Java Applets
Applets

- An applet is a Panel that allows interaction with a Java program
- Typically embedded in a Web page and can be run from a browser
- You need special HTML in the Web page to tell the browser about the applet
- For security reasons applets run in a sandbox
  - Sandbox is a
    - Byte-code verifier
    - Class loader
    - Security manager
  - Only the correct classes are loaded
  - The classes are in the correct format
  - Untrusted classes
    - Will not execute dangerous instructions
    - Are not allowed to access protected system resources
Applet Support

- Java 1.4 and above are supported from the most modern browsers if these browsers have the appropriate plug-in
- Basic browsers that support applets
  - Internet Explorer
  - Netscape Navigator (sometimes)
- However, the best support isn't a browser, but the standalone program appletviewer
- In general you should try to write applets that can be run with any browser
Notion of Applets in Java

- You can write an applet by extending the class `Applet`.
- `Applet` class
  - Contains code that works with a browser to create a display window.
  - Is just a class like any other
    - You can even use it in applications if you want.
- When you write an applet you are only writing part of a program.
- The browser supplies the `main` method.
- **NOTE**: If you use Swing components in your applet you must use the `JApplet` class.
  - `JApplet` extends the class `Applet`.
The genealogy of the Applet class

- **Applet** inherits **awt Component** class and **awt Container** class
- **JApplet** inherits from **Applet** class
The Simplest Possible Applet

TrivialApplet.java

```java
import java.applet.Applet;
public class TrivialApplet extends Applet {
}
```

TrivialApplet.html

```html
<applet
code="TrivialApplet.class"
width=150 height=100>
</applet>
```
import java.awt.*;
import java.applet.Applet;

public class HelloWorld extends Applet {
    public void paint(Graphics g) {
        g.drawString("Hello World!", 30, 30);
    }
}
Applet methods

- Basic methods
  - public void init ()
  - public void start ()
  - public void stop ()
  - public void destroy ()

- Other Supplementary methods
  - public void showStatus(String)
  - public String getParameter(String)
How a Java Applet works?

- You write an applet by extending the class `Applet`.
- `Applet` class defines methods as:
  - `init( )`
  - `start( )`
  - `stop( )`
  - `destroy( )`
  - and some others...
- These methods **do not do anything**
  - They are **stubs**
- You make the applet do something by **overriding these methods**
- You don’t need to override all these methods
  - Just the ones you care about
Method `init()`

- This is the first of your methods to be executed
- It is automatically called by the system when the JVM launches the applet for the first time
- It is only executed once
- It is the best place to
  - Initialize variables
  - Define the GUI Components
    - E.g. buttons, text fields, scrollbars, etc.
  - Lay the components out
  - Add listeners to these components
  - Pick up any HTML parameters
- Almost every applet you ever write will have an `init()` method
Method start( )

- Not usually needed
- It is automatically called after the JVM calls the init( ) method
- Also called whenever a user returns to the HTML page containing the applet after having gone to other pages
  - i.e. each time the page is loaded and restarted
- Can be called repeatedly
  - Common place to restart a thread
    - E.g. resuming an animation
- Used mostly in conjunction with stop( )
Method stop( )

- Not usually needed
- It is **automatically called** when the user moves off the page on which the applet sits
- Can be **called repeatedly** in the same applet
- Called just before **destroy( )**
- Gives a chance to **stop time-consuming activity** from slowing down the system when the user is not paying attention to the applet
- Should not be called **directly**
- Used mostly **in conjunction with start( )**
Method destroy() 

- Almost **never** needed
- Called after **stop()**
- The JVM guarantees to call this method when the browser shuts down **normally**
- Use to explicitly **release** system resources
  - E.g. threads
- System resources are usually released **automatically**
- Commonly used for reclaiming non-memory-dependent resources
Order of Methods’ Calls

- `init()` and `destroy()` are only called once each.
- `start()` and `stop()` are called whenever the browser enters and leaves the page.
- `do some work` is code called by the listeners that may exist in the applet.
Other Useful Applet Methods

- `System.out.println(String)`
  - Works from `appletviewer`, not from browsers
  - Automatically opens an output window

- `showStatus(String)`
  - Displays the String in the applet’s status line
  - Each call overwrites the previous call
  - You have to allow time to read the line!
Structure of an HTML page

- Most HTML tags are containers
  - Not Java Containers !!!!
- A container is `<tag>` to `</tag>`
Invocation of Applets in HTML Code

```html
<html>
<head>
  <title>Hi World Applet</title>
</head>
<body>
  <applet code="HiWorld.class"
         width=300 height=200>
    <param name="arraysize" value="10">
  </applet>
</body>
</html>
```

Not a container
Method `getParameter(String)`

- This method is called for the retrieval of the value of a parameter with specific name which is set inside the HTML code of the applet
  - This name is the only argument of the method

- E.g. let the HTML code for the applet

  ```html
  <applet code="HiWorld.class" width=300 height=200>
    <param name="arraysize" value="10">
  </applet>
  ```

- A possible method call could be

  ```java
  String s = this.getParameter("arraysize");
  try { size = Integer.parseInt(s) } catch (NumberFormatException e) {...}
  ```
Class and attributes’ declarations

```java
import java.awt.Graphics;  // import Graphics class
import javax.swing.*;  // import swing package

public class AdditionApplet extends JApplet {

    // sum of the values entered by the user
    double sum;
```
An Applet that adds two floating-point numbers

- Method `init()`

```java
public void init() {
    String firstNumber, secondNumber;
    double number1, number2;

    // read in first number from user
    firstNumber = JOptionPane.showInputDialog(  // Enter first floating-point value
        "Enter first floating-point value" );

    // read in second number from user
    secondNumber = JOptionPane.showInputDialog(  // Enter second floating-point value
        "Enter second floating-point value" );
```
An Applet that adds two floating-point numbers

- Method `init( )` cont.(1)

```java
// convert numbers from type String to type double
number1 = Double.parseDouble( firstNumber );
number2 = Double.parseDouble( secondNumber );

// add the numbers
sum = number1 + number2;
}
```
An Applet that adds two floating-point numbers

- Method `paint(Graphics)`

```java
public void paint( Graphics g ){
    // draw the results with g.drawString
    g.drawRect( 15, 10, 270, 20 );
    g.drawString( "The sum is " + sum, 25, 25 );
}
``` //end of paint

} //end of AdditionApplet class

- HTML source for the applet

```html
<html>
<applet code=AdditionApplet.class width=300 height=50>
</applet>
</html>
```
An Applet that adds two floating-point numbers

- Output

```
Applet Viewer: AdditionApplet.class

Applet

Applet loaded.

Applet Viewer: AdditionApplet.class

Applet

Applet loaded.

Input
Enter first floating-point value
45.5
OK  Cancel

Warning: Applet Window

Input
Enter second floating-point value
72.37
OK  Cancel

Warning: Applet Window

```

The sum is 117.87

Applet started.
A Digital Clock Applet

Class and attributes’ declarations

```java
import java.awt.*;
import java.util.Calendar;
import java.applet.Applet;

public class DigitalClock extends Applet implements Runnable {
    protected Thread clockThread;
    protected Font font;
    protected Color color;
}
```
A Digital Clock Applet

- Initialization of fields in method `init()`

```java
public void init() {
    clockThread = null;
    font = new Font("Monospaced", Font.BOLD, 48);
    color = Color.green;
}
```

- Method `start()`

```java
public void start() {
    if (clockThread == null) {
        clockThread = new Thread(this);
        clockThread.start();
    }
}
```

calls the `run()` method
A Digital Clock Applet

- Method `stop()`

```java
public void stop() {
    clockThread = null;
} //end of stop
```

- Method `run()` that runs the `clockThread`

```java
public void run() {
    while (Thread.currentThread() == clockThread) {
        repaint();
        try {
            Thread.currentThread().sleep(1000);
        } catch (InterruptedException e) {
        }
    }
} //end of run
```

- `sleep()` must be invoked inside the `try` block

- `paint(Graphics)` method calls the `repaint()` method
A Digital Clock Applet

- Method `paint(Graphics)`

```java
public void paint(Graphics g) {
    Calendar calendar = Calendar.getInstance();
    int hour = calendar.get(Calendar.HOUR_OF_DAY);
    int minute = calendar.get(Calendar.MINUTE);
    int second = calendar.get(Calendar.SECOND);
    g.setFont(font);
    g.setColor(color);
    g.drawString(hour +
            "" :" + minute / 10 + minute % 10 + 
            ":" + second / 10 + second % 10,
            10, 60);
} //end of paint
} //end of DigitalClock class
```
A Digital Clock Applet

- The HTML source for the applet

```html
<html>
  <head>
    <title>Digital Clock Applet</title>
  </head>
  <body bgcolor=white>
    <h1>The Digital Clock Applet</h1>
    <applet code=DigitalClock.class width=250 height=80>
    </applet>
  </body>
</html>
```
A Digital Clock Applet

- Output

![Digital Clock Applet Output]

14:18:01

Applet started.
A Scrolling Banner Applet

- Class and attributes’ declarations

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

public class ScrollingBanner extends Applet implements Runnable {

    protected Thread bannerThread;
    protected String text;
    protected Font font;
    protected int x, y;
    protected int delay;
    protected int offset;
    protected Dimension d;
```
A Scrolling Banner Applet

- Initialization of fields in method `init`:

```java
public void init() {
    font = new Font("Sans-serif", Font.BOLD, 24);
    delay = 100;
    offset = 1;
    // get parameter "text"
    String att = getParameter("text");
    if (att != null) {
        text = att;
    } else {
        text = "Scrolling banner.";
    }
    // set initial position of the text
    d = getSize();
    x = d.width;
    y = font.getSize();
} //end of init
```
A Scrolling Banner Applet

- **Method `start()`**

```java
public void start() {
    bannerThread = new Thread(this);
    bannerThread.start();
}
```

- **Method `stop()`**

```java
public void stop() {
    bannerThread = null;
}
```
A Scrolling Banner Applet

- Method `run()`

```java
public void run() {
    while (Thread.currentThread() == bannerThread) {
        try {
            Thread.currentThread().sleep(delay);
        }
        catch (InterruptedException e) {}
        repaint();
    }
} //end of run
```
public void paint(Graphics g) {
    // get the font metrics to determine the length of the text
    g.setFont(font);
    FontMetrics fm = g.getFontMetrics();
    int length = fm.stringWidth(text);
    // adjust the position of text from the previous frame
    x -= offset;
    // if the text is completely off to the left end
    // move the position back to the right end
    if (x < -length)
        x = d.width;
    // set the pen color and draw the background
    g.setColor(Color.black);
    g.fillRect(0,0,d.width,d.height);
    // set the pen color, then draw the text
    g.setColor(Color.green);
    g.drawString(text, x, y);
} //end of paint   } // end of ScrollingBanner class
A Scrolling Banner Applet

- The HTML source for the applet

```html
<html>
  <head>
    <title>Scrolling Banner Applet</title>
  </head>
  <body bgcolor=white>
    <h1>The Scrolling Banner</h1>
    <applet code=ScrollingBanner.class width=300 height=50>
      <param name="text" value="Java Rules!">
    </applet>
  </body>
</html>
```
A Scrolling Banner Applet

Output

Java Rules!

Java Rules!

Java Rules!
How to Avoid Flickering?

- In the previous applet the window flickers consecutively.
- Flickering is caused by `repaint()`
  - `repaint()` calls the `update(Graphics)` method.
  - The default `update(Graphics)` method does the following:
    - Paints the whole area with the background color.
    - Sets the foreground color.
    - Calls the `paint(Graphics)` method.
  - The `update(Graphics)` method is also called by the system to update windows.
- Solution
  - Override the `update(Graphics)` method.
  - Use an off-screen image.
Class and attributes’ declarations

```java
import java.awt.*;

public class ScrollingBanner2 extends ScrollingBanner {

    // The off-screen image
    protected Image image;

    // The off-screen graphics
    protected Graphics offscreen;
```
An Extended Scrolling Banner (Flickering prevention)

- The overridden method `update(Graphics)`

```java
public void update(Graphics g) {
    // create the offscreen image if it is the first time
    if (image == null) {
        image = createImage(d.width, d.height);
        offscreen = image.getGraphics();
    }
    // draw the current frame into the off-screen image
    // using the paint method of the superclass
    super.paint(offscreen);
    // copy the off-screen image to the screen
    g.drawImage(image, 0, 0, this);
} // end of update
```
An Extended Scrolling Banner (Flickering prevention)

- The overridden method `paint(Graphics)`

```java
public void paint(Graphics g) {
    update(g);
} // end of paint
} // end of ScrollingBanner2 class
```
A Bouncing Ball Applet

- Class and attributes’ declarations

```java
import java.awt.*;
import java.applet.Applet;
public class BouncingBall extends Applet
    implements Runnable {
    protected Color color;
    protected int radius;
    protected int x, y;
    protected int dx, dy;
    protected Image image;
    protected Graphics offscreen;
    protected Dimension d;
    protected Thread bouncingThread;
    protected int delay;
```
A Bouncing Ball Applet

- Initialization of fields in method `init()`

```java
public void init() {
    color = Color.green;
    radius = 20;
    dx = -2;
    dy = -4;
    delay = 100;
    d = getSize();
    x = d.width * 2 / 3;
    y = d.height - radius;
}
```

//end of init
A Bouncing Ball Applet

● Method `start()`

```java
public void start() {
    bouncingThread = new Thread(this);
    bouncingThread.start();
}
```

//end of start

● Method `stop()`

```java
public void stop() {
    bouncingThread = null;
}
```

//end of stop
A Bouncing Ball Applet

Method run( )

```java
public void run() {
    while (Thread.currentThread() == bouncingThread) {
        try {
            Thread.currentThread().sleep(delay);
        } catch (InterruptedException e) {}
        repaint();
    }
} //end of run
```
A Bouncing Ball Applet

- Method `update(Graphics)`

```java
public void update(Graphics g) {
    // create the off-screen image buffer
    // if it is invoked the first time
    if (image == null) {
        image = createImage(d.width, d.height);
        offscreen = image.getGraphics();
    }
    // draw the background
    offscreen.setColor(Color.white);
    offscreen.fillRect(0, 0, d.width, d.height);
}
```
Method `update(Graphics)`

```java
// adjust the position of the ball
// reverse the direction if it touches
// any of the four sides
if (x < radius || x > d.width - radius) {
    dx = -dx;
}
if (y < radius || y > d.height - radius) {
    dy = -dy;
}
x += dx;
y += dy;
```
A Bouncing Ball Applet cont.

- **Method `update(Graphics)`**

```java
// draw the ball
offscreen.setColor(color);
offscreen.fillOval(x - radius, y - radius,
                    radius * 2, radius * 2);
// copy the off-screen image to the screen
g.drawImage(image, 0, 0, this);
```

- **Method `paint(Graphics)`**

```java
public void paint(Graphics g) {
    update(g);
} // end of paint
```
A Bouncing Ball Applet cont.

- The HTML source for the applet

```html
<html>
    <head>
        <title>Bouncing Ball Applet</title>
    </head>
    <body bgcolor=white>
        <h1>The Bouncing Ball</h1>
        <applet code=BouncingBall.class width=300 height=300>
        </applet>
    </body>
</html>
```
A Bouncing Ball Applet

- Output

![Applet Viewer: BouncingBall.class](image1)

- Applet started.

![Applet Viewer: BouncingBall.class](image2)

- Applet started.
Remote applets may or **may not** be trusted

**Malicious** applets can cause

- Denial of Service
  - Deny platform use (busy threads, loop, exhaust GUI resources)
  - Kill other threads
- Invasion of Privacy
- Annoyance
  - E.g. constant sound
- Flashing display
  - Causes seizures in some users
- Steal CPU cycles
  - E.g. crack encryption
Java Security in Applets

- For that reason, applets always run with Java's security model
  - I.e. a sandbox model allocated by the web browser
- Inside this model applets cannot
  - Access (read/write/delete/create) to local file system
  - Modify other system resources
    - E.g. Configuration
  - Access the internals of web browser
What is a Sandbox?

- A byte-code verifier
  - Ensures that only legitimate Java bytecodes are executed
  - Together with the JVM, guarantees language safety at run time
- A class loader
  - Defines a local name space, which can be used to ensure that an untrusted applet cannot interfere with the running of other programs
- A security manager
  - Checks access to crucial system resources that is mediated by the JVM
  - Restricts the actions of a piece of untrusted code to the bare minimum
Java Security in Applets

- However, the sandbox model is too restricted
  - E.g. for LAN
- A solution for loosing this restriction is to use signed applets
  - An applet can be signed using digital signature
- A local site configuration can specify which signers are trusted
  - Applets signed by trusted parties are treated as trusted local codes and have full system access
JDK 1.2 Security
JDK 1.3 Security