



# Java Applets

---

# Introduction to Java and Java Applets



---

- Java **applications**
  - Run in **stand-alone** mode
  - No **additional software required** (such as a Web browser)
- Java **applets**
  - **Compiled Java class files**
  - Run within a **Web browser** (or an appletviewer)
  - Loaded from **anywhere on the Internet**
    - **security restrictions!**



# Java Basic Concepts

---

- Source Code converted to Byte code
  - Byte code -machine code of JVM (Java Virtual Machine)
  - Each real machine must have own JVM
    - Interpretation
    - JIT compilation
    - Direct Execution
  - Java Byte Code consists of
    - 1 Byte opcode
    - 1 or more operands

# Capabilities and Limitations of Applets



---

- Build full-featured graphical user interfaces (suitable for the Web)
- Communicate over the Internet to a host server (support Client-Server architecture)
- Communicate with other applets on a form
- **Environment-neutral** (any platform)
- Limitations on Java applets to ensure **client security**

# Capabilities and Limitations of Applets



---

- *Bytecode verification*
  - Forces loaded Java applets to undergo a **rigorous set of checks** in order to run on the local system
  - The **verifier** checks each bytecode before it is executed to make sure that it is not going to perform an **illegal operation**
- *Client-side precautions*
  - Most **Web browsers** preclude Java applets from doing **file access** or communicating with any **computer on the Internet** other than the computer that the applet was loaded from
  - Enforced by the client Web browser (or other applet loader) but done by a part of the Java runtime engine known as the **class loader**



# First Java Applet

---

```
import java.awt.*; //Contains all of the classes for creating user interfaces
                    //and for painting graphics and images
import java.applet.Applet;
public class HelloFromVenus extends Applet {

    public void paint(Graphics g) {
        Dimension d = getSize();
        g.setColor(Color.orange);
        g.fillRect(0,0,d.width,d.height);
        g.setFont(new Font("Sans-serif",Font.BOLD,24));
        g.setColor(new Color(255, 10, 0));
        g.drawString("Hello From Venus, a Mars Colony!",
40, 25);
        g.drawImage(getImage(getCodeBase(),"venus.jpg"),
20, 60, this);
    }
}
```



# HTML Source

---

```
<html>
  <head>
    <title> Hello From Venus Applet </title>
  </head>
  <body bgcolor=black text=white>
  <h2>Here is the <em>Hello From Venus</em>
    Applet</h2>
  <center>
    <applet code="HelloFromVenus.class" width=700
      height=500>
    </applet>
  </center>
  <hr>
  <a href="HelloFromVenus.java">The source.</a>
  </body>
</html>
```



# Elements of Java Applets

---

- **Superclass:** `java.applet.Applet`
  - extend `javax.swing.JApplet` if you have swing components
    - *Swing*: Sun's set of GUI components that give much fancier screen displays than the raw AWT
- No `main()` method
- `paint()` method paints the picture
- Applet tags:  
`code`    `width`    `height`





# Compile and Run an Applet

---

To compile: `javac HelloFromVenus.java` → Generates  
`HelloFromVenus.class`

To run:

a) Use the appletviewer from JDK

```
appletviewer Venus.html
```

b) Open page from browser:

```
Venus.html
```



# Applet's Life

---

- Each applet has four major events in its lifetime:
  - Initialization --- `init()`
  - Starting --- `start()`
  - Painting --- `paint(Graphics)`
  - Stopping --- `stop()`
  - Destroying --- `destroy()`
- The methods
  - defined Applet class
    - Except for `paint()` → in class `java.awt.Container`
  - do nothing--they are stubs
  - You make the applet do something by overriding these methods



# Applet's Life

---

- When an applet begins the following sequence of methods is called
  - `init()`
    - informs applet that it has been **loaded into the system**
    - Called only once
    - an ideal place to **initialize variables** and **create UI** objects
  - `start()`
    - informs applet that it should **start its execution**
    - Right **after** `init()`
    - Each time the page is **loaded and restarted**
  - `paint(Graphics)`
- When an applet dies (or is terminated), the following sequence of method calls takes place:
  - `stop()`
    - informs applet that it should **stop its execution**
    - When a web browser **leaves the HTML** document

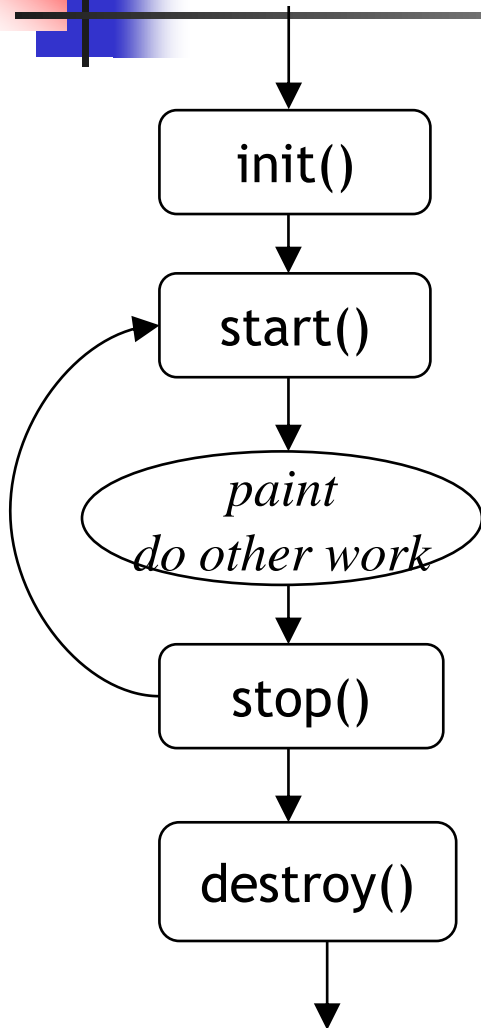


# Applet's Life

---

- `destroy()`
  - informs applet that it is being reclaimed and that it **should destroy any resources that it has allocated**
  - Use `destroy()` to explicitly **release system resources** (like threads)
    - Usually released automatically (Auto garbage collection)
  - Called only once
    - when the **environment determines** that your applet needs to be removed completely from memory
    - The `stop()` **method is always called before** `destroy()`
    - **no guarantee that this method will be completely executed**
      - The Java Virtual Machine might exit before a long `destroy` method has completed

# Methods are called in this order



- `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 your *listeners*
- `paint` is called again when the applet needs to be **repainted**



```
public void paint(Graphics g)
```

---

- Needed if you do **any drawing or painting** other than just using standard GUI Components
- Any painting you want to do should be done here, **or in a method you call from here**
- For painting done in other methods
  - *Never call `paint(Graphics)`, always call `repaint()`*
- **Life Cycle Applet via AppletViewer**
- Automatically called when
  - when the **applet begins execution**
  - the **window in which the applet is running may be overwritten** by another window and then uncovered
  - the **applet window is resized**



# Other Applet Methods

---

- `public void repaint()`
- `public void update (Graphics)`
- `public void showStatus (String)`
- `public String getParameter (String)`
- <http://download.oracle.com/javase/6/docs/api/java/applet/Applet.html>



# repaint ( )

---

- Call `repaint ( )` when you have changed something and want your changes to show up on the screen
  - after drawing commands (`drawRect (...)`, `fillRect (...)`, `drawString (...)`, etc.)
  - Outside paint
- `repaint ( )` is a *request*
  - it might not happen!
  - When you call `repaint ( )`, Java schedules a call to `update (Graphics g)`
  - ```
public void update (Graphics g) {  
    // Fills applet with background  
    // color, then  
    repaint (g);  
}
```






# Sample Graphics methods

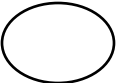
---

- A Graphics is something you can paint on


`g.drawString("Hello", 20, 20);`      **Hello**

`g.drawRect(x, y, width, height);`      

`g.fillRect(x, y, width, height);`      

`g.drawOval(x, y, width, height);`      

`g.fillOval(x, y, width, height);`      

`g.setColor(Color.red);`      

# Drawing Strings

```
g.drawString("A Sample String", x, y)
```



baseline

(x,y)

A sample string



# The `java.awt.Color` Class

---

- Instances of the `Color` class represent colors
  - `new Color(r, g, b)`
- where *r*, *g*, *b* are the values of the red, green, and blue components, respectively
- Range of 0 to 255
- Set of constants defined in `java.awt.Color`



# The `java.awt.Font` Class

---

- Fonts are specified with three attributes:
  - *font name*: `Serif Sans-serif Monospaced Dialog DialogInput TimesRoman Helvetica Courier Dialog`
  - *font style*: `PLAIN BOLD ITALIC`
    - Styles can be combined: `Font.BOLD|Font.ITALIC`
  - *font size*: a positive integer
- A font can be created as follows:
  - `new Font(name, style, size)`

# The `java.awt.Graphics` Class



---

Represent *Graphics Context*

A *graphics context* is an abstraction of various *drawing surfaces*:

- screen

- printer

- off-screen image (an image stored in memory)

Provide a rich set of graphics methods

|                              |                              |
|------------------------------|------------------------------|
| <code>drawString()</code>    | <code>drawLine()</code>      |
| <code>drawArc()</code>       | <code>fillArc()</code>       |
| <code>drawOval()</code>      | <code>fillOval()</code>      |
| <code>drawPolygon()</code>   | <code>fillPolygon()</code>   |
| <code>drawRect()</code>      | <code>fillRect()</code>      |
| <code>drawRoundRect()</code> | <code>fillRoundRect()</code> |

# The java.awt.Graphics Class (cont'd)



---

```
setColor(color)  
setFont(font)  
setPaintMode()  
setXORMode(color)  
getColor()  
getFont()  
getFontMetrics()  
getFontMetrics(font)
```

```
set the current color  
set the current font  
set the paint, or overwrite mode  
set the XOR mode  
get the current color  
get the current font  
get the font metrics of the current font  
get the font metrics for the specified font
```



`showStatus (String s)`

---

- `showStatus (String s)` displays the String in the applet's status line
  - Each call overwrites the previous call
  - You have to allow time to read the line!



# Example Applet

---

```
import java.awt.*;
import java.applet.Applet;
import javax.swing.JOptionPane;
//try it in eclipse using AppletViewer
public class LifeCycleApplet extends Applet
{
    Font theFont = new Font("Helvetica", Font.BOLD, 20);
    String Status;

    public void init(){
        Status = "Initializing!";
        showStatus("The applet is initializing!");
        JOptionPane.showMessageDialog(this, Status);
        repaint();}

    public void start(){
        Status += "--Starting!";
        showStatus("The applet is starting!");
        JOptionPane.showMessageDialog(this, Status);
        repaint();}

}
```





# Example Applet

---

```
public void stop() {
    Status += "--Stopping!";
    showStatus("The applet is stopping!");
    JOptionPane.showMessageDialog(this, Status);
    repaint();}

public void destroy() {
    Status += "--Destroyed!";
    showStatus("The applet is being destroyed!");
    JOptionPane.showMessageDialog(this, Status);
    //might cause freezing problems due to
    //unpredictability of when VM calls this method
    repaint();
}
```



# Example Applet

---

```
public void paint(Graphics g) {
    Status += "--Painting!";

    Dimension d = getSize();
    g.setColor(Color.orange);
    g.fillRect(0, 0, d.width, d.height);
    g.setFont(theFont);
    g.setColor(Color.blue);

    g.drawString("Author:" + getParameter("FName") + " " + getParameter("LName"), 50, 50);
    g.drawString("URL of the applet : " + getCodeBase(), 50, 100);
    g.drawString("URL of document : " + getDocumentBase(), 50, 150);
    g.drawString(Status, 50, 200);
    showStatus("The applet is painting!");
    // JOptionPane.showMessageDialog(this, Status); }
}
```



# HTML Tags

---

```
<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>
```



# HTML Source

---

```
<!--Clock.html-->
<html>
  <head>
    <title>Clock</title>
  </head>
  <body bgcolor=white>
  <h1>The Digital Clock Applet</h1><p>
<applet code= DigitalClock.class
width=400 height=100>
</applet>
  <p><hr>
  <a href= LifeCycleApplet.java>The source</a>
</body>
</html>
```



# The <APPLET> Tag

---

- The syntax for using the <APPLET> tag is the following:
  - `<APPLET attributes>`  
`<applet_parameter_tags>`  
`alternate_content`  
`</APPLET>`
- The APPLET attributes are standard values that all applets accept and are a standard part of HTML
- The `applet_parameter_tags` contain applet-specific parameters that are read by the applet at runtime
- This is a handy way of passing arguments to an applet to allow the applet to be more generic



# The <APPLET> Tag

---

## ■ <APPLET> Tag Attributes

- ALT-Alternate text that can be displayed by text-only browsers
- ALIGN-The ALIGN attribute designates the alignment of the applet within the browser page
- CODE-(Required) The CODE attribute is used to indicate the `.class` file that loads the applet
- CODEBASE-The CODEBASE attribute is used to indicate the location of the `.class` file that loads the applet
- HEIGHT-(Required) The HEIGHT attribute is used to set the applet's bounding rectangle height
- HSPACE-The HSPACE attribute sets the amount of horizontal space to set off around the applet
- NAME-The NAME attribute sets the symbolic name of the applet
- VSPACE-The VSPACE attribute sets the amount of vertical space to set off around the applet
- WIDTH-(Required) The WIDTH attribute is used to set the applet's box width



# The <APPLET> Tag

---

## ■ Passing Parameters to Java Applets

- Parameters are an easy way to **configure Java applets** without actually changing the source file
  - Background color based on preference (different HTML files)
- In the previous applet example, the text drawn on the screen was drawn using the blue color
  - This was "hardwired" into the applet's code
- However, just as easily, we could have passed a parameter to the applet specifying that it use the blue tag
- See next example



# The <APPLET> Tag

---

- **// Passing parameters to the applet using HTML parameters.**

```
<HTML>
<HEAD>
<TITLE>This is the LifeCycle applet!</TITLE>
</HEAD>
<BODY>
<H1>Prepare to be amazed!</H1>
<BR>
<APPLET CODE="LifeCycleApplet.class" WIDTH=600
HEIGHT=50>
<PARAM NAME=color VALUE="blue">
If you can see this, your browser does not support Java
applets
</APPLET>
</BODY>
</HTML>
```

- The only question left to be answered is this: how does the Java applet determine the value of the parameters?





# The `<APPLET>` Tag

---

- The answer is that the applet has to call the `getParameter()` method supplied by the `java.applet.Applet` parent class
- Calling `getParameter("color")` using the previous Java applet example would return a `String` value containing the text "blue"
- It is then left up to the applet to take advantage of this information and actually paint the text blue on the screen
- Here are three methods commonly used by applets:
  - `String getParameter(String name)`: Returns the value for the specified parameter string
  - `URL getCodeBase()`: Returns the URL of the applet
  - `URL getDocumentBase()`: Returns the URL of the document containing the applet