JYOTHISHMATHI INSTITUTE OF TECHNOLOGY AND SCIENCE

DEPARTMENT OF COMPUTER SCIENCE ENGINEEING

OBJECT OREINTED PROGRAMMING THROUGHT JAVA

APPLETS(UNIT-V)

K.RACHANA

ASST. PROFESSOR

CSE DEPT.

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

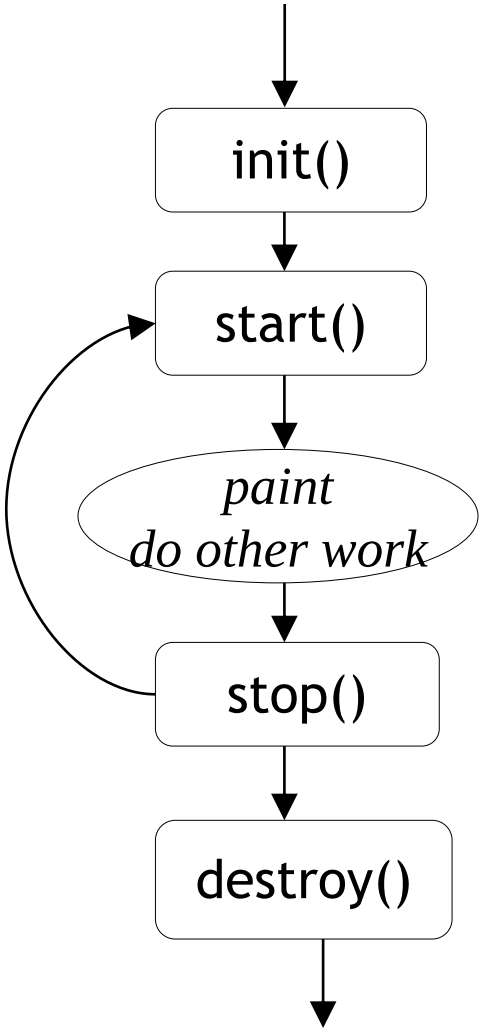
Applet's Life

- 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
    paint(g);
}
```

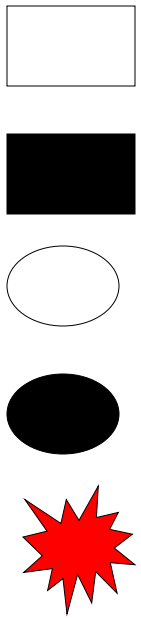
Sample Graphics methods

- A Graphics is something you can paint on

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

Hello



Drawing Strings

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

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

`drawString()`	`drawLine()`
`drawArc()`	`fillArc()`
`drawOval()`	`fillOval()`
`drawPolygon()`	`fillPolygon()`
`drawRect()`	`fillRect()`
`drawRoundRect()`	`fillRoundRect()`

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

`setColor(color)`	set the current color
`setFont(font)`	set the current font
`setPaintMode()`	set the paint, or overwrite mode
`setXORMode(color)`	set the XOR mode
`getColor()`	get the current color
`getFont()`	get the current font
`getFontMetrics()`	get the font metrics of the current font
`getFontMetrics(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();}

}
```