] && moves[2]) ||
           (moves[3] && moves[4] && moves[5]) ||
           (moves[6] && moves[7] && moves[8]) ||
           (moves[0] && moves[3] && moves[6]) ||
           (moves[1] && moves[4] && moves[7]) ||
           (moves[2] && moves[5] && moves[8]) ||
           (moves[0] && moves[4] && moves[8]) ||
           (moves[2] && moves[4] && moves[6]);
}

```

The method used by the computer to determine its next move is naive to say the least! It simply tries random squares until an unused square is found. You might like to try to implement a more sophisticated strategy.

```

void computerMove () {
    // Just play in the first empty square found randomly
    // Can you do better? Shouldn't be hard!

    int m;
    m = (int) (Math.random()*9);
    while (!MoveOK(m)) {
        m = (int) (Math.random()*9);
    }
    movesmade++;
    computer[m] = true;
}

```

GUI Representation

The `start` method sets up the initial empty board, then waits for the player to make a move. This uses the event driven approach from previous lectures and a little geometry to work out which square has been clicked. If the move is valid it is made, and then, unless the game is over, the computer responds with its own move. The `repaint()` method is invoked to draw the updated game.

```

public void start() {
    movesmade = 0;
    for (int i=0; i<9; i++) {
        computer[i] = false;
        player[i] = false;
    }
    this.addMouseListener(new MouseAdapter() {
        public void mouseReleased(MouseEvent e) {
            int x = e.getX();
            int y = e.getY();

            // Work out which square we are in, r for row, c for column
            // Its number on the grid will be r*3 + c
            Dimension d = getSize();
            int r = (y * 3) / d.height;
            int c = (x * 3) / d.width;

            if (MoveOK(r*3 + c) && !won(player) && !won(computer)) {
                player[r*3+c] = true; // Make the move
                movesmade++;
                repaint();
                // Computer's turn now, unless game is over
                if (!won(player) && (movesmade < 9)) {
                    computerMove();
                    repaint();
                }
            }
        }
    });
}

```

Method `paint` draws the grid in the obvious way, then adds the symbols for the moves made already by looking up the internal representation.

```
public void paint(Graphics g) {
    Dimension d = getSize();
    setBackground(Color.white);
    g.setColor(Color.black);
    int xoff = d.width / 3;
    int yoff = d.height / 3;

    // Draw the grid
    g.fillRect(xoff-2, 0, 4, d.height);
    g.fillRect(2*xoff-2, 0, 4, d.height);
    g.fillRect(0, yoff-2, d.width, 4);
    g.fillRect(0, 2*yoff-2, d.width, 4);

    // Draw the moves made so far
    for (int r = 0 ; r < 3 ; r++) {
        for (int c = 0 ; c < 3 ; c++) {
            if (player[r*3+c]) {
                drawX(g, r, c, d.width/3, d.height/3);
            } else if (computer[r*3+c]) {
                drawO(g, r, c, d.width/3, d.height/3);
            }
        }
    }
}
```

Methods `drawX` and `drawO` make straightforward use of the Graphics toolkit to draw the corresponding symbols.

```
void drawO (Graphics g, int r, int c, int boxw, int boxh) {

    g.setColor(Color.red);
    g.fillOval(c*boxw + boxw/4, r*boxh + boxh/4, boxw/2, boxh/2);
    g.setColor(Color.white);
    g.fillOval(c*boxw + boxw/4 + boxw/10, r*boxh + boxh/4 + boxw/10,
              boxw/2 - boxw/5, boxh/2 - boxw/5);
}

void drawX (Graphics g, int r, int c, int boxw, int boxh) {

    Polygon p1 = new Polygon();
    Polygon p2 = new Polygon();

    // Top left to bottom right swish
    p1.addPoint(c*boxw + boxw/4, r*boxh + boxh/4);
    p1.addPoint(c*boxw + boxw/4, r*boxh + boxh/4 + boxw/5);
    p1.addPoint((c+1)*boxw - boxw/4, (r+1)*boxh - boxh/4);
    p1.addPoint((c+1)*boxw - boxw/4, (r+1)*boxh - boxh/4 - boxw/5);

    // Bottom left to top right swish
    p2.addPoint(c*boxw + boxw/4, (r+1)*boxh - boxh/4);
    p2.addPoint(c*boxw + boxw/4, (r+1)*boxh - boxh/4 - boxw/5);
    p2.addPoint((c+1)*boxw - boxw/4, r*boxh + boxh/4);
    p2.addPoint((c+1)*boxw - boxw/4, r*boxh + boxh/4 + boxw/5);

    // Now draw them
    g.setColor(Color.blue);
    g.fillPolygon(p1);
    g.fillPolygon(p2);
}
```

Murray Cole, 27th February 2003.