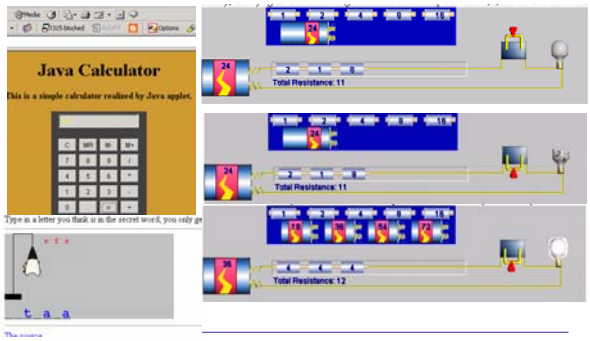
Applets

Ian Ruthven
ir@cis.strath.ac.uk

Introduction

- Applets
  - Java programs embedded in web pages
  - when page is opened compiled Java code is transferred to client and run there
    - client-side programs
  - we are going to look at
    - structure and coding of applets
    - <APPLET> tag

Example applets



Basic anatomy of a servlet

```
import java.awt.*;
import javax.swing.*;

public class HelloWorld extends JApplet
{
    ……
}
```

Import graphics functionality

Applet class name

Applet functionality

- applet is compiled as normal Java code
  - –javac HelloWorld.java
- subclass of Japplet if we use Swing
  - –subclass of Applet if we use awt

Lifecyle of an applet

- init()
  - initialises the applet each time the page is loaded
  - code that would normally be put in a constructor
  - can also do things like start downloading any files you need
  - often all code goes here
- start ()
  - often code that performs function of applet
  - code dealing with threads or time (e.g. playing sounds)
    - called when applet becomes visible
- stop()
  - stops the applet's execution
    - Useful if application is resource-heavy
  - browser is quit or user leaves the page
- destroy ()
  - performs a clean-up for unloading before browsing quitting
  - not always run (browser may crash)
  - usually do not override this method

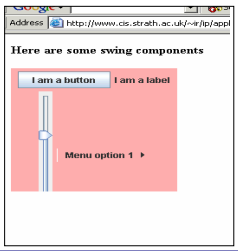
Example

- e.g. for an applet that plays a video file
- init()
  - draw the controls and start loading the video file
- start()
  - wait until the file was loaded, and then start playing it
- stop()
  - pause the video, but not rewind it.
  - if start() were called again, the video picks up where it left off it would not start over from the beginning
- destroy() then init()
  - video starts over from the beginning
- note: no main method

Drawing controls

- applets can be formed from Java Swing graphics components, in particular
  - buttons, labels, menus, sliders, tables, etc
  - displays on a content pane

```
import java.awt.*;
import javax.swing.*;
public class Buttons extends JApplet {
  public void init() {
   Container content = getContentPane();
    content.setBackground(Color.pink);
    content.setLayout(new FlowLayout());
    content.add(new JButton("I am a button"));
    content.add(new JLabel("I am a label"));
    content.add(new JSlider(JSlider.VERTICAL,1,10,5));
    content.add(new JMenu("Menu option 1"));
  }}
```



Event handling

- we can tell the applet to do things when
- the user
  - clicks a button
  - presses return while typing in a text field
  - chooses a menu item
  - closes a frame (main window)
  - presses a mouse button while the cursor is over a component
  - moves the mouse over a component
- or when
  - component becomes visible
  - component gets the keyboard focus
  - table or list selection changes

Event handling

- each of these events are handled by *listeners*
  - objects that wait for events and then react to them
- *e.g.*
  - user clicks a button, presses return while typing in a text field, or chooses a menu item
    - handled by ActionListener
  - user presses a mouse button while the cursor is over a component
    - handled by MouseListener
  - user moves the mouse over a component
    - handled by MouseMotionListener
- lots of different types of listeners
  - http://java.sun.com/docs/books/tutorial/uiswing/events/api.html

Listeners

- Listeners react to different events
  - e.g. mouseListener
    - mouseClicked(MouseEvent), mouseEntered(MouseEvent), mouseExited(MouseEvent), mousePressed(MouseEvent), mouseReleased(MouseEvent)
  - menuKeyListener
    - menuKeyPressed(MenuKeyEvent), menuKeyReleased(MenuKeyEvent), menuKeyTyped(MenuKeyEvent)
  - actionListener
    - actionPerformed(ActionEvent)
- the listeners are attached to the interface objects

Example

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class Listener1 extends JApplet implements ActionListener {
   public void init() {
      JButton button = new JButton("Click Me");
      getContentPane().add(button, BorderLayout.CENTER);
      button.addActionListener(this);
   }
   public void actionPerformed(ActionEvent e) {
      Toolkit.getDefaultToolkit().beep();
   }}
```

import listener and event classes

tell applet what type of events to listen for

add Listener to object

tell applet what to do

Example

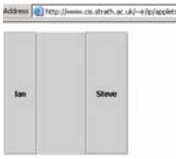
- Applet can use different types of Listener
  - implements MouseListener, ActionListener
- can attach same listener to multiple objects

….same as before

```
public void init() {

    String string1 = "Steve";
    String string2 = "Ian";
   JButton button1 = new JButton(string1);
   JButton button2 = new JButton(string2);
   getContentPane().add(button1, BorderLayout.EAST);
   getContentPane().add(button2, BorderLayout.WEST);
   button1.addActionListener(this);
   button2.addActionListener(this);
}

public void actionPerformed(ActionEvent e) {
   Toolkit.getDefaultToolkit().beep();   }}
```

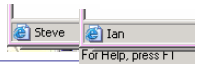


Examples

- sometimes we want response to vary
  - e.g. to do something different when Steve's button is pressed or when Ian's button is pressed
    - actionCommand(String) is a string that differentiates between actions
      - essentially *names* the action
      - often state information (what to do)
      - especially useful for menus
    - set using *component*.setActionCommand(String commandName)
    - retrieved through event.getActionCommand()
    - alternatively can use event.getSource()
      - Object source = event.getSource();

Examples

```
public void init() {
   String string1 = "Steve";
   String string2 = "Ian";
      JButton button1 = new JButton(string1);
   button1.setActionCommand(string1);
      JButton button2 = new JButton(string2);
   button2.setActionCommand(string2);
      getContentPane().add(button1, BorderLayout.EAST);
      getContentPane().add(button2, BorderLayout.WEST);
      button1.addActionListener(this);
      button2.addActionListener(this);
   }
   public void actionPerformed(ActionEvent e) {
      Toolkit.getDefaultToolkit().beep();
   showStatus(e.getActionCommand()); }}
```



Examples

```
private String string1 = "Steve";
private String string2 = "Ian";
private  JButton button1 = new JButton(string1);
private JButton button2 = new JButton(string2);

public void init() {
      getContentPane().add(button1, BorderLayout.EAST);
      getContentPane().add(button2, BorderLayout.WEST);
      button1.addActionListener(this);
      button2.addActionListener(this);
   }
   public void actionPerformed(ActionEvent e) {
      Toolkit.getDefaultToolkit().beep();
      if (e.getSource() == button1) {
                  showStatus("Steve was here");}
            else if (e.getSource() == button2){
                  showStatus("Ian was here");
            };
```


Examples

```
public void actionPerformed(ActionEvent e) {
   if (e.getSource() == sumButton) {
               total = sum(total, newValue)}
         else if (e.getSource() == minusButton){
               total = subtract(total, newValue)}
         };
```


Applets functionality

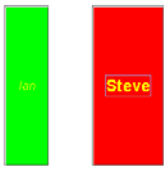
- Applets let us
  - be notified by the browser of milestones
    - Events
  - display short status strings
    - showStatus(String)
  - make the browser display a document
  - play sounds
  - make pretty interfaces
  - do animations by refreshing applet with new image

Applet actions

- public void showDocument(java.net.URL *url*)
- public void showDocument(java.net.URL *url*, String *targetWindow*)
  - targetWindow can have one of several options, e.g.
    - "_blank" open in a new, nameless window.
    - "*windowName*" display the document in a window named *windowName*. This window is created if necessary.
    - "_self" Display the document in the window and frame that contain the applet.
- getAudioClip(URL)
  - return an object that implements the AudioClip interface.
- play(URL)
  - play the AudioClip corresponding to the specified URL

Painting stuff

```
public void init() {
        Container content = getContentPane();
        content.setBackground(Color.white);
        String string1 = "Steve";
        String string2 = "Ian";
        JButton button1 = new JButton(string1);
        button1.setBackground(Color.red);
        button1.setForeground(Color.yellow);
        Font font = new Font("Arial", Font.BOLD,20);
        button1.setFont(font);
        JButton button2 = new JButton(string2);
        button2.setBackground(Color.green);
        button2.setForeground(Color.yellow);
        font = new Font("Arial", Font.ITALIC,15);
        button2.setFont(font);
    getContentPane().add(button1, BorderLayout.EAST);
    getContentPane().add(button2, BorderLayout.WEST);
}
```



<APPLET> Tag

- Applets are embedded in HTML using <APPLET> </APPLET> tag

```
<b>Here is some text<b>
<p>
<p>
<APPLET CODE="Buttons.class"> </APPLET>
</BODY>
</HTML>
```


Applet tag

< APPLET
[CODEBASE = *codebaseURL*] – where code for applet is located
assumes same page as html page as default
CODE = *appletFile* – name of applet (i.e. Java class file)
[ALT = *alternateText*] – alternate text
appears if applet cannot be run
[NAME = *appletInstanceName*] – give the applet a name
WIDTH = *pixels* – width of applet
HEIGHT = *pixels* – height of applet
[ALIGN = *alignment*] – alignment (same as IMG left, right, top, etc)
[VSPACE = *pixels*] – number of pixels above/below applet
[HSPACE = *pixels*] > - number of pixels either side of applet
> - end of <APPLET
[< PARAM NAME = *appletParameter1* VALUE = *value* >] - inputs
[*alternateHTML*] – what to display instead of applet (if client cannot understand the applet tag)
</APPLET>

Examples

- < APPLET CODE = "*SimpleApplet*" WIDTH = 100 HEIGHT = 100 > </APPLET>
  - simplest applet
- < APPLET CODE = "SimpleAppletInDifferentPlace" CODEBASE = *myCode* WIDTH = 100 HEIGHT = 100> </APPLET>
  - myCode is directory containing code for applet
  - absolute, e.g. "http://www.cis.strath.ac.uk/~ir/ip/applets"
  - relative to html page, e.g. "/applets"
- < APPLET CODE = "SimpleWithDefault" NAME = "FirstApplet" CODEBASE = *myCode WIDTH = 100 HEIGHT = 100*> </APPLET>
  - FirstApplet is name of applet not name of class
- < APPLET CODE = "SimpleWithDefault" CODEBASE = *myCode* WIDTH = 100 HEIGHT = 100> "Get a better browser" </APPLET>
  - "Get a better browser" displayed if applet cannot be recognised

Parameters

- can pass input values to applet using PARAM tag
  - < PARAM NAME = "name" VALUE = "value" >
  - e.g. <PARAM NAME = "message" Value = "Wotcha!">
- go between <applet..></applet> tags
  - < APPLET CODE ... >
  - < PARAM NAME = *appletParameter1* VALUE = *value* >
  - </APPLET>
- in applet code

```
public class Parameter extends JApplet
{ public void init() {
Container content = getContentPane();
String inputFromPage = getParameter("message");
    content.add(new JButton(inputFromPage)); } }
```

- so can reuse applets with different inputs

Applets and client

- Applets can be signed or unsigned
  - note this changes depending on browser and politics between Microsoft and everyone else
- most applets are unsigned
  - limits what applet can do, e.g. start up applications, create local files
- sometimes we want this behaviour
  - applet must be *signed* with a signature
    - and the correct signature
  - a sequence of characters embedded in the applet's code telling who the applet came from
    - http://java.sun.com/developer/technicalArticles/Security/Signed/
    - but says nothing about quality of applet
  - AND the user must grant the applet the requested permissions (using browser preferences)

Signed applets

- signed applets can have different security levels
  - decided by author of applet
  - client decides on what to accept
    - but may be decided by sys admin
  - high, medium, low security, untrusted
    - **low** security can read/write/delete local files
    - **medium** will warn before doing this
    - **highly** secure cannot do this
    - **untrusted** will not run
- unsigned applets have untrusted, high, or medium security (generally untrusted) and user decides on which level to employ

25

What applets cannot do

- write data to any of the server machine's disks
  - only to client
  - cannot delete files either
- (usually) read any data from the server's disks
  - depends on environment but even then only with permission
- make a network connection to a host
  - other than the one from which it was downloaded
- introduce a virus or trojan horse into the host system
- session tracking
  - can access some data from client but no real support

26

Summary

- Applets
  - client side programs
  - simple to construct and embed in HTML
  - still popular
  - like javascript they run client-side
    - but faster to run than javascript and graphical
    - although slower to download
  - examples of applets in action
    - http://java.sun.com/developer/codesamples/applets.html

27