Java Applets

Introducing Applets

- Applets
  - Java programs called from within another application
  - Frequently run from a Web page
  - Display as rectangular area
  - Can respond to user-initiated events
  - Behaviors come from Java class named `JApplet`
- Steps to write an applet
  - Set up layout for applet
  - Create components and add them to applet

Applets

- An applet is a program that is 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
- You don't need to supply a main method; the browser does that
  - When you write an applet, you are writing only *part* of a program
  - You supply certain methods that the browser calls
- For security reasons, applets run in a sandbox: they have no access to the client's file system

What an applet is

- You write an applet by extending the class JApplet
- JApplet 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
  - Once you understand how applets work, you can write a program that function either as an applet or as an application—just write a main method that calls the right methods at the right time
    - Such programs have the ugly name "appletcations"

The genealogy of JApplet

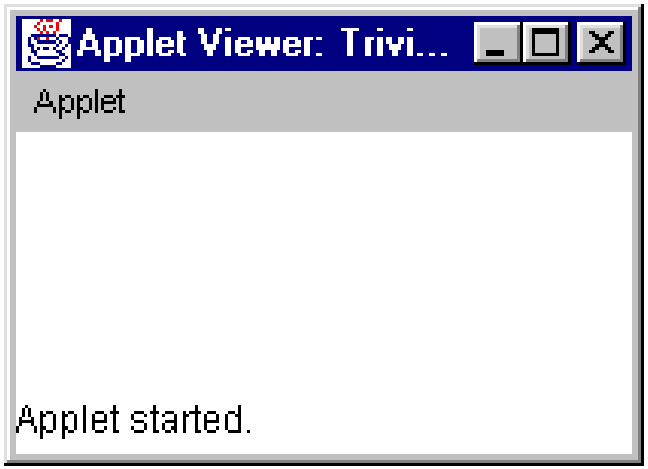
The simplest possible applet

TrivialApplet.java

```
import javax.swing.JApplet;
public class TrivialApplet extends JApplet { }
```

TrivialApplet.html

```
<applet
    code="TrivialApplet.class"
    width="150"
    height="100">
</applet>
```



The simplest reasonable applet

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

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

What are the disadvantages of applets?

- Applets can’t run any local executable programs
- Applets can’t with any host other than the originating server
- Applets can’t read/write to local computer’s file system

What are the disadvantages of applets? (Cont'd)

- Applets can't find any information about the local computer
- All java-created pop-up windows carry a warning message
- Stability depends on stability of the client's web server
- Performance directly depend on client's machine

What are the advantages of applets?

- Automatically integrated with HTML; hence, resolved virtually all installation issues.
- Can be accessed from various platforms and various java-enabled web browsers.
- Can provide dynamic, graphics capabilities and visualizations
- Implemented in Java, an easy-to-learn OO programming language

What are the advantages of applets? (Cont'd)

- Alternative to HTML GUI design
- Safe! Because of the security built into the core Java language and the applet structure, you don't have to worry about bad code causing damage to someone's system
- Can be launched as a standalone web application independent of the host web server

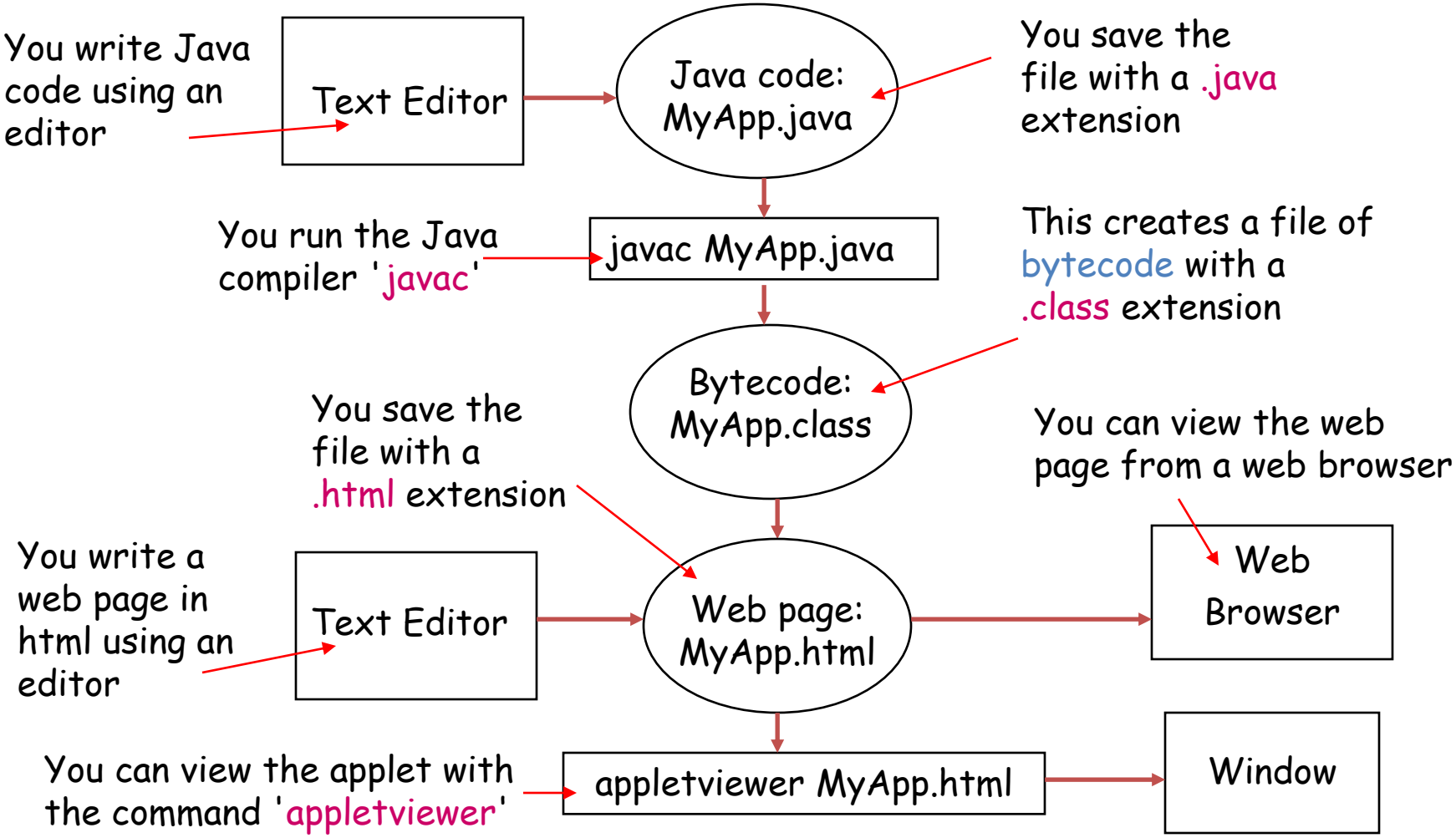
Java Applets Can not

Applets are programs designed to run as part of a Web Page (Applet = little application).

Applets are similar to normal Java Applications but have extra security features to prevent a downloaded Applet damaging your computer or transmitting information from it. For instance an Applet cannot:

- Access local files
- Delete local files
- Run another program
- Find out your name
- Connect to another host

Running a Java Applet



Creating an Applet

- Open "Notepad" (Start → Programs → Other → Notepad)
- Type this in:
- Save As "Greetings.java" (Put the " " round the name otherwise it adds .txt to the end!)

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

public class Greetings extends Applet {

    public void paint(Graphics g) {
        g.drawString("Hello World!", 50, 50);
    }
}
```

- Open a DOS Window (Start → MS-DOS Prompt)

```
G:\> javac Greetings.java
G:\>
```

- Type javac Greetings.java
- If you type dir Greetings.* you should see Greetings.java and Greetings.class

If it gives an error check you typed it in exactly right.

Creating the Web Page

In order to run an applet you have to embed it in a web page using a special <applet> tag e.g:

```
<applet code="name.class" width=www height=hhh></applet>
```

Using Notepad type in the following and save it as "Greetings.html":

Size of the applet in pixels

```
<html>
<head>
<title>Greetings Applet</title>
</head>
<body>
<applet code="Greetings.class" width=300 height=200 ></applet>
</body>
</html>
```

Running the Program

In the DOS window type appletviewer Greetings.html

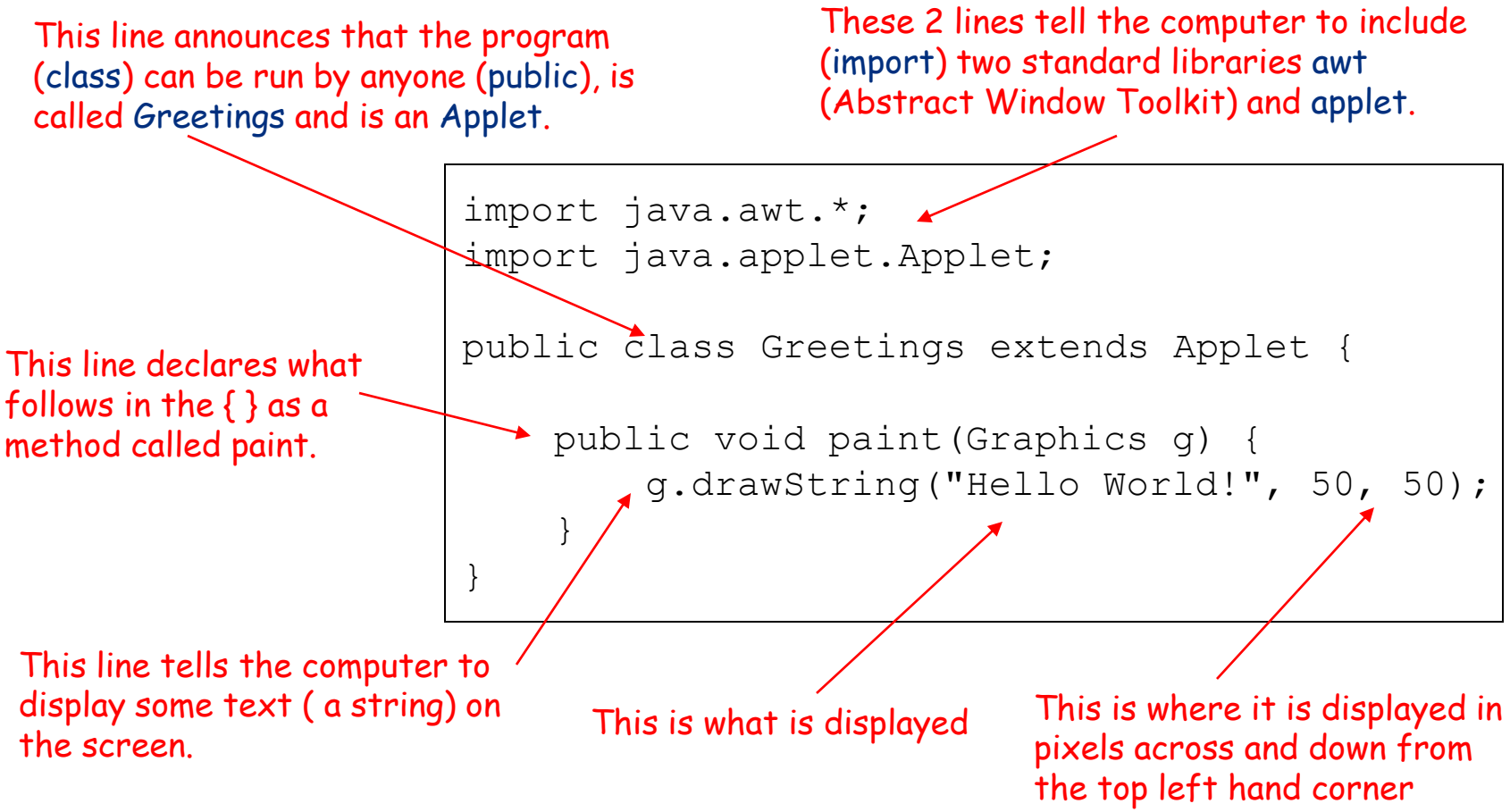
```
G:\> appletviewer Greetings.html
```

You should see something like this:

Running in a Web Browser

In Netscape go to the File menu then Open Page ...
Press Choose File...
Find your file Greetings with the Netscape symbol alongside it (Greetings.html) - click on it and press Open (or double click on it)
Back in the Open Page dialog press Open
You should see something like:

What does it mean?



General Form of an Applet

- New applets are created by extending the Applet class contained in the java.applet package.
- Also, the package java.awt is needed for the graphical interface of the applet.
- In general, an applet program would look like the following:

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

public class AppletName extends Applet
{
    .  .  .
}
```

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)

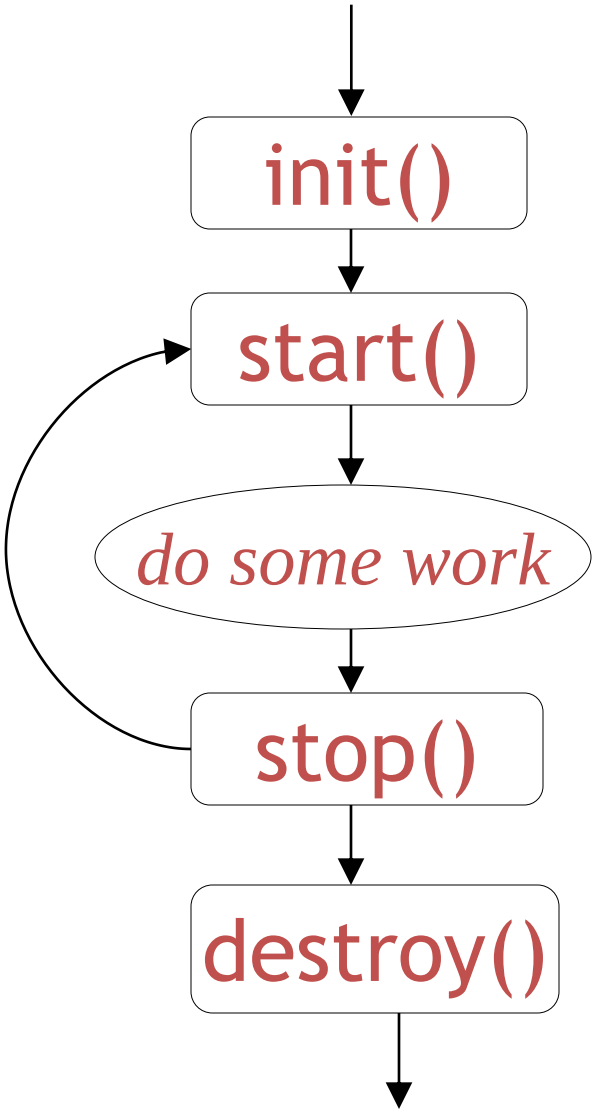
public void init ( )

- 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.

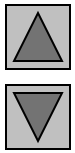
Comparing Applets with Applications

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>
```

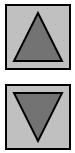
- **`appletviewer`** only understands **`<applet>`** tags
  - Ignores everything else
  - Minimal browser
- Executing the applet
  - **`appletviewer WelcomeApplet.html`**
  - Perform in directory containing **`.class`** file

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

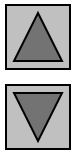
1. **import**

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

3. **paint**

3.1 **drawString**

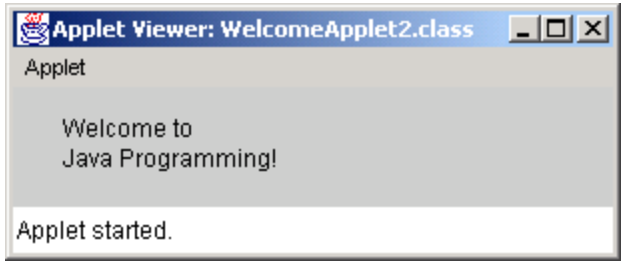
3.2 **drawString on same x coordinate, but 15 pixels down**



Outline

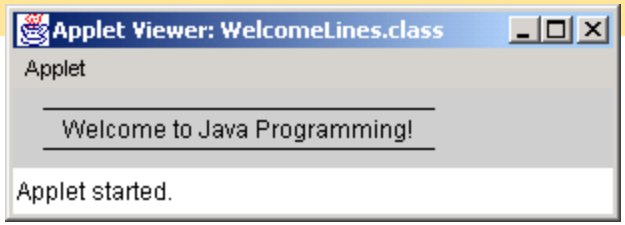
```
1    <html>
2    <applet code = "WelcomeApplet2.class" width = "300" height = "60">
3    </applet>
4    </html>
```

HTML file

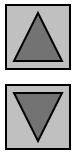


Program Output

```
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

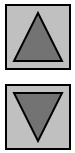
3.1 drawLine

3.2 drawLine

3.3 drawString

Program Output

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



Outline

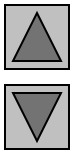
HTML file

3.4 Two More Simple Applets: Drawing Strings and Lines

- Method **`drawLine`** of class **`Graphics`**
  - Takes as arguments **`Graphics`** object and line's end points
  - X and y coordinate of first endpoint
  - X and y coordinate of second endpoint

3.5 Another Java Applet: Adding Floating-Point Numbers

- Next applet
  - Mimics application for adding two integers (Fig 2.9)
    - This time, use floating point numbers (numbers with a decimal point)
      - Using primitive data types
        - **`Double`** – double precision floating-point numbers
        - **`Float`** – single precision floating-point numbers
  - Show program, then discuss



Outline

AdditionApplet.java

1. import

2. Class AdditionApplet

3. Instance variable

showInputDialog

4.3 parseDouble

```
1   // Fig. 3.12: AdditionApplet.java
2   // Adding two floating-point numbers.
3
4   // Java core packages
5   import java.awt.Graphics;    // import class Graphics
6
7   // Java extension packages
8   import javax.swing.*;        // import package javax.swing
9
10  public class AdditionApplet extends JApplet {
11     double sum;  // sum of values entered by user
12
13     // initialize applet by obtaining values f
14     public void init()
15     {
16        String firstNumber;    // first string entered by user
17        String secondNumber;   // second string entered by user
18        double number1;        // first number to add
19        double number2;        // second number to ad
20
21        // obtain first number from user
22        firstNumber = JOptionPane.showInputDialog(
23           "Enter first floating-point value" );
24
25        // obtain second number from user
26        secondNumber = JOptionPane.showInputDialog(
27           "Enter second floating-point value" );
28
29        // convert numbers from type String to type double
30        number1 = Double.parseDouble( firstNumber );
31        number2 = Double.parseDouble( secondNumber );
32
```

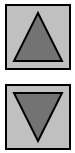
***** allows any class in the the package to be used.

Instance variable **sum** may be used anywhere in the class, even in other methods.

Data type **double** can store floating point numbers.

```
33          // add numbers
34          sum = number1 + number2;
35       }
36
37       // draw results in a rectangle on applet's background
38       public void paint( Graphics g )
39       {
40          // call inherited version of method paint
41          super.paint( g );
42
43          // draw rectangle starting from (15, 10) that is 270
44          // pixels wide and 20 pixels tall
45          g.drawRect( 15, 10, 270, 20 );
46
47          // draw results as a String at (25, 25)
48          g.drawString( "The sum is " + sum, 25, 25 );
49
50       }  // end method paint
51
52    }  // end class AdditionApplet
```

`drawRect` takes the upper left coordinate, width, and height of the rectangle to draw.



Outline

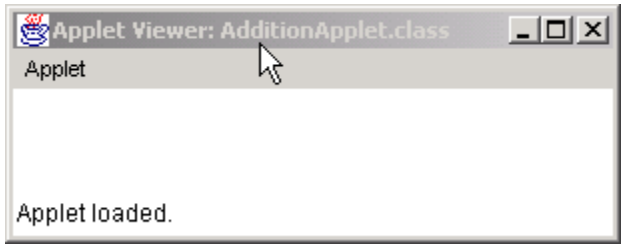
5. Draw applet contents

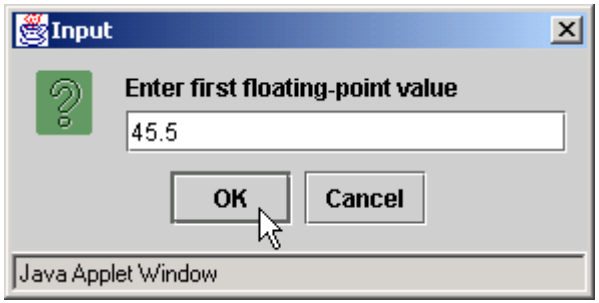
5.1 Draw a rectangle

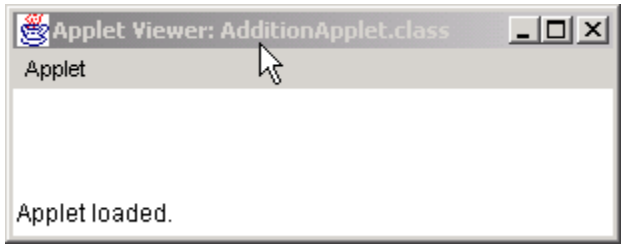
5.2 Draw the results

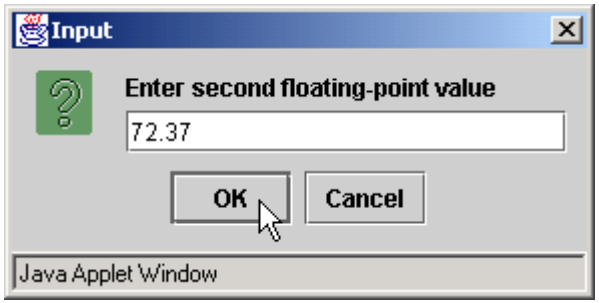
HTML file

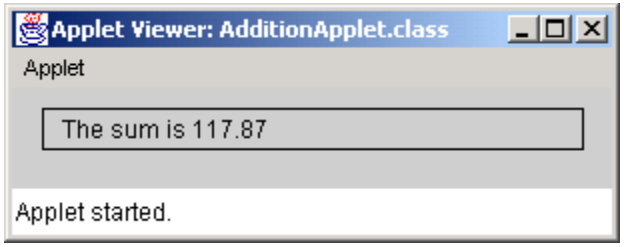
```
1    <html>
2    <applet code = "WelcomeLines.class" width = "300" height = "40">
3    </applet>
4    </html>
```











3.5 Another Java Applet: Adding Floating-Point Numbers

- Lines 1-2: Comments

```
5    import java.awt.Graphics;
```

- Line 5: **`import`**s class **`Graphics`**
  - **`import`** not needed if use full package and class name

    `public void paint ( java.awt.Graphics g )`

```
8    import javax.swing.*;
```

- Line 8: specify entire **`javax.swing`** package
  - **`*`** indicates all classes in **`javax.swing`** are available
    - Includes **`JApplet`** and **`JOptionPane`**
    - Use **`JOptionPane`** instead of **`javax.swing.JOptionPane`**
  - **`*`** does not not load all classes
    - Compiler only loads classes it uses

3.5 Another Java Applet: Adding Floating-Point Numbers

```
10   public class AdditionApplet extends JApplet {
```

- Begin class definition
  - Inherit from **`JApplet`**, **`import`**ed from package **`javax.swing`**

```
11      double sum;  // sum of values entered by user
```

- Instance variable declaration
  - Each object of class gets own copy of the instance variable
  - Declared in body of class, but not inside methods
    - Variables declared in methods are local variables
    - Can only be used in body of method
  - Instance variables can be used anywhere in class
  - Have default value (`0.0` in this case)

3.5 Another Java Applet: Adding Floating-Point Numbers

```
11      double sum;  // sum of values entered by user
```

- Primitive data type **`double`**
  - Used to store floating point (decimal) numbers

```
14      public void init()
```

- Method **`init`**
  - Normally initializes instance variables and applet class
  - Guaranteed to be first method called in applet
  - First line must always appear as above
    - Returns nothing (**`void`**), takes no arguments

```
15      {
```

- Begins body of method **`init`**

3.5 Another Java Applet: Adding Floating-Point Numbers

```
16          String firstNumber;   // first string entered by user
17          String secondNumber;  // second string entered by user
18          double number1;       // first number to add
19          double number2;       // second number to add
```

- Declare variables
- Two types of variables
  - Reference variables (called references)
    - Refer to objects (contain location in memory)
      - Objects defined in a class definition
      - Can contain multiple data and methods
    - **`paint`** receives a reference called **`g`** to a **`Graphics`** object
    - Reference used to call methods on the **`Graphics`** object
  - Primitive data types (called variables)
    - Contain one piece of data

3.5 Another Java Applet: Adding Floating-Point Numbers

```
16          String firstNumber;   // first string entered by user
17          String secondNumber;  // second string entered by user
18          double number1;       // first number to add
19          double number2;       // second number to add
```

- Distinguishing references and variables
  - If data type is a class name, then reference
    - **`String`** is a class
    - **`firstNumber`**, **`secondNumber`**
  - If data type a primitive type, then variable
    - **`double`** is a primitive data type
    - **`number1`**, **`number2`**

3.5 Another Java Applet: Adding Floating-Point Numbers

```
22          firstNumber = JOptionPane.showInputDialog(
23             "Enter first floating-point value" );
```

- Method **`JOptionPane.showInputDialog`**
  - Prompts user for input with string
  - Enter value in text field, click **OK**
    - If not of correct type, error occurs
    - In Chapter 14 learn how to deal with this
  - Returns string user inputs
  - Assignment statement to string
- Lines 26-27: As above, assigns input to **`secondNumber`**

3.5 Another Java Applet: Adding Floating-Point Numbers

```
30          number1 = Double.parseDouble( firstNumber );
31          number2 = Double.parseDouble( secondNumber );
```

- **`static`** method **`Double.parseDouble`**
  - Converts **`String`** argument to a **`double`**
  - Returns the **`double`** value
  - Remember static method syntax
    - ClassName.methodName(arguments)

```
34          sum = number1 + number2;
```

- Assignment statement
  - **`sum`** an instance variable, can use anywhere in class
    - Not defined in **`init`** but still used

3.5 Another Java Applet: Adding Floating-Point Numbers

```
33    }
```

- Ends method **`init`**
  - **`appletviewer`** (or browser) calls inherited method **`start`**
  - **`start`** usually used with multithreading
    - Advanced concept, in Chapter 15
    - We do not define it, so empty definition in **`JApplet`** used
  - Next, method **`paint`** called

```
45        g.drawRect( 15, 10, 270, 20 );
```

- Method **`drawRect( x1, y1, width, height )`**
  - Draw rectangle, upper left corner (**`x1, y1`**), specified **`width`** and **`height`**
  - Line 45 draws rectangle starting at (15, 10) with a width of 270 pixels and a height of 20 pixels

3.5 Another Java Applet: Adding Floating-Point Numbers

```
48          g.drawString( "The sum is " + sum, 25, 25 );
```

- Sends **drawString** message (calls method) to **Graphics** object using reference **g**
  - **"The sum is" + sum** - string concatenation
    - **sum** converted to a string
  - **sum** can be used, even though not defined in **paint**
    - Instance variable, can be used anywhere in class
    - Non-local variable

3.6 Viewing Applets in a Web Browser

- Applets can execute on Java-enabled browsers
  - Many different browser version supporting different Java version specifications
    - Some support for Java 1.0, many for Java 1.1 inconsistently
  - Netscape Navigator 6 supports Java 2 (section 3.6.1)
  - Use Java Plug-in to execute Java 2 applets on other browsers (section 3.6.2)

Fonts

- By default, strings are drawn using the default font, plain style and default font size.
- To change the font, style or size, we need to create an object of class *java.awt.Font* and pass it to the setFont() method of the graphics object used in the paint() method. To create such a Font object, we need to specify the following parameters:
  - The font name.
  - The Style (Font.PLAIN, Font.BOLD, Font.ITALIC, or Font.BOLD + Font.ITALIC).
  - The font size.
- The font name can be the name of any font available on the particular computer (such as: TimesRoman, Courier, Helvetica etc.,) or any of the logical names shown in the table below:

Serif	A font with small segments at the end, e.g. Times New Roman
SansSerif	A font without small segments, e.g. Helvetica
Monospaced	A font in which all characters have the same width. e.g. Courier
Dialog	A screen font suitable for labels in dialogs
DialogInput	A screen font suitable for user input in text fields

Fonts (cont'd)

- Example: The following applet displays the word **Applet** in large SansSerif font.

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

public class BigFontApplet extends Applet{
  public void paint(Graphics g){
    final int SIZE = 48;
    Color myColor = new Color(0.25F, 0.5F, 0.75F);
    Font myFont = new Font("SansSerif", Font.BOLD,
SIZE);
    g.setColor(myColor);
    g.setFont(myFont);
    g.drawString("Applet",5,60);
  }
}
```

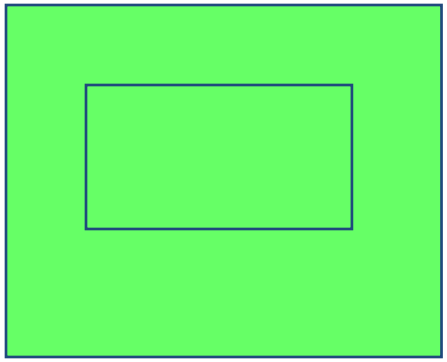
Fonts (cont'd)

- If an applet will use several fonts, it is good to create the font objects in the **init()** method:

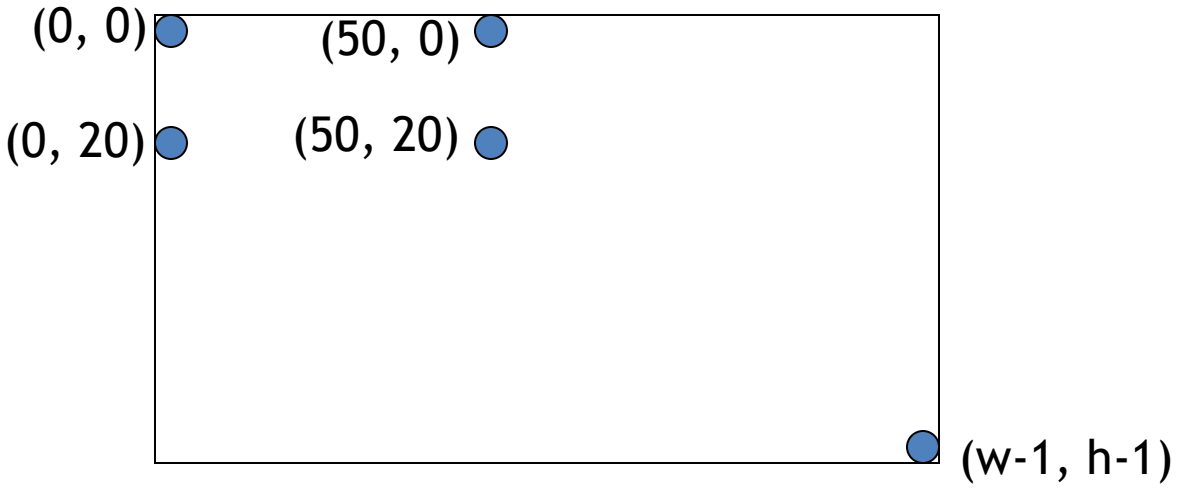
```
import java.applet.*;
import java.awt.*;
   public class FontExamples extends Applet{
       private Font f, fb, fi, fbi;
       public void init() {
                   setBackground(Color.yellow);
                   f = new Font("TimesRoman", Font.PLAIN, 18);
                   fb = new Font("Courier", Font.BOLD, 20);
                   fi = new Font("TimesRoman", Font.ITALIC, 18);
                   fbi = new Font("Helvetica", Font.BOLD + Font.ITALIC, 25);
        }
        public void paint(Graphics g){
                   g.setColor(Color.blue);
                   g.setFont(f);
                   g.drawString("This is TimesRoman plain font", 10, 25);
                   //...
       }
   }
```

Drawing in an Applet

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

// assume that the drawing area is 150 by 150
public class SquareAndRectangle extends Applet
{
  final int areaSide = 150 ;
  final int width = 100, height = 50;

  public void paint ( Graphics gr )
  {
    setBackground( Color.green );
    gr.setColor( Color.red );

    // outline the drawing area
    gr.drawRect( 0, 0, areaSide-1, areaSide-1 );

    // draw interiour rectange.
    gr.drawRect( areaSide/2 - width/2 ,
        areaSide/2 - height/2, width, height );
  }
}
```



Java's coordinate system



- Java uses an (x, y) coordinate system
- (0, 0) is the top left corner
- (50, 0) is 50 pixels to the right of (0, 0)
- (0, 20) is 20 pixels down from (0, 0)
- (w - 1, h - 1) is just inside the bottom right corner, where w is the width of the window and h is its height

Drawing rectangles

- There are two ways to draw rectangles:
- g.drawRect(***left*** , ***top*** , ***width*** , ***height***);

- g.fillRect(***left*** , ***top*** , ***width*** , ***height***);

Drawing strings

- A String is a sequence of characters enclosed in double quote marks
  - "Hello, World!"
- A double quote mark in a String must be preceded by a backslash (\ )
  - "He said, \"Please don't go!\" "
  - To draw a string, you need to specify not only *what* you want to say, but *where* to say it
  - g.drawString(***string***, ***left***, ***top***);
- For example,
  - g.drawString("Example JApplet", 20, 80);

The complete applet

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

// CIT 591 example

public class Drawing extends JApplet {

    public void paint(Graphics g) {

        g.setColor(Color.BLUE);
        g.fillRect(20, 20, 50, 30);

        g.setColor(Color.RED);
        g.fillRect(50, 30, 50, 30);

        g.setColor(Color.BLACK);
        g.drawString("Example JApplet", 20, 80);
    }
}
```

More java.awt.Graphics methods

- g.drawLine(***x1***, ***y1***, ***x2***, ***y2***);
- g.drawOval(***left***, ***top***, ***width***, ***height***);
- g.fillOval(***left***, ***top***, ***width***, ***height***);
- g.drawRoundRect(***left***, ***top***, ***width***, ***height,***
  arcWidth, ***arcHeight***);
  - ***arcWidth***, ***arcHeight*** define the "roundedness" of corners
- g.fillRoundRect(***left***, ***top***, ***width***, ***height, arcWidth***, ***arcHeight***);
- g.drawArc(***left***, ***top***, ***width***, ***height***, ***startAngle***, ***arcAngle***);
  - Angles are in degrees
  - 0 degrees is the 3 o'clock position
  - Positive angles are to the right
- g.FillArc(***left***, ***top***, ***width***, ***height***, ***startAngle***, ***arcAngle***);

Still more Graphics methods

- g.drawPolygon(*xPoints*, *yPoints*, *n*);
- g.fillPolygon(*xPoints*, *yPoints*, *n*);
  - *xPoints* and *yPoints* are int arrays of size *n*
  - One way to write an int array is:
    new int[] { *value1*, *value2*, …, *valueN*}
  - Example: g.drawPolygon(new int[] { 250, 290, 210 },
    new int[] { 210, 290, 290 }, 3);
    draws a triangle using the 3 points (250, 210), (290, 290), and (210, 290).
- g.drawPolyline(*xPoints*, *yPoints*, *n*);
  - A "polyline" is like a polygon, except the first and last points are not automatically connected
  - Hence, there is no "fillPolyline" method

The HTML page

- You can only run an applet from an HTML page
- The HTML looks something like this:
  -
```
<html>
    <body>
        <h1>Drawing Applet</h1>
        <applet code="Drawing.class"
                        width="100"  height="150">
        </applet>
    </body>
</html>
```
- Eclipse will create this HTML for you
- You don’t even need to think about the HTML just yet

Other useful JApplet methods

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