COMP519
Web Programming
Autumn 2015
Combining Java and JavaScript in Webpages
JavaScript vs. Java

JavaScript is very good for simple tasks and simple GUI layout can be handled by CSS.

Flexible data typing, primitive object types are fine for quick development.

Integration with HTML makes layout and control of GUI elements via DOM manipulation possible.

Drawbacks: Only primitive data structuring capabilities.

Not well-suited to multi-file projects and complex OO approach.
Java is better at complex tasks, especially graphics.
  ▶ Full-featured, more robust, extensive libraries of classes/routines.
  ▶ Can support large projects, interacting objects.

GUI layout in a webpage can difficult, integration with HTML not obvious.

IDEALLY: we can make use of the the strengths of each language.

Include Java applets in a page when needed (e.g., graphics).

Allow communication between applet and JavaScript code.
Calling Java Routines from JavaScript

On some previous versions of web browsers, you used to be able to call (some) Java methods directly inside a JavaScript block.

There were additional limitations, such as not being able to call Java functions to read and write from the local hard drive (for obvious security reasons).

This functionality seems to have largely disappeared from modern versions of web browsers.

There is some additional technology, such as the Java Server Pages, which can give web programmers more of the expressive and computational power of Java. However, this requires a web server with a compatible “servelet container” for this to work (we don’t have that technology installed on the web servers here).
Java Applets

These days, we generally include a more complicated applet in a page to perform some task, and control the applet via HTML events, event handlers, and JavaScript.

This requires a Java virtual machine (or Java runtime environment) to be installed on the client’s machine, i.e. one that can execute Java bytecode.

To call a Java applet method from JavaScript

```javascript
document.appletName.methodCall(…)
```

What follows is a simple “Hello World” applet.
Java source code:

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

/**
 * This class displays "Hello world!" on the applet window.
 */
public class HelloWorld extends Applet
{
    public void paint(Graphics g)
    {
        // writes starting at pixel row 15, col 15
        g.drawString("Hello world!", 15, 15);
    }
}
```

Compile this source code as normal into a Java class.
"Hello World" Java Applet (cont.)

<html>
<!-- COMP519  HelloWorld.html  2015.10.19 -->
<head>
  <title>Hello World Page</title>
  <style type="text/css">
    table { border: 3px double;
           margin-left: auto; margin-right: auto; }
  </style>
</head>
<body>
  <table>
    <tr><td>
      <object codetype="application/java"
             codebase="http://www.csc.liv.ac.uk/~martin/teaching/comp519/Java/"
             classid="java:HelloWorld.class"
             name="HelloApplet" height="30" width="100">
        You must use a Java-enabled browser to view this applet.
      </object>
    </td></tr>
  </table>
</body>
</html>
Browser differences...

As one might expect, this may not really work on all browsers, due to the way different browsers have implemented the `<object>` element.

There are pure HTML methods you can use that (are supposed to) work for different browsers, or you can use JavaScript to detect different browsers and write appropriate HTML code depending upon the browser.

The previous example actually contains more code than was shown, as I used some JavaScript to detect the type of browser, and use (I hope) suitable HTML code that would work to load the applet. (Of course, this also relies upon the user allowing JavaScript to run in his/her browser.)

You might also have to deal with “deploying” your applet in a way to deal with security issues. (Most browsers give warnings for “unsigned” applets.)
Browser differences... (cont.)

```html
<script>
    var _app = navigator.appName;
    if (_app == 'Microsoft Internet Explorer') // ** Handle IE
    {
        document.write('<object
            classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93" ',
            'codetype="application/java" ',
            'codebase="http://www.csc.liv.ac.uk/~martin/teaching/comp519/Java/" ',
            'name="HelloApplet" height="20" width="100"> ',
            '<param name="code" value="HelloWorld.class" /> ',
            'You must use a Java-enabled browser to view this applet. ',
            '</object>');
    }
    else /* Handle all other browsers (I hope!) */
    {
        document.write('<object codetype="application/java" '麼,
            'codebase="http://www.csc.liv.ac.uk/~martin/teaching/comp519/Java/" ',
            'classid="java:HelloWorld.class" ',
            'name="HelloApplet" height="20" width="100"> ',
            'You must use a Java-enabled browser to view this applet. ',
            '</object>');
    }
</script>
```
In the examples that follow, I will include JavaScript code like the previous case in the HTML source code, but (for purposes of simplification) won’t include all of that in the example code shown.

I will only show the syntax that works for (all?) browsers other than Internet Explorer, but it’s easy to replicate the JavaScript code for the previous example.
Suppose we return to the example on permutations.....
We had a JavaScript method to perform this task. Let’s replace that method with a Java applet.

We can reuse much of the HTML code and need to supply Java code to perform the operation for us.

We continue using JavaScript methods to access the input (and output) fields on the HTML page, only using Java for the actual “calculation”.
import java.awt.*;
import java.applet.*;
import java.util.Random;
import java.lang.Math.*;
/**
 * This class can be used to generate a permutation to be
 * inserted into a webpage.
 */
public class JavaPermutation extends Applet
{
    public String permutation(int n)
    {
        Random r = new Random();
        int[] p = new int[n];
        int i, k, temp;
        for (i = 0; i < n ; i++)
            p[i] = i+1;
        double finish = Math.pow(n,3) * Math.log(n) * 12;
        for (i = 1; i <= finish; i++)
        {
            if (r.nextDouble() < 0.5)
            {
                k = r.nextInt(p.length - 1);
                temp = p[k];
                p[k] = p[k + 1];
                p[k+1] = temp;
            }
        }
    }
}
We compile this code as normal to make a Java class.

As stated, we can reuse much of the HTML code, we just need to add the part to insert the applet in the document and call the Java function (and we can remove the non-used JavaScript code).
Permutations (cont.)

<html>
<head>
  <title>Generating random permutations</title>
  <style type="text/css">
    p { text-align: center; }
  </style>
  <!-- Script for verifying that a textbox contains an integer in a specified range. -->
  <script src="verify.js"></script>
  <!-- Scripts for other functions used in this page -->
  <script>
    function checkForEnter(Box, event) {
      if ( (event.keyCode == 13) && (Box.value != "") ) {
        return true;
      }
      return false;
    }
    function Process() {
      var n = document.getElementById('N');
      if (VerifyIntInRange(n, 1, 50)) {
        var perm = document.PermApplet.permutation(parseInt(n.value));
        var out = document.getElementById('Output');
        out.value = perm;
      }
      n.focus();
    }
  </script>
</head>
</html>
Here is a function that will generate random permutations of a set of integers, where a permutation is just an arrangement of the integers \{1, 2, \ldots, n\}.

Enter a positive integer \( n \) (1-50): 

```
<input type="text" id="N" size="1" maxlength="2" value="15">
```

```
<input type="button" value="Go!" id="go">
```

```
<br><br>
Random permutation of \{1, \ldots, n\}.  
<textarea id="Output" rows="10" cols="60"></textarea>
```

```
A (much?) more complicated example

Consider MontePI example:

- We want to draw random dots inside a square (with an inscribed circle).
- We could build GUI interface into applet, but requires tricky layout manager.

Instead, we will leave the graphics up to the applet, and control the operation of the applet via JavaScript.
import java.awt.*;
import java.applet.*;
import java.util.Random;

public class Monte6 extends Applet
{
    private static Random randy;
    private int SIZE;
    private Image offScreenImage;
    private Graphics offScreenGraphics;
    private int randomInRange(int low, int high) { ... }
    private double distance(int x1, int y1, int x2, int y2) { ... }

    public void init()
    {
        randy = new Random();
        Dimension dim = getSize();
        SIZE = dim.width;
        drawCircle();
    }

    public void drawCircle()
    {
        // DRAWS CIRCLE ON BOTH getGraphics() AND
        // offScreenGraphics
    }

    public void drawDots(int numPoints)
    {
        // DRAWS numPoints RANDOM DOTS ON BOTH getGraphics()
        // AND offScreenGraphics
    }

    public void paint(Graphics g)
    {
        g.drawImage(offScreenImage, 0, 0, null);
    }
}
A Brief Explanation of the Applet

init creates the random number generator and gets applet size.

drawDots draws the dots on the screen and to the off-screen buffer.

paint redraws the screen using the buffer.

Interested readers can download the Java applet code (and accompanying webpages to run the applet) and can view the code to perform these tasks.
<html>
   <!-- COMP519 Monte6.html 2015.10.20 -->
<head>
   <title>Monte Carlo Darts Page</title>
   <style type="text/css">
      body { background-color: gray; }
   </style>
</head>
<body>
   <div style="text-align: center;">
      <object codetype="application/java"
          codebase="http://www.csc.liv.ac.uk/~martin/teaching/comp519/Java/"
          classid="java:Monte6.class" name="MonteApplet"
          height="300" width="300">
         You must use a Java-enabled browser to view this applet.
         </object>
         <form id="MonteForm">
            <p> <input type="button" value="Generate points" onclick="document.MonteApplet.drawDots(500);"/> </p>
         </form>
   </div>
</body>
</html>
A Better Interface to the Applet

We can allow the user to specify the number of dots in an input text box.

Each click adds new dots using the user input value.

We also add a separate button to clear the picture and start again.

Note that the Java code itself doesn’t change, but the HTML is where these changes are implemented.
<html> <!-- COMP519 Monte6a.html 2015.10.20 -->
<head>
    <title>Monte Carlo Darts Page</title>
    <style type="text/css">
        body { background-color: gray; }
    </style>
</head>
<body>
    <div style="text-align: center;">
        <object codetype="application/java"
            codebase="http://www.csc.liv.ac.uk/~martin/teaching/comp519/Java/
            classid="java:Monte6.class"
            name="MonteApplet" height="300" width="300">
            You must use a Java-enabled browser to view this applet.
        </object>
        <form id="MonteForm">
            <p> <input type="button" value="Generate"
                onclick="numDots =
                parseFloat(getElementById('numPoints').value);
                document.MonteApplet.drawDots(numDots);">
                <input type="text" id="numPoints" size="6" value="500">
                points <br>
                <input type="button" value="Clear the screen"
                    onclick="document.MonteApplet.drawCircle();">
            </p>
        </form>
    </div>
</body> </html>
Where the control lies affects the efficiency/usability of an applet.

We want the applet to be as self-contained as possible to take advantage of speed and more advanced of Java.

But if the GUI controls are in HTML, then JavaScript needs overall control.

Let us consider adding counters for the number of dots inside and outside the circle.

We can have the applet keep track of the dots in instance variables.

Call a method to draw all dots, then JavaScript accesses counts and displays them. This is fast, but only see counts when the drawing is done.
We could return more control to the page by having the applet draw one dot at a time.

The repetition is handled by JavaScript and we can update boxes after each dot. This is slower, but more flexible (and we can see counts change).

Alternatively, we could have the applet update the HTML text boxes itself. This is more tricky and ties the applet to the page as it must access the contents of various HTML elements.
Let us have the applet keep track of number inside and out.

Instance variables `numInside` and `numOutside` are initialized in `drawCircle` and updated in `drawDots`.

Since these variables are public, these instance variables can be accessed in the webpage.

We modify the MontePi Java applet to add these public variables and alter the methods to update them as appropriate.
import java.awt.*;
import java.applet.*;
import java.util.Random;
public class Monte7 extends Applet
{
    public int numInside, numOutside;
    public void drawCircle()
    {
        numInside = 0; numOutside = 0;
        ...
    }
    public void drawDots(int numDots)
    {
        ...
        for (int i = 0; i < numPoints; i++) {
            int x = randomInRange(0, SIZE);
            int y = randomInRange(0, SIZE);
            if (distance(x, y, SIZE/2, SIZE/2) < SIZE/2) {
                offScreenGraphics.setColor(Color.white);
                g.setColor(Color.white);
                numInside++;
            } else { offScreenGraphics.setColor(Color.black);
                      g.setColor(Color.black);
                      numOutside++;          }
        }
    }
}

<head>
	<title>Monte Carlo Darts Page</title>

<style type="text/css">
table { margin-left: auto; margin-right: auto; }
body { background-color: gray; }
</style>

<script>

function drawAll()
{
    var numDots = 
        parseFloat(document.forms[‘MonteForm’].numPoints.value);
    document.MonteApplet.drawDots(numDots);
    document.forms[‘MonteForm’].numIn.value = 
        document.MonteApplet.numInside;
    document.forms[‘MonteForm’].numOut.value = 
        document.MonteApplet.numOutside;
    document.forms[‘MonteForm’].estPi.value = 
        4.0*document.MonteApplet.numInside / 
        (document.MonteApplet.numInside 
            + document.MonteApplet.numOutside);
}

function clearAll()
{
    document.MonteApplet.drawCircle();
    document.forms[‘MonteForm’].numIn.value = 0;
    document.forms[‘MonteForm’].numOut.value = 0;
}

</script>

</head>
<form id="MonteForm">
  <table>
    <tr>
      <td>
        <object codetype="application/java"
             codebase="http://www.csc.liv.ac.uk/~martin/teaching/comp519/Java/
             classid="java:Monte7.class"
             name="MonteApplet" height="300" width="300">
          You must use a Java-enabled browser to view this applet. </object>
      </td>
      <td>
        <input type="button" value="Generate" onclick="drawAll();">
        <input type="text" name="numPoints" size="6" value="500"> points
        <hr>
        <input type="text" name="numIn" size="6" value="0"> points inside<br>
        <input type="text" name="numOut" size="6" value="0"> points outside
        <hr><input type="button" value="Clear the screen" onclick="clearAll()">
      </td>
    </tr>
  </table>
</form>
More “Animation”

As an alternative to the previous example, we could draw each dot individually, and update the counts after each dot.

Again, this is done using JavaScript, so the Monte7 applet isn’t modified, but the JavaScript that is controlling the execution is modified.

(Ok, in principle this should give a more “animated” webpage as the counts are updated after each dot is inserted. In practice, this doesn’t appear to occur, despite my efforts to increase the delay between inserting individual dots.)
An Alternative Method to Draw the Dots

```html
<!-- COMP519 Monte7a.html 2015.10.20 -->
<html>
  <head>  <title>Monte Carlo Darts Page</title>
  <script>
    function drawAll() {
      var numDots = 
        parseFloat(document.forms['MonteForm'].numPoints.value);
      for (var i = 0; i < numDots; i++) {
        document.MonteApplet.drawDots(1);
        document.forms['MonteForm'].numIn.value = 
          document.MonteApplet.numInside;
        document.forms['MonteForm'].numOut.value = 
          document.MonteApplet.numOutside;
        document.forms['MonteForm'].estPi.value = 
          4.0*document.MonteApplet.numInside / 
          (document.MonteApplet.numInside + 
          document.MonteApplet.numOutside);
      }
    }

    function clearAll() {
      document.MonteApplet.drawCircle();
      document.forms['MonteForm'].numIn.value = 0;
      document.forms['MonteForm'].numOut.value = 0;
    }
  </script>
</head>
```
<body style="background-color: gray;">
<form id="MonteForm">
<table style="margin-left: auto; margin-right: auto;">
<tr><td <object codetype="application/java"
    codebase="http://www.csc.liv.ac.uk/~martin/teaching/comp519/Java/"
    classid="java:Monte7.class"
    name="MonteApplet" height="300" width="300">
    You must use a Java-enabled
    browser to view this applet. </object> </td>
    <td><input type="button" value="Generate"
        onclick="drawAll();">  
        <input type="text" name="numPoints" size="6"
            value="500"> points  
    <hr>
    <input type="text" name="numIn" size="6" value="0">  
        points inside <br>
    <input type="text" name="numOut" size="6" value="0">  
        points outside  
    <hr>
    <input type="button" value="Clear the screen"
        onclick="clearAll()"> 
</td></tr>
</table> </form></body>
Accessing HTML/JavaScript from the Applet

It is possible for the applet to access elements in the page. This requires the `JSObject` class from the Java `netscape.javascript` package.

```java
import netscape.javascript.JSObject;

Use `getWindow` and `getMember` methods to access components.

```
JSObject jsWin = JSObject.getWindow(this); // GETS WINDOW
JSObject jsDoc = (JSObject) jsWin.getMember("document"); // GETS DOCUMENT
JSObject MonteForm = (JSObject) jsDoc.eval("document.forms[0]"); // GETS FORM
numInside = (JSObject) MonteForm.getMember("numIn"); // GETS TEXT BOX
```
Use `getMember` and `setMember` methods to access component attributes.

```java
int num=Integer.parseInt((String)numInside.getMember("value"));

numInside.setMember("value", "+(num+1));
```
import java.awt.*;
import java.applet.*;
import java.util.Random;
import netscape.javascript.JSObject; // need plugin.jar in classpath

public class Monte8 extends Applet
{

private JSObject numDots, numInside, numOutside;

public void init()
{
    try {
        JSObject jsWin = JSObject.getWindow(this);
        JSObject jsDoc = (JSObject) jsWin.getMember("document");
        JSObject MonteForm = (JSObject) jsDoc.eval("document.forms[0]");
        numDots = (JSObject) MonteForm.getMember("numDots");
        numInside = (JSObject) MonteForm.getMember("numIn");
        numOutside = (JSObject) MonteForm.getMember("numOut");
    }
    catch (netscape.javascript.JSException jse) { }

    drawCircle();
}
public void drawDot()
{
    int inCount = 0; outCount = 0;
    if (distance(x, y, SIZE/2, SIZE/2) < SIZE/2) {
        inCount++;
    } else {
        outCount++;
    }
}

int newIn =
    Integer.parseInt((String)numInside.getMember("value")) +
    inCount;
numInside.setMember("value", "+"+newIn);
int newOut =
    Integer.parseInt((String)numOutside.getMember("value")) +
    outCount;
numOutside.setMember("value", "+"+newOut);
Example

```html
<html>
<!-- COMP519 Monte8a.html 2015.10.20 -->
<head>
  <title>Monte Carlo Darts Page</title>
  <style type="text/css">
    table { margin-left: auto; margin-right: auto; }
    body { background-color: gray; }
  </style>
</head>

<body>
  <form id="MonteForm">
    <table>
      <tr><td>
        <object codetype="application/java"
          codebase="http://www.csc.liv.ac.uk/~martin/teaching/comp519/Java/
          classid="java:Monte8a.class" name="MonteApplet"
          height="300" width="300">
          <param name="mayscript" value="true">
            You must use a Java-enabled browser to view this applet.
          </param>
        </object>
      </td>
      <td><input type="button" value="Generate"
        onclick="document.MonteApplet.drawDots();">"</td>
    </table>
  </form>
</body>
</html>
```
View page

(The behavior of the applet generally looks the same as before, but the internals of how it operates is different. It will also work quicker than the previous version.)
Related Topics

JAR files

- For applets that are comprised of multiple classes, can bundle all necessary files into a Java Archive (JAR) file.
- Uses the popular ZIP file format.
- Download using ARCHIVES attribute, automatically unzipped by browser.

JavaBeans

- Components (e.g., buttons, menus) that can be packaged and reused.
- Requires special tools for compiling and packaging (e.g., BDK).

Interested readers can find more information on these topics online.