General Form of an Applet (cont'd)

- An applet overrides a set of methods in the class Applet to implement its functionality. These methods are used as an interface with the browser or the applet viewer.
- An applet does not need to override those methods it does not use.
- The following lists the most important methods that are usually used:

```
import java.applet.*;
import java.awt.*;
public class AppletName extends Applet
{
     public  void  init(){ .   .   . }
     public  void  start(){ .   .   . }
     public  void  stop(){ .   .   . }
     public  void  destroy(){ .   .   .}
     public  void  paint(Graphics  g ){ .   .   .}
}
```

Applet Initialization and Termination

- When an applet begins, the browser calls the following methods, in this sequence: init(), start().
- Every time the applet is redrawn, the method paint() is called.
- When an applet is terminated, the following sequence of methods is invoked: stop(), destroy().

Method	Comment
`init()`	Applets do not usually have **main** method; instead they have the **init() method** that, like **main()**, is invoked by the execution environment. *It is the first method called for any applet*. It is called *only* ***once*** during the run-time of an applet.
`start()`	Called by the execution environment when an applet should start or resume execution. It is automatically called after **init()** when an applet first begins.
`stop()`	Called to suspend execution of the applet. Once stopped, an applet is restarted when the execution environment calls **start()**.
`destroy()`	Called just before the applet is terminated. Your applet should override this method if it needs to perform any cleanup prior to its destruction.

Applet methods

public void init ()
public void start ()
public void stop ()
public void destroy ()
public void paint (Graphics)

Also:

public void repaint()
public void update (Graphics)
public void showStatus(String)
public String getParameter(String)

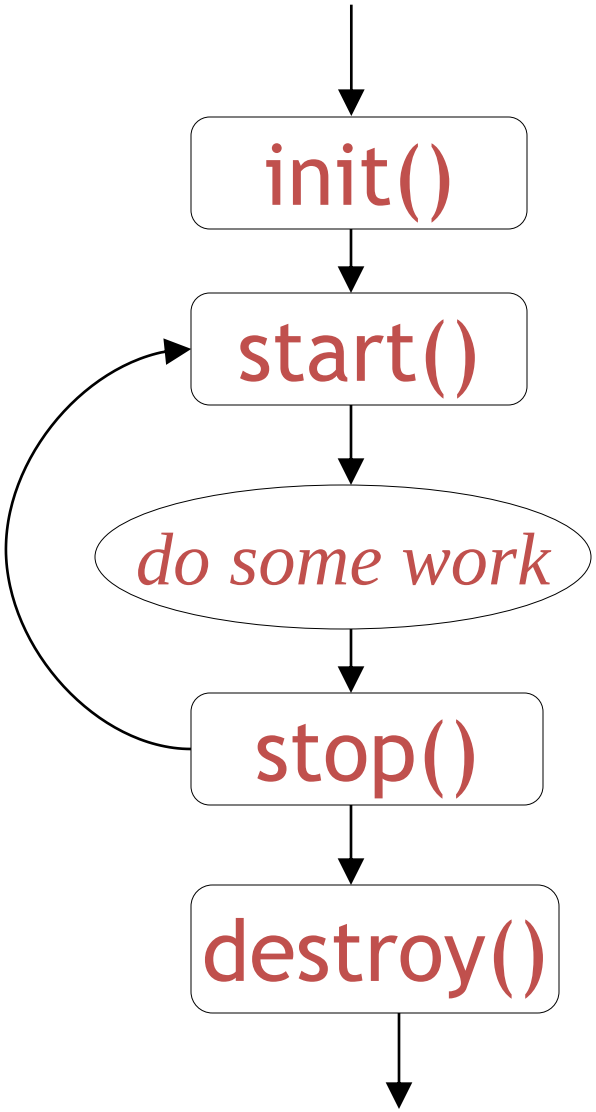
public void init ( )

- init() is the first method to execute
  - init() is an ideal place to initialize variables
  - init() is the best place to define the GUI Components (buttons, text fields, checkboxes, etc.), lay them out, and add listeners to them
  - Almost every applet you ever write will have an init( ) method

start( ), stop( ) and destroy( )

- start() and stop( ) are used when the Applet is doing time-consuming calculations that you don't want to continue when the page is not in front
- public void start() is called:
  - Right after init( )
  - Each time the page is loaded and restarted
- public void stop( ) is called:
  - When the browser leaves the page
  - Just before destroy( )
- public void destroy( ) is called after stop( )
  - Use destroy() to explicitly release system resources (like threads)
  - System resources are usually released automatically

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 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
- Painting that you do in other methods may *or may not* happen
- *Never call* paint(*Graphics*)*, call* repaint( )

repaint( )

- Call repaint( ) when you have changed something and want your changes to show up on the screen
  - You do *not* need to call repaint() when something in Java's own components (Buttons, TextFields, etc.)
  - You *do* need to call repaint() after drawing commands (drawRect(…), fillRect(…), drawString(…), etc.)
- repaint( ) is a *request*--it might not happen
- When you call repaint( ), Java schedules a call to update(Graphics ***g***)

update( )

- When you call repaint( ), Java schedules a call to update(Graphics g)
- Here's what update does:

```
public void update(Graphics g) {
    // Fills applet with background color,
then
    paint(g);
}
```

The paint() method

- The paint() method is called by the execution environment (i.e. the browser) each time the applet has to be redrawn.
- The inherited paint() method is empty. In order to draw anything on the applet, this method must be overridden.
- paint() method takes an object of class *Graphics* as an input argument, which is passed by the execution environment.

```
public  void  paint(Graphics g){
              .   .   .
}
```

- This Graphics object represents a drawing area. It has methods to draw strings and many shapes. Also, it can manipulate fonts and colors.

The Graphics Object

- A Graphics object has a coordinate system that is illustrated below:

- Anything that is drawn on the Graphics object, appears on the applet.

- Some of the drawing methods of the Graphics object are:
  - drawString()
  - drawLine()
  - drawRect()
  - drawOval()

Displaying Strings Using the Graphics Object

- To display a string on the Graphics object, the method drawString() can be used. It has the following arguments:

```
void drawString( String str, int x, int y)
```

  - str is the string to be displayed, x and y are the coordinates of the top left point of the string.

- For example, the following applet displays the string "Hello World!!" starting at the point (50,25). Its file name must be HelloApplet.java.

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

public class HelloApplet extends Applet {
     public void paint(Graphics g) {  // overriding paint() method
               g.drawString("Hello world!", 50, 25);
     }
}
```

Placing an Applet in a Web Page

- Recall that web pages are written in HTML. HTML language describes the appearance of a page using *tags*. For example, **<html>** is a tag. Another tag is **<body>**. Some tags have *a closing tag*. For example, **<html>** is closed by **</html>**.

- HTML is based on text, just like Java. You can use any editor (like Notepad or JCreator) to write HTML files. HTML files should have the extension HTML, like **(first.html).** All HTML pages should look like:

```
<html>
<body>
      The body of the html page… write whatever you like here.
</body>
</html>
```

Placing an Applet in a Web Page (cont'd)

- To place an applet in a web page, the **<applet>** tag is used in the body of an HTML page as follows:

```
<applet code="HelloApplet.class" width=600 height=100>
</applet>
```

- The parts in green are called *attributes*. The applet tag has three *mandatory* (non-optional) attributes:
  - code: the name of the class file of the applet.
  - width: the width of the applet, in pixels.
  - height: the height of the applet, in pixels.

- If the class file is not at the same folder as the HTML page, the codebase attribute is used to indicate the location of the class file *relative to* the directory that has the HTML page.

```
<applet code="HelloApplet.class" codebase="app\" width=600
height=100>
</applet>
```

Colors

- The class *Color* of java.awt package is used to define Color objects.
- All colors can be specified as a mix of three primary colors: red, green, and blue. A particular color can be specified by three integers, each between 0 and 255, or by three float values, each between 0.0 and 1.0.
- The class Color has some pre-defined colors that are commonly used.

Color	RGB Value (float)	RGB Value (integer)
Color.black	0.0F, 0.0F, 0.0F	0, 0, 0
Color.blue	0.0F, 0.0F, 1.0F	0, 0, 255
Color.cyan	0.0F, 1.0F, 1.0F	0, 255, 255
Color.gray	0.5F, 0.5F, 0.5F	128, 128, 128
Color.darkGray	0.25F, 0.25F, 0.25F	64, 64, 64
Color.lightGray	0.75F, 0.75F, 0.75F	192, 192, 192
Color.green	0.0F, 1.0F, 0.0F	0, 255, 0

Color	RGB Value (float)	RGB Value (integer)
Color.magenta	1.0F, 0.0F, 1.0F	255, 0, 255
Color.orange	1.0F, 0.8F, 0.0F	255, 200, 0
Color.pink	1.0F, 0.7F, 0.7F	255, 175, 175
Color.red	1.0F, 0.0F, 0.0F	255, 0, 0
Color.white	1.0F, 1.0F, 1.0F	255, 255, 255
Color.yellow	1.0F, 1.0F, 0.0F	255, 255, 0

Colors (cont'd)

- A Color object can be created using one of two constructors:

```
Color(int red, int green, int blue)
Color(float red, float green, float blue)
```

- For example:

```
Color c1 = new Color(255, 100, 18);
Color c2 = new Color(0.2F, 0.6F, 0.3F);
```

- By default, the Graphics object has a black foreground and a light gray background. This can be changed using the following methods (of the Graphics object):

```
void setBackground(Color newColor)
void setForeground(Color newColor)
void setColor(Color newColor)
```

Colors (cont'd)

- The following example displays some strings in different colors.
- Although it is possible to set the background and foreground colors in the paint() method, a good place to set these colors is in the init() method.

```
import java.awt.*; import java.applet.*;
public class MyApplet extends Applet {
     public void init() {
            setBackground(Color.blue);
            setForeground(Color.yellow);
     }
     public void paint(Graphics  g) {
            g.drawString("A yellow string", 50, 10);
            g.setColor(Color.red) ;
            g.drawString("A red string", 50, 50);
            g.drawString("Another red string", 50, 90);
            g.setColor(Color.magenta) ;
            g.drawString("A magenta string", 50, 130);
     }
}
```

Drawing Some Shapes

- An oval can be drawn using the method drawOval() as follows:

```
void drawOval( int x, int y, int width, int height )
```

- A rectangle can be drawn using the method drawRect() as follows:

```
void drawRect( int x, int y, int width, int height )
```

- A line linking two points can be drawn using the method drawLine() as follows:

```
void drawLine( int x1, int y1, int x2, int y2 )
```

- To draw a shape using a specific color, the method setColor() should be used before drawing the shape.

- There are no methods called drawCircle() or drawSquare().
  - How can we draw a circle or a square..???

Executing a Java Applet

- A Java applet must be compiled into bytecode before it can be used in a web page.

- When a web page containing an <applet> tag is opened, the associated bytecode is downloaded from the location specified by the CODE or CODEBASE attribute. This location can be in *the local machine* or in a *machine across the web*.

- To interpret the applet bytecode, the browser must have a *Java plug-in*.

- Also, an applet can be executed using the *applet viewer*, which comes with the JDK.

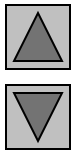
Comparing Applets with Applications

An Application	An Applet
Runs *independently*	Has to run *inside* another program, called execution environment (like *a web browser* or *an applet viewer*)
Starts by the **main()** method	Starts by the **init()** method
Doesn't have to extend any class	Has to extend java.applet.Applet class
Can work with *command-line* (like what are always doing), or using a *graphical user-interface* (GUI) {More on this in ICS-201}	Almost always works with GUI
Has an *unrestricted access* to the machine resources	Has a *restricted access* to the machine resources (cannot open files or run other programs) **{Security reasons}**

A Simple Java Applet: Drawing a String

- Now, create applets of our own
  - Take a while before we can write applets like in the demos
  - Cover many of same techniques
- Upcoming program
  - Create an applet to display

    `"Welcome to Java Programming!"`
  - Show applet and HTML file, then discuss them line by line



Outline

Java applet

```
1   // Fig. 3.6: WelcomeApplet.java
2   // A first applet in Java.
3
4   // Java core
5   import java.a                    ass Graphics
6
7   // Java exten
8   import javax.                    ass JApplet
9
10  public class WelcomeApplet extends JApplet {
11
12     // draw text on applet's background
13     public void paint( Graphics g )
14     {
15        // call inherited version of method paint
16        super.paint( g );
17
18        // draw a String at x-coordinate
19        g.drawString( "Welcome to Java P
20
21     }  // end method paint
22
23  }  // end class WelcomeApplet
```

`import` allows us to use predefined classes (allowing us to use applets and graphics, in this case).

`extends` allows us to inherit the capabilities of class **`JApplet`**.

Method **`paint`** is guaranteed to be called in all applets. Its first line must be defined as above.

Program Output

3.3 A Simple Java Applet: Drawing a String

```
1    // Fig. 3.6: WelcomeApplet.java
2    // A first applet in Java.
```

- Comments
  - Name of source code and description of applet

```
5    import java.awt.Graphics;    // import class Graphics
8    import javax.swing.JApplet;  // import class JApplet
```

- Import predefined classes grouped into packages
  - **`import`** statements tell compiler where to locate classes used
  - When you create applets, **`import`** the **`JApplet`** class (package **`javax.swing`**)
  - **`import`** the **`Graphics`** class (package **`java.awt`**) to draw graphics
    - Can draw lines, rectangles ovals, strings of characters
  - **`import`** specifies directory structure

3.3 A Simple Java Applet: Drawing a String

- Applets have at least one class definition (like applications)
  - Rarely create classes from scratch
    - Use pieces of existing class definitions
    - Inheritance - create new classes from old ones (ch. 15)

```
10   public class WelcomeApplet extends JApplet {
```

- Begins **`class`** definition for class **`WelcomeApplet`**
  - Keyword **`class`** then class name
- **`extends`** followed by class name
  - Indicates class to inherit from (**`JApplet`**)
    - **`JApplet`** : superclass (base class)
    - **`WelcomeApplet`** : subclass (derived class)
  - **`WelcomeApplet`** now has methods and data of **`JApplet`**

3.3 A Simple Java Applet: Drawing a String

```
10   public class WelcomeApplet extends JApplet {
```

- Class **`JApplet`** defined for us
  - Someone else defined "what it means to be an applet"
    - Applets require over 200 methods!
  - **`extends JApplet`**
    - Inherit methods, do not have to define them all
  - Do not need to know every detail of class **`JApplet`**

3.3 A Simple Java Applet: Drawing a String

```
10   public class WelcomeApplet extends JApplet {
```

– Class **`WelcomeApplet`** is a blueprint
  - **`appletviewer`** or browser creates an object of class **`WelcomeApplet`**
    – Keyword **`public`** required
    – File can only have one **`public`** class
    – **`public`** class name must be file name

3.3 A Simple Java Applet: Drawing a String

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

- Our class inherits method **`paint`** from **`JApplet`**
  - By default, **`paint`** has empty body
  - Override (redefine) **`paint`** in our class
- Methods **`paint`**, **`init`**, and **`start`**
  - Guaranteed to be called automatically
  - Our applet gets "free" version of these by inheriting from **`JApplet`**
    - Free versions have empty body (do nothing)
    - Every applet does not need all three methods
      - Override the ones you need
- Applet container “draws itself” by calling method **`paint`**

3.3 A Simple Java Applet: Drawing a String

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

- Method **paint**
  - Lines 13-21 are the definition of paint
  - Draws graphics on screen
  - **void** indicates **paint** returns nothing when finishes task
  - Parenthesis define parameter list - where methods receive data to perform tasks
    - Normally, data passed by programmer, as in **JOptionPane.showMessageDialog**
  - **paint** gets parameters automatically
    - **Graphics** object used by **paint**
  - Mimic **paint**'s first line

3.3 A Simple Java Applet: Drawing a String

```
16          super.paint( g );
```

- Calls version of method paint from superclass **`JApplet`**
- Should be first statement in every applet's paint method

```
19          g.drawString( "Welcome to Java Programming!", 25, 25 );
```

- Body of **`paint`**
  - Method **`drawString`** (of class **`Graphics`**)
  - Called using **`Graphics`** object **`g`** and dot operator (**`.`**)
  - Method name, then parenthesis with arguments
    - First argument: **`String`** to draw
    - Second: x coordinate (in pixels) location
    - Third: y coordinate (in pixels) location
- Java coordinate system
  - Measured in pixels (picture elements)
  - Upper left is (**`0`**,**`0`**)

3.3.1 Compiling and Executing WelcomeApplet

- Running the applet
  - Compile
    - **`javac WelcomeApplet.java`**
    - If no errors, bytecodes stored in **`WelcomeApplet.class`**
  - Create an HTML file
    - Loads the applet into **`appletviewer`** or a browser
    - Ends in **`.htm`** or **`.html`**
  - To execute an applet
    - Create an HTML file indicating which applet the browser (or **`appletviewer`**) should load and execute

3.3.1 Compiling and Executing WelcomeApplet

```
1 <html>
2 <applet code = "WelcomeLines.class" width = "300" height = "40">
3 </applet>
4 </html>
```

- Simple HTML file (**`WelcomeApplet.html`**)
  - Usually in same directory as **`.class`** file
  - Remember, **`.class`** file created after compilation
- HTML codes (tags)
  - Usually come in pairs
  - Begin with **<** and end with **>**
- Lines 1 and 4 - begin and end the HTML tags
- Line 2 - begins **`<applet>`** tag
  - Specifies code to use for applet
  - Specifies **`width`** and **`height`** of display area in pixels
- Line 3 - ends **`<applet>`** tag

3.3.1 Compiling and Executing WelcomeApplet

```
1 <html>
2 <applet code = "WelcomeLines.class" width = "300" height = "40">
3 </applet>
4 </html>
```

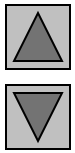
- **`appletviewer`** only understands **`<applet>`** tags
  - Ignores everything else
  - Minimal browser
- Executing the applet
  - **`appletviewer WelcomeApplet.html`**
  - Perform in directory containing **`.class`** file

3.4 Two More Simple Applets: Drawing Strings and Lines

- More applets
  - First example
    - Display two lines of text
    - Use **`drawString`** to simulate a new line with two **`drawString`** statements
  - Second example
    - Method **`g.drawLine(x1, y1, x2, y2 )`**
      - Draws a line from (**`x1`**, **`y1`**) to (**`x2`**, **`y2`**)
      - Remember that (**`0`**, **`0`**) is upper left
    - Use **`drawLine`** to draw a line beneath and above a string

```
1   // Fig. 3.8: WelcomeApplet2.java
2   // Displaying multiple strings in an applet.
3
4   // Java core packages
5   import java.awt.Graphics;    // import class Graphics
6
7   // Java extension packages
8   import javax.swing.JApplet;  // import class JApplet
9
10  public class WelcomeApplet2 extends JApplet {
11
12     // draw text on applet's background
13     public void paint( Graphics g )
14     {
15        // call inherited version of method paint
16        super.paint( g );
17
18        // draw two Strings at different locations
19        g.drawString( "Welcome to", 25, 25 );
20        g.drawString( "Java Programming!", 25, 40 );
21
22     }  // end method paint
23
24  }  // end class WelcomeApplet2
```

The two **drawString** statements simulate a newline. In fact, the concept of lines of text does not exist when drawing strings.



Outline

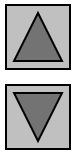
1. **import**

2. Class **WelcomeApplet2** (**extends JApplet**)

3. **paint**

3.1 **drawString**

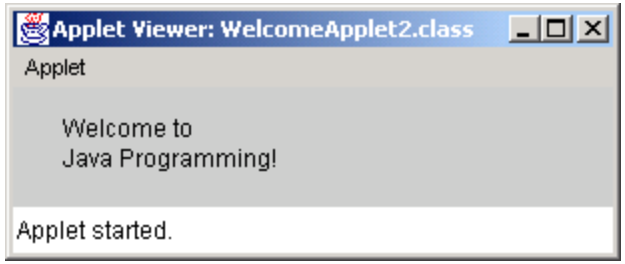
3.2 **drawString on same x coordinate, but 15 pixels down**



Outline

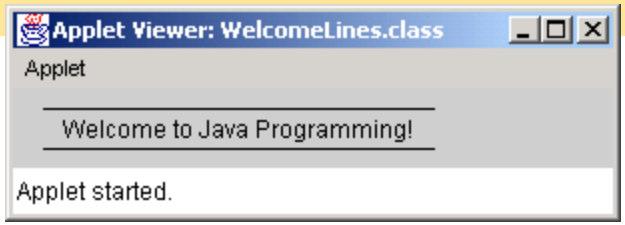
```
1    <html>
2    <applet code = "WelcomeApplet2.class" width = "300" height = "60">
3    </applet>
4    </html>
```

HTML file

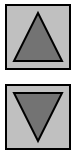


Program Output

```
1  // Fig. 3.10: WelcomeLines.java
2  // Displaying text and lines
3
4  // Java core packages
5  import java.awt.Graphics;    // import class Graphics
6
7  // Java extension packages
8  import javax.swing.JApplet;  // import class JApplet
9
10 public class WelcomeLines extends JApplet {
11
12    // draw lines and a string on applet's background
13    public void paint( Graphics g )
14    {
15       // call inherited version of method paint
16       super.paint( g );
17
18       // draw horizontal line from (15, 10) to (210, 10)
19       g.drawLine( 15, 10, 210, 10 );
20
21       // draw horizontal line from (15, 30) to (210, 30)
22       g.drawLine( 15, 30, 210, 30 );
23
24       // draw String between lines at location (25, 25)
25       g.drawString( "Welcome to Java Programming!", 25, 25 );
26
27    }  // end method paint
28
29 }  // end class WelcomeLines
```



Draw horizontal lines with **drawLine** (endpoints have same y coordinate).



Outline

WelcomeLines.java

2. Class WelcomeLines (extends JApplet)

3. paint

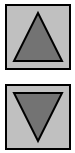
3.1 drawLine

3.2 drawLine

3.3 drawString

Program Output

```
1    <html>
2    <applet code = "WelcomeLines.class" width = "300" height = "40">
3    </applet>
4    </html>
```



Outline

HTML file