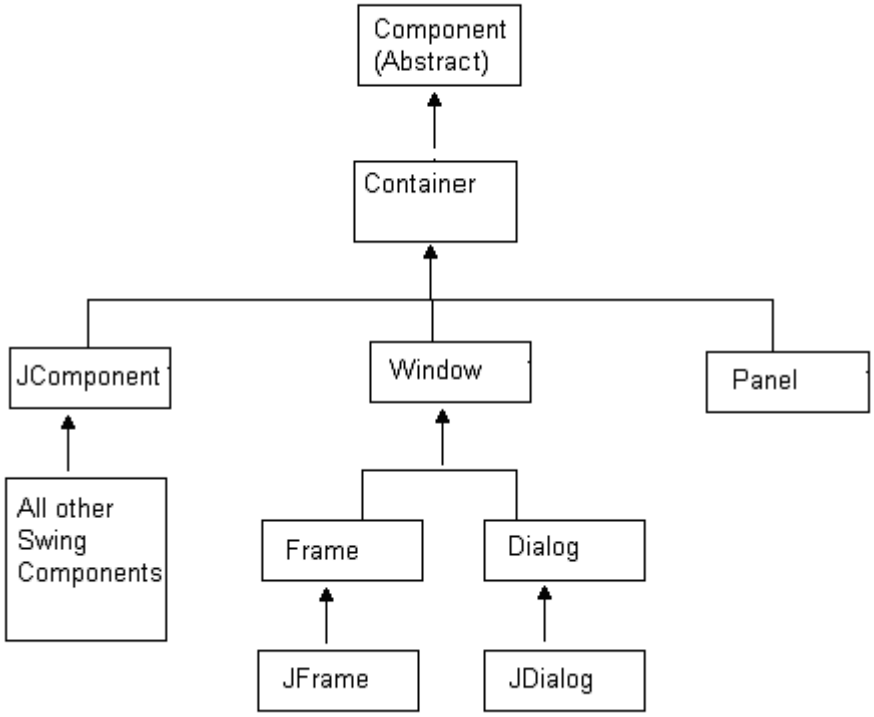
Class 16 Notes

Graphics: AWT and Swing

Components and Containers

Java provides a hierarchy of classes that facilitates GUI programming. Notice that Component is at the top and everything else ***is-a*** Component. We will look more closely at the classes as we move on.



The *Component* hierarchy

At the top of the hierarchy is the (abstract) Component class. We will discuss the other classes later.

Components:

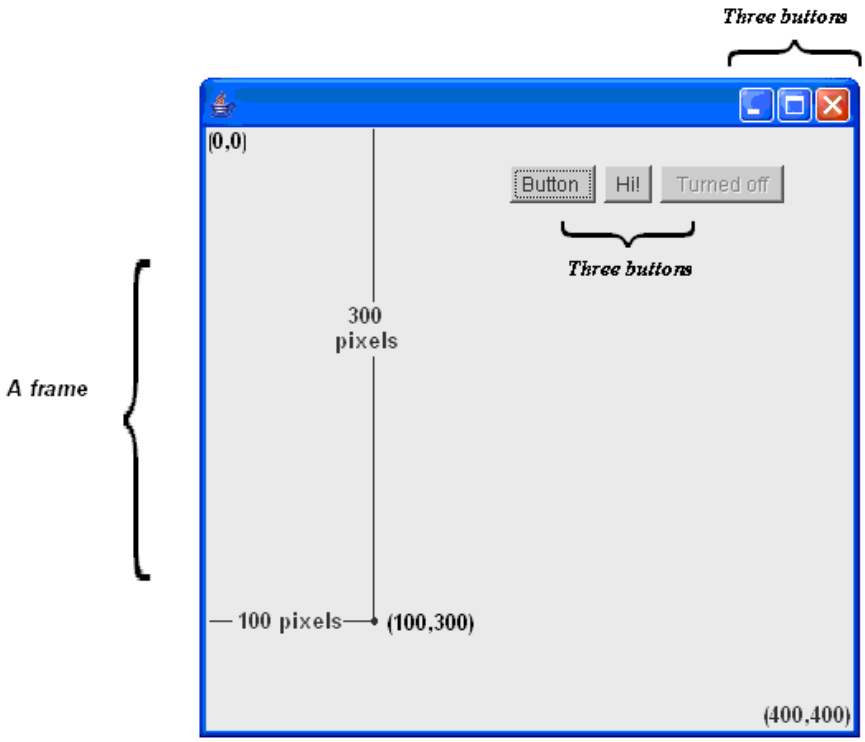
A ***component*** is an object that can be displayed on the screen.

For example, buttons, text boxes, checkboxes and labels are all components. A window is also a component. All the things that you see in a GUI (Graphical User Interface) are components and they extend Java's Component

class. So when I use the term component it can refer to a button, or a check box, or a text box etc. Pretty much everything ***is-a*** Component.

Under Component is Container. Look at the diagram above. The Container class extends Component.

A *container* is an object that holds components. A window is a container. A window holds buttons, pictures, check boxes , menus etc**. Note: in Java what you traditionally call a window is called a *frame*. (In Java a "window" is part of a frame. A window is the gray area in the picture below. A frame is the whole thing, including the blue title bar at the top.)**



Six buttons in a frame

COORDINATES
The upper left corner of a container (eg. A frame) is designated as position (0, 0) and (*x, y*) is the point located *x* pixels to the right of and *y* pixels **down** from (0, 0). For example, the dot in the frame above marks the point (100, 300), which is 100 pixels right of (0, 0) and 300 pixels down from (0, 0).
In this example, (400, 400) is the point in the bottom right corner—400 across and 400 down.

Notice, there are no negative coordinates. This is not the standard xy coordinate syster,

So at this point there is not much to understand:

- Buttons, labels, text boxes, checkboxes are all Components
- What you traditionally call a window, Java calls a frame
- A frame is=a Container that holds components. You will place components such as buttons etc in a frame
- For any component, the upper left corner has coordinates (0,0) and the point (x,y) is located x pixels to the right and y pixels down.

These are the packages you need to import.

Abstract Windows Toolkit and Swing

- We will use two packages that contain the classes for graphics programming:
  the original Abstract Windows Toolkit (AWT) and
  the Swing package.
  (There is another that we will not be using.)

- AWT is the original class library for graphics programming. Swing is newer

- The AWT classes are in java.awt

- The Swing classes reside in javax.swing. == Notice the **x** in javax

  Swing classes are all prefixed with uppercase *J*. For example JButton, JCheckBox, JFrame and JMenu are Swing classes that encapsulate buttons, checkboxes, frames, and menus – your everyday, standard components.

- All Swing components except JFrame and JDialog extend JComponent
  Go back and look at the hierarchy.

- We will be using Swing components in our programs but most of our programs will use both packages. So you will usually import both:

```
import java.awt.*:
Import javax.swing.*;    // notice javax---  don't forget the x
```

So the classes that we will need are in javax.swing.* and java.awt.*

OK…now we are ready to set up a frame. It will do nothing other than appear on the screen.

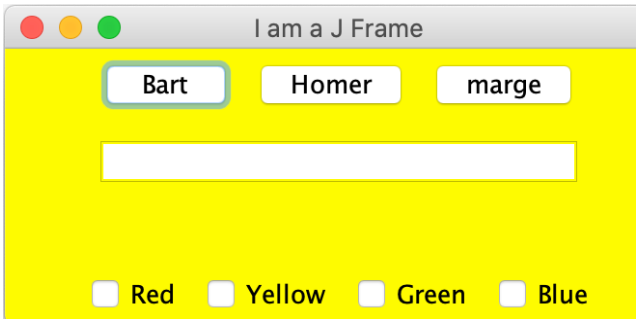
Windows and Frames

A Java Window is a "window" without borders and a title bar.

> **A JFrame is what you normally think of as a "window," and it is the primary container used in all our applications. Every GUI application will have a JFrame**

Remember, all Swing classes begin with J.

Here is a JFrame created on my Mac. It will look a little different if created on a Windows machine but will have the same components



Notice

- A title bar with the title "I am a JFrame"
- Three buttons -- JButton objects
- One textfield -- a JTextField object
- Four labeled Check boxes -- three JCheckBox objects

To create a GUI we **start with Java's plain vanilla JFrame and extend it** adding components JButtons, JTextFields, JMenus, JCheckBoxes etc.

So for example we might begin

```
public class MyFrame extends JFrame      // MyFrame is-a JFrame plus whatever I add
```

To do that we rely on the methods of JFrame and those inherited from Component

Two JFrame constructors are:

- JFrame()
  creates a new JFrame object that is initially invisible.

- JFrame(String title)
  creates a new JFrame with a title that appears on the title bar. In the picture above the title is "I am a JFrame"

- **void setResizable(boolean x)**
  If x is true, the frame can be resized by the user; if x is false the frame cannot be resized. By default, a frame is resizable ..that means a user can drag it bigger or smaller. So setResizable(false) will not allow the user to resize the frame with the mouse

In addition a JFrame (and EVERY component) inherit methods from Component:
Here are just a few there are many more.

A few useful methods of the Component class and all classes that extend Component are:

- **void setBounds**(int x, int y, int width, int height)
  places a component at position (x, y) of its container and sizes the component to the number of pixels specified by the parameters width and height.
  If you are creating a frame then setBounds(0,0, 500,200) places the frame at position (0,0) **of your screen** and makes it size 500 x 200 pixels. setBounds (50,90, 300,300) places the frame at screen position (50,90) with size 300x300.

  If you leave this out, your frame will be size 0 x 0 and you will not see it.

- **int getHeight()**
  returns the height in pixels of a component.

- **int getWidth()**
  returns the width in pixels of a component

- **void setVisible(boolean x)**
  hides the component if the parameter is false; displays the component if the parameter is true.
  Without this your frame will be invisible.

- **int getX()**
  returns the *x*-coordinate of the component, i.e., the *x*-coordinate of the upper left corner of the component.

- **int getY()**
  returns the *y*-coordinate of the component, i.e., the *y*-coordinate of the upper left corner of the component.

The Container class defines additional methods. The most important of these is

add(Component c),

which places a component, c, in a container. Use this method to place a component, such as a button, in a frame. So if b is a button, add(b) places the button in the frame.

Read through the following example. Make sure you understand every line of code. If you have a question, please ask me. The run the example yourself.

Example : Make an empty frame. Place it at position(0,0) on your screen. Make the size 300 x 300 and give it a title: "This is a frame" See the picture below
Notice that the work is done in the constructor.

```
import javax.swing.*;

public class MyFirstFrame extends JFrame
{
    public MyFirstFrame ()                          // creates a frame with title "I've been framed!"
    {
        super("This is a frame ");                  // call the one-argument constructor of JFrame
        setBounds(0,0,300,300);                     // placed at screen position (0,0); size 300 by 300
         setResizable(false);                       // user cannot adjust the size
         setVisible(true);                          // very important or you will see nothing
    }
}
```

Do not forget to include setVisible(true). It should be the last statement

The following test class creates, displays, a MyFirstFrame frame.

```
import javax.swing.*;
public class TestMyFirstFrame
{
    public static void main(String[] args)
    {
        JFrame frame = new MyFirstFrame ();  // since MyFirstFrame is-a JFrame
    }
}
```

A frame in the upper left-hand corner of the screen

Even though this is a very simple program, you should type it into Java and make sure you know what each line does.

The method setBackground(Color.BLUE) makes the background of a JFrame blue. (You can use Color.RED or Color.GREEN etc).

Everything should be done in the constructor, as in the last example. Use the last example as a guide.

Adding Components to a Frame

For now we will just add buttons to our frames. The buttons will do nothing. That will come later.

> **A *button* is a component that displays some text or image and allows some action to occur when the button is "pressed" – i.e., when the mouse is clicked on the button.**
> So a button can have a picture on it or just some text such as "Exit"

A button is a member of the JButton class. Three constructors of JButton are:

- JButton(),
  creates a button with no text

- JButton(String text),
  For example
  JButton b = new JButton("Exit") creates a button, b, with the word "Exit" on it

- JButton(**new ImageIcon (String filename))**
  displays an image on the button , where filename is the name of an image file, such as Homer .jpg or sheldon.gif.

  For example
  JButton b = new JButton(**new ImageIcon("Homer.jpg")** creates a button with a picture (Homer.jpg) on it.

OK…Now how do we place buttons on a JFrame?????

To add components to a frame, Java provides *layout managers*.

> **A *layout manager* is an object that arranges components in a container such as a frame.**

Think of a layout manager as a decorator whom you've hired to arrange furniture in a new apartment. The decorator may place the couch in one place, the TV in another, a table in another.

A Java layout manager does just that with components. A layout manager places components (rather than furniture) in a frame (instead of a room). We will use a layout manager to place buttons (or any other component) on a JFrame. There are several different layout managers and each places components differently. Each has a mind and scheme of its own;

A layout manager is an object and thus belongs to a class. Those classes that we discuss are:

- BorderLayout,
- FlowLayout, and
- GridLayout.

There are others. Each layout manager works differently and place components in a frame differently just as different decorators might lay out furniture differently in a room. **I know this seems weird but an example should help and you will catch on.**

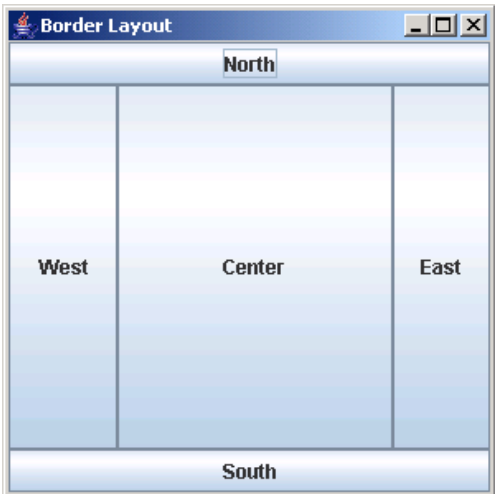
BorderLayout

BorderLayout is the default layout manager for JFrame.
That is, unless you specifically choose a different layout manager for a frame, components are placed in a frame using the BorderLayout layout manager.

So how does a border layout manager place components in a frame?????.

The BorderLayout manager is rather simplistic and divides a frame into five areas:

NORTH WEST SOUTH EAST CENTER



***BorderLayout* partitions a frame into five regions**

The method
add(Component c, int region)

places a component into a container (eg. A frame). The parameter, *region*, is specified by one of the constants
BorderLayout.NORTH,
BorderLayout.SOUTH,
BorderLayout.EAST,
BorderLayout.WEST, or
BorderLayout.CENTER.

For example, if b is a button the statement,
add (b, BorderLayout.CENTER)
places b in the center of the frame.

If no region is specified, a BorderLayout layout manager places a component in the center region. Only one component can be placed in a region and components are resized to fit the entire region. SO ONLY FIVE COMPONENTS CAN BE PUT IN A JFRAME. The picture above has five buttons but **the buttons fill each entire region. Wow, big buttons!**

Here is an example. Again, be sure to understand every line of code. One thing to notice is that I created the frame in the constructor.

Example

- Create a class, **BorderLayoutFrame, that extends JFrame** and displays five buttons.
- Place the five buttons in the frame using the default BorderLayout layout manager.
- The center button should display the picture, seinfeld.jpg.
- The other four buttons should display the words Jerry , Kramer, Elaine, and George
- The size of the frame should be 400 by 350 and the frame should be positioned at (0,0).
- Include a main(...) method that instantiates the frame.

```
1.  import javax.swing.*;
2.  import java.awt.*;

3.  public class BorderLayoutFrame extends JFrame
4.  {
5.      public BorderLayoutFrame()                  // default constructor
6.      {
7.          super("BorderLayout ")                   //call one-argument constructor of JFrame
8.          setBounds(0,0, 400,350 );                // position and size of the  frame

9.          // add the center button; the button displays the image  "seinfeld.jpg"
10.         JButton picture = new JButton(new ImageIcon("seinfeld.jpg");      // create a button with a jpg
11.         add(picture, BorderLayout.CENTER);                                // place in the CENTER region

12.         // add four buttons to NORTH, SOUTH, EAST, and WEST
13.         JButton jerry = new JButton("Jerry");                 // create a JButton with label "Jerry"
14.         add(jerry, BorderLayout.NORTH);                       // place it in the NORTH region
15.
16.         JButton elaine = new JButton("Elaine");               // create a JButton --elaine
17.         add(elaine,BorderLayout.SOUTH);                       //  place it in the SOUTH region
18.
19.          JButton kramer = new JButton("Kramer");              // create a JButton--kramer
20.          add(kramer, BorderLayout.EAST);                      // place in the EAST region
21.
22.          JButton george= new JButton("George");               // create a JButton – George
23.          add(george,BorderLayout.WEST);                       // place in WEST region
24.         setVisible(true);                                     // VERY IMPORTANT
25.     }

26.     public static void main(String[] args)
27.     {
28.         JFrame frame = new BorderLayoutFrame ();  // or BorderLayoutFrame frame = new BorderLayoutFrame ();
29.     }
30. }
```

Note: instead of add(button, region) you could also say *this.add(button, region)*. Either works.
For example, you could have coded the same statement as

```
this.add(kramer, BorderLayout.EAST);
```

The output is a frame on the next page

Five *JButtons*, one displaying an *ImageIcon*, placed with the default layout manager, *BorderLayout*

You should run this program you use and make sure you understand every line.
The pictures are in the content section of the course on elearn

Using BorderLayout the frame can hold only five components, and components can be covered by other components. For instance, if the additional statement

```
JButton newman = new JButton(new ImageIcon("Newman.jpg");
add(newman, BorderLayout.CENTER);
```

is added to the constructor after line 23 the frame would appear as. The original center button is covered and we see notorious Newman in the center. Only five components can be placed using BorderLayout.

OK—This is the end of our first GUI class. You should be able to

- **make a JFrame and**
- **add buttons to the frame.**

It may seem like BorderLayout is dumb because there can be just five components in the frame but in the next few classes you will see it is pretty useful.

In the next class we will look at two other layout managers. These layout managers have their own unique ways of placing components in a frame.