EECS 1720
Building Interactive Systems

Lecture 12 :: Graphical User Interfaces (GUI) - 1

Aside: Inheritance within Processing?

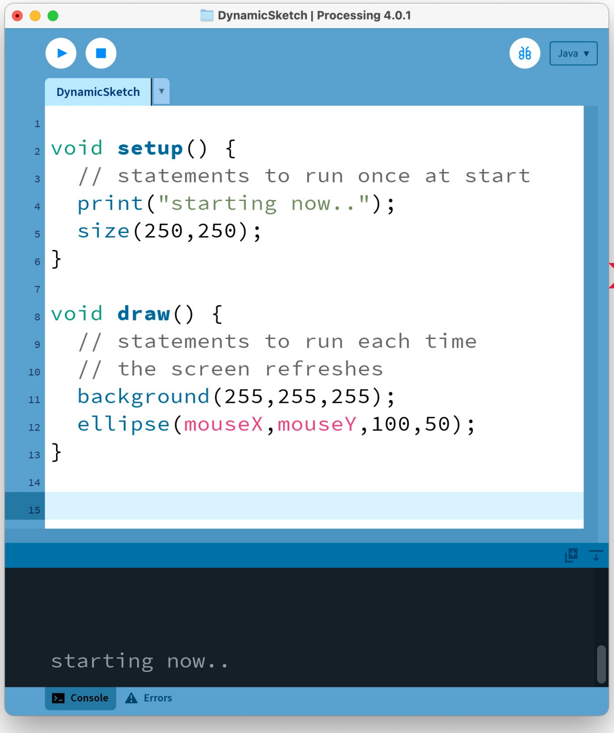
Let’s review how a Processing application is actually setup and run in full Java…

This will illustrate both HAS-A and IS-A relationships in action

- PApplet class
  - base (super) class that holds much of the Processing core
  - to use this, need to add the “core.jar” file from processing to your project (and link it in the build configuration for the project)… then it may be easily imported

Recall: Processing sketch converted to Java

DynamicSketch.pde



```

void setup() {
  // statements to run once at start
  print("starting now..");
  size(250,250);
}

void draw() {
  // statements to run each time
  // the screen refreshes
  background(255,255,255);
  ellipse(mouseX,mouseY,100,50);
}

```

```
starting now..
```

Processing ~ Java (lite)

DynamicSketch.java

```
import processing.core.PApplet;

public class DynamicSketch extends PApplet {

        public void settings() {
         print("starting now..");
         size(250,250);
        }

        public void draw() {
         background(255,255,255);
         ellipse(mouseX,mouseY,100,50);
        }

        public static void main(String[] args) {
         String[] processingArgs = {"HelloSketch"};
         DynamicSketch mySketch = new DynamicSketch();
         PApplet.runSketch(processingArgs, mySketch);
        }
}
```

Java (full) – note the extra scaffolding!

DynamicSketch.class
(Byte Code)

Processing within Eclipse IDE

- https://www.eclipse.org/downloads/packages/
    - download “Eclipse IDE for Java Developers” for your operating system, and install (1st step – already done for most of you)

- Assumes you already have Processing installed…
    - we will have to then tell Eclipse about where Processing is
        - Can add processing core as an external jar (java archive) file to a new java project
        - Easier way: install “Proclipsing” plugin (which allows us to open ”processing” projects – which include all the relevant jar files we need for processing)

Proclipsing plugin for Eclipse

- https://github.com/ybakos/proclipsing

Installing

1. Download a release **.zip** file.
2. Unzip the file and place the folder in a convenient location.
3. In Eclipse, select the menu item *Help > Install New Software*. In the dialogue that appears, select the *Add...* button. Enter **Proclipsing** as the *Name:*, and select the *Local...* button. Navigate to the location of your Proclipsing folder, and within, select the **proclipsingSite** folder and select the *Open* button.
4. Back in the *Install* dialogue, select the *Select All* button, and then select *Next >*.
5. Accept the license agreement and select *Finish*. If you are prompted to restart Eclipse, do it.

Making an Eclipse Project that uses Processing??

```
package simpleproject;
import processing.core.PApplet;

public class SimpleProject extends PApplet {

    public void setup() {

    }

    public void draw() {

    }

}
```

Opening a Processing Project

```
package simpleproject;
import processing.core.PApplet;

public class SimpleProject extends PApplet {

    public void settings() {

    }

    public void draw() {

    }

    public static void main(String _args[]) {
        SimpleProject myApp = new SimpleProject();
        myApp.runSketch();
    }

}
```

rename setup to “settings()” and put setup() code here

put draw() loop code here

These methods override PApplet’s versions

Use main (or a client) to instantiate your class and invoke runSketch() to run it (a method inherited from PApplet)

PApplet

- PApplet is a Java class that “**encapsulates**” core methods/components from Processing
  - i.e. everything needed to support a Processing “sketch”

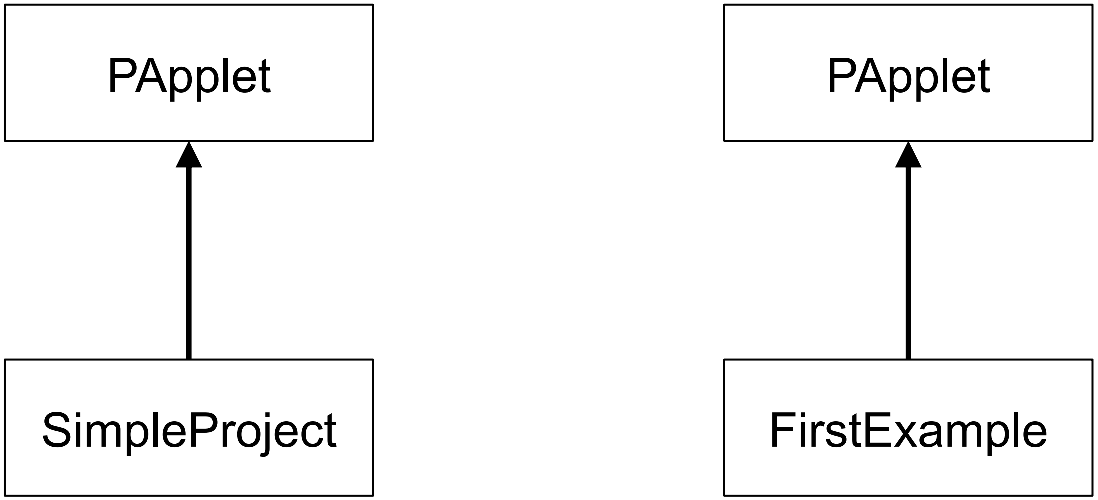
- A class that is capable of being “run”
  - PApplet.*runSketch*(..);

- What else can we tell about PApplet here?
  - Can use as a base class for your own “app”
  - Can use a reference to this class (that extends) to access Processing functionality
  - Most of the accessible features are STATIC!
  - Can create instances (objects) of this class (or PApplet) and then invoke runSketch() method to launch the app window
  - Any reference to PApplet or an instance of (PApplet or a child class of PApplet), can be used to access all of processing’s features (methods, constants, etc.)

How do we use this class to make an app?

- Use the new subclass (of PApplet), to override settings() and draw() methods … or even other methods like mousePressed() etc..
    - Develop your own classes, and have this subclass maintain HAS-A relationships with any new classes you construct, then they can be used within your class, which ”IS-A” PApplet
    - If you need to draw/access graphics from your own classes, make sure those classes have a reference to this subclass!
        - i.e. Develop your own Java classes, such that they each ”hold” or “HAS-A” reference to this PApplet
        - Now, we can use that reference within the class to draw/invoke any processing functionality

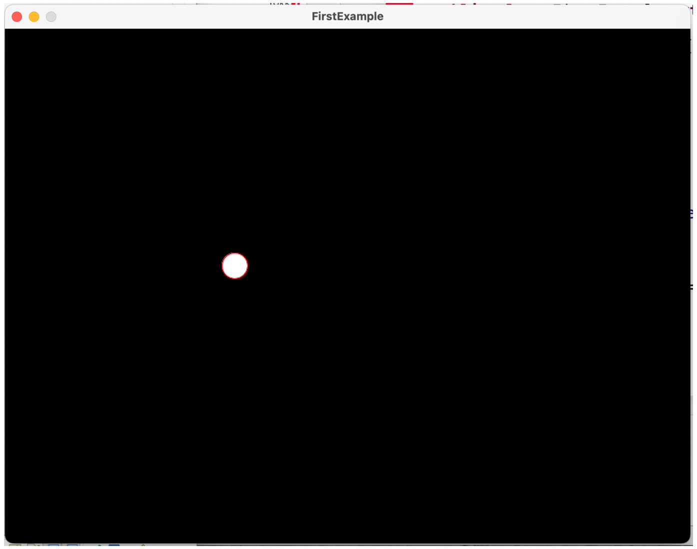
These examples will CLARIFY the different class relationships so far (HAS-A aggregation/composition and IS-A)

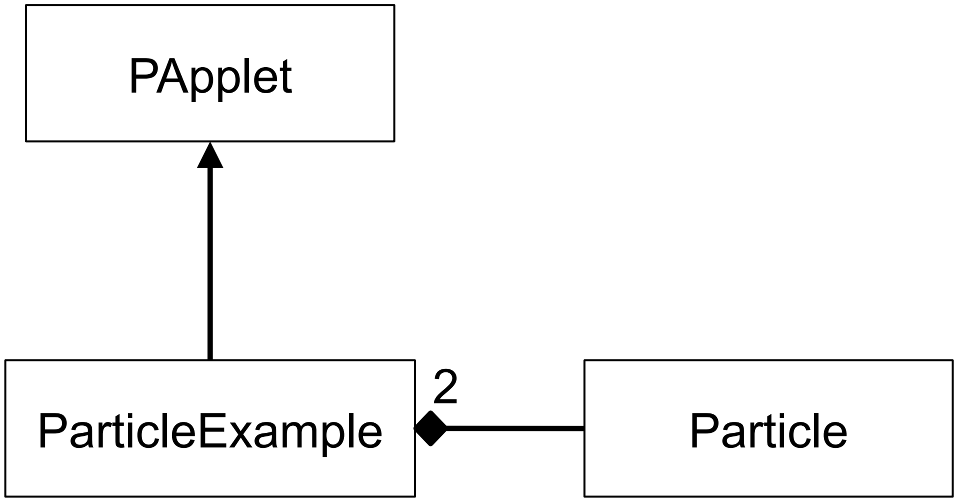


SimpleProject extends PApplet (SimpleProject “is-a” PApplet)
FirstExample extends PApplet (FirstExample “is-a” PApplet)

Example (from 1710 - simple draw app)

```
package firsttest;
import processing.core.PApplet;

public class FirstExample extends PApplet {

    public void settings() {
        // USE THIS INSTEAD OF SETUP()
        size(800,600);
    }

    public void draw() {
        background(0);
        stroke(255,0,0);
        fill(255);
        ellipse(mouseX,mouseY,30,30);
    }

    public static void main(String[] args) {
        FirstExample myApp = new FirstExample();
        myApp.runSketch();
    }
}
```





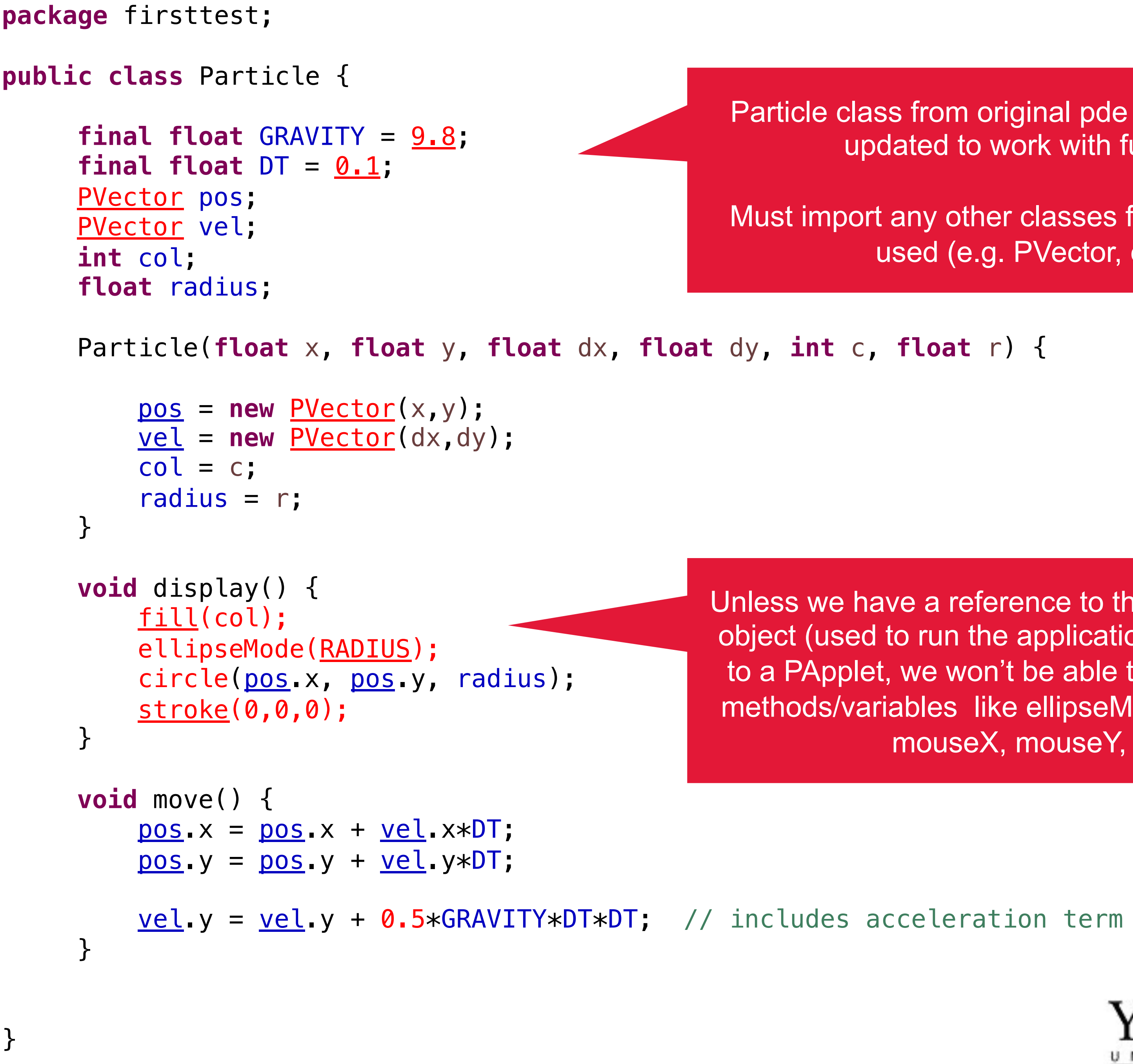
ParticleExample extends PApplet (ParticleExample “is-a” PApplet)
ParticleExample “has-a” Particle (actually 2 Particles: bullet and firework)

```
package firsttest;

import processing.core.PApplet;

public class ParticleExample extends PApplet {

    public Particle bullet;
    public Particle firework;

    public void settings() {

        size(800,600);

        bullet = new Particle(0,height/2,10,-10,
                     color(random(255),random(255),random(255)),
                     random(10)+10);

        firework = new Particle(width/4,height, 0,(float) (-10.0+random(10)-10.0),
                     color(random(255),random(255),random(255)),
                     random(10)+10);
    }

    public void draw() {
        background(255,255,255);
        bullet.display(this);
        bullet.move();
        firework.display(this);
        firework.move();

    }

    public static void main(String _args[]) {
        PApplet.main(new String[] { firsttest.ParticleExample.class.getName() });

    }
}
```

“HAS-A”

settings() is run once after instantiation of this class.. essentially PApplet’s runSketch() runs this for us

Another way to instantiate this class: PApplet has a main method defined, it is static, and can be invoked by passing a String[ ] args to it (just like passing from the command line). PApplet’s main() will then take care of instantiating this class and calling runSketch



```
package firsttest;

public class Particle {

    final float GRAVITY = 9.8;
    final float DT = 0.1;
    PVector pos;
    PVector vel;
    int col;
    float radius;

    Particle(float x, float y, float dx, float dy, int c, float r) {

        pos = new PVector(x,y);
        vel = new PVector(dx,dy);
        col = c;
        radius = r;
    }

    void display() {
        fill(col);
        ellipseMode(RADIUS);
        circle(pos.x, pos.y, radius);
        stroke(0,0,0);
    }

    void move() {
        pos.x = pos.x + vel.x*DT;
        pos.y = pos.y + vel.y*DT;

        vel.y = vel.y + 0.5*GRAVITY*DT*DT;  // includes acceleration term
    }


}
```

Particle class from original pde sketch must be updated to work with full java

Must import any other classes from processing used (e.g. PVector, etc.)

Unless we have a reference to the ParticleExample object (used to run the application), or a reference to a PApplet, we won't be able to use processing methods/variables like ellipseMode, fill, RADIUS, mouseX, mouseY, etc.

```
package firsttest;

import processing.core.PVector;

public class Particle {

    public final float GRAVITY = 9.8f;
    public final float DT = 0.1f;
    public PVector pos;
    public PVector vel;
    public int col;
    public float radius;

    public Particle( float x, float y, float dx, float dy, int c, float r) {
        pos = new PVector(x,y);
        vel = new PVector(dx,dy);
        col = c;
        radius = r;
    }

    public void display(ParticleExample p) {
        p.fill(col);
        p.ellipseMode(ParticleExample.RADIUS);
        p.circle(pos.x, pos.y, radius);
        p.stroke(0,0,0);
    }



    public void move() {
        pos.x = pos.x + vel.x*DT;
        pos.y = pos.y + vel.y*DT;
        vel.y = vel.y + 0.5f*GRAVITY*DT*DT;  // includes acceleration term
    }

}
```

One option is to pass the reference to the methods that need processing features.. Here we pass ParticleExample as a parameter (p) to the display method of Particle.

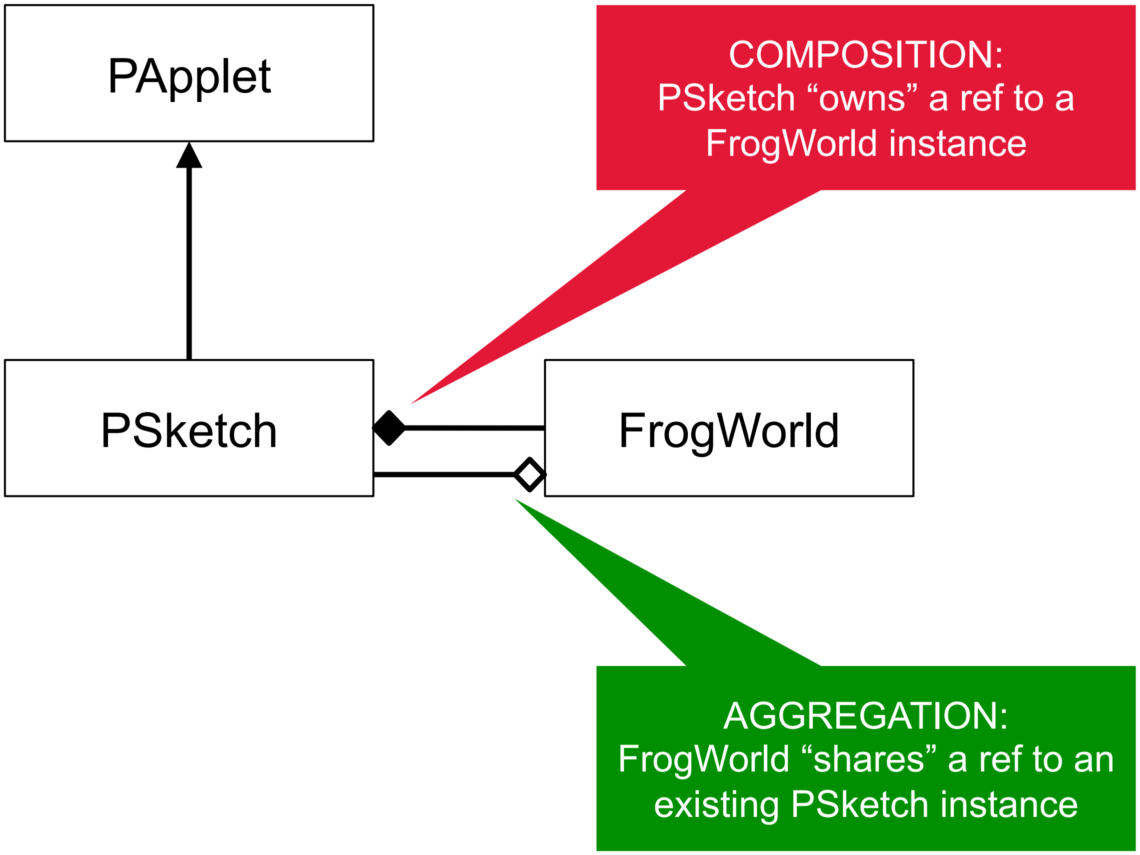
Now the display() method has access to processing draw methods

Note: RADIUS is a static constant inherited from PApplet, so must be accessed using the class name

Another approach ?

- Any classes we want to work with the class that extends PApplet:
  - Pass a reference to its constructor(s) so that they always have a way of accessing processing features.

- Example :   lets re-visit the early FrogWorld example (but this time leveraging processing to do the drawing etc)
  - Let FrogWorld (which has a method display() to render itself), also maintain its own alias to a PApplet instance



PApplet

PSketch

FrogWorld

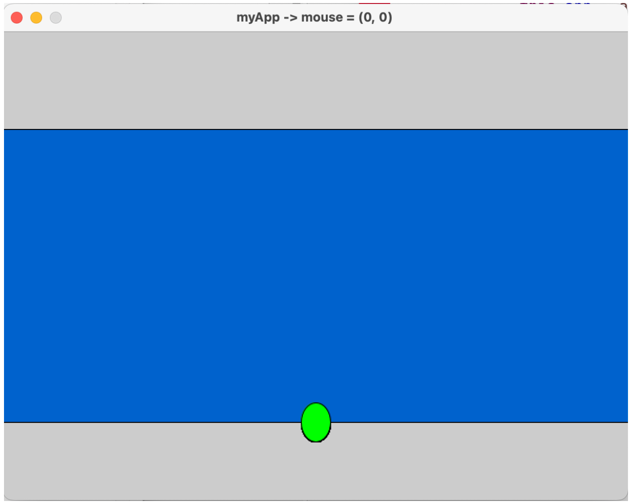
COMPOSITION:
PSketch “owns” a ref to a FrogWorld instance

AGGREGATION:
FrogWorld “shares” a ref to an existing PSketch instance

```
package week06_processing;

import java.awt.Color;
import processing.core.*;


public class PSketch extends PApplet {

     public FrogWorld frogger;  // PSketch “is-a” PApplet, PSketch “has-a” FrogWorld
                               // ref to this FrogWorld instance only exists inside
                               // the PSketch instance (composition)

     public void settings() {
          // run once when PSketch is instantiated and runSketch() invoked

          this.size(640,480);
          this.frogger = new FrogWorld(this);  // pass PSketch ref to FrogWorld ctor

     }

     public void draw() {
          // run at every draw iteration … tell frogger to render

          frogger.render();
     }

     public static void main(String[] args) {

          PSketch myApp = new PSketch();
          myApp.runSketch();

     }
}
```

Most straight-forward way to instantiate and run a PApplet instance..

```
package week06_processing;

import java.awt.Color;

public class FrogWorld {

     public PSketch app;  // FrogWorld "has-a" PSketch (used to access processing)
                          // Note: this ref exists outside of FrogWorld (in PSketch)
                          // therefore, app becomes an “alias” (aggregation)

     public String title;
     public Color frogCol ;
     public Color waterCol ;



     public FrogWorld(PSketch app) {
          this.app = app;
          this.title = "Frogger 1 : ";
          this.frogCol = Color.GREEN;
          this.waterCol = new Color(0, 100, 200);

     }

     public void render() {
          app.windowTitle("myApp -> mouse = (" + app.mouseX + ", "
                                                   + app.mouseY + ")");

          app.fill(waterCol.getRGB());
          app.rect(0, 100, 640, 300);
          app.fill(frogCol.getRGB());
          app.ellipse(320, 400, 30, 40);

     }
}
```



myApp -> mouse = (0, 0)

For Assignment 1

- You may consider creating your classes this way (so that they are linked to, and can leverage processing to do any graphics)

  OR

- Use RasterImage & Graphics2D

… moving on

GUI’s (Graphical User Interfaces)

GUI frameworks
:: make excessive use of Inheritance !!

- Java AWT
  - Abstract Window Toolkit
- Java AWT/Swing
  - Builds on AWT (a little more flexible than AWT alone)
- Java FX (other)
  - More recent (simplifies some aspects, a little more seamless when combining & synchronizing graphics/media)
- G4P (gui tools in processing)
  - Limited toolset (but provides standard GUI controls)
  - Need to install (a bit finicky)

Java AWT

Built in (core) framework for Graphics & GUI’s

Java AWT

- Abstract Window Toolkit (AWT)
  - Helps programmer develop graphics, and..
  - Graphical user interfaces (GUI's)

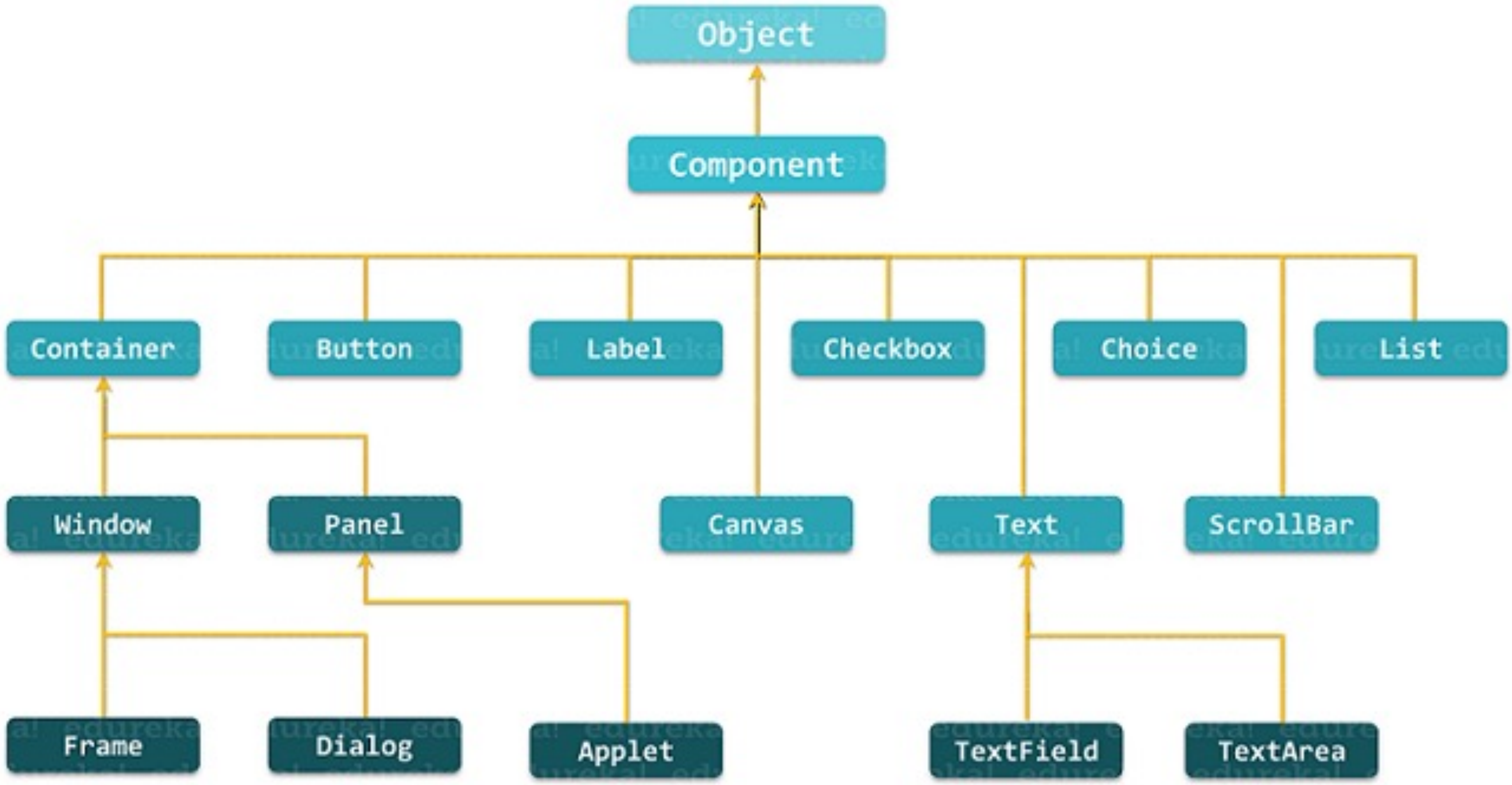
- Abstracts native window toolkits for different platforms
  - e.g. will use windows “window elements”
  - or mac's ”window elements”

when creating windows to draw within and associated interactive elements like scrollbars, buttons etc)

Java AWT Elements

- UI Elements
  - Core visual elements for interacting with an application (via mouse clicks, keys etc)
- Layouts
  - Define how UI elements are organized on the screen
  - Also manages GUI look and feel
- Behaviour
  - Events and Event handling (later)

Java AWT Class hierarchy



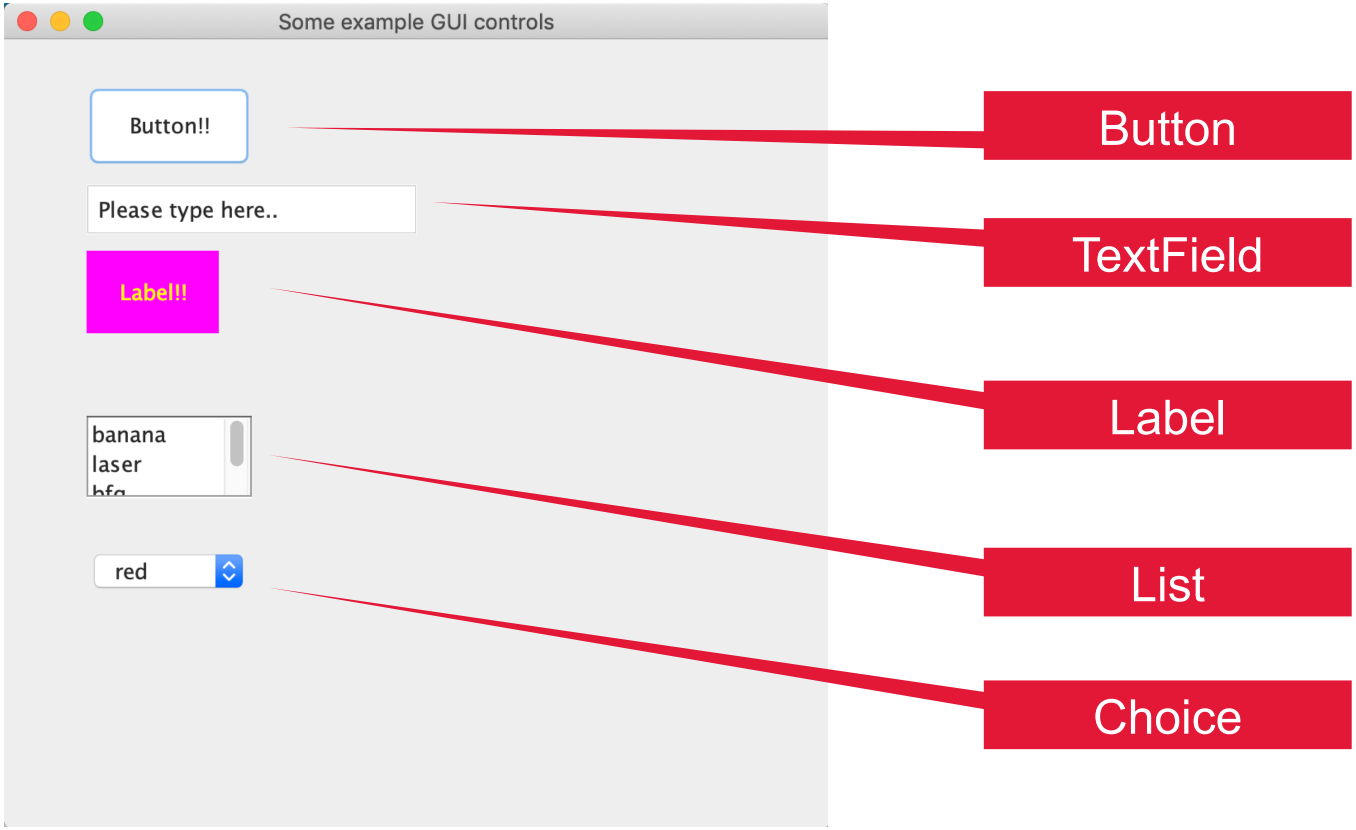
AWT Components - Containers

- A component to hold other components (e.g. text fields, buttons, etc)

- Types:
  - Window
    - Window instance with no border/title
  - Frame
    - Subclass of Window – contains title, border, menu bars
  - Dialog
    - Subclass of Window – contains title and border (always needs another Frame in order to exist – i.e. must be instantiated with respect to an existing Frame)
  - Panel
    - Subclass of Container, no title bar, menu or border (usually used to hold GUI elements inside a window)

GUI Elements

- Button
  - Allows for a trigger for a certain program action by clicking on it
- Text Field
  - Element allowing for text entry
- Label
  - Text used for descriptive purpose only (not user input)
- Canvas
  - Defines a Rectangular area for drawing
- Choice
  - Pop up (or dropdown) menu of choices (can select one)
- Scroll Bar
- List
  - Text list of menu items (can select multiple)
- CheckBox
  - graphical component that is used to create a checkbox. It has two state options; true and false. At any point in time, it can have either of the two.



Please type here..

red

Button

TextField

Label

List

Choice

2 ways to create a GUI

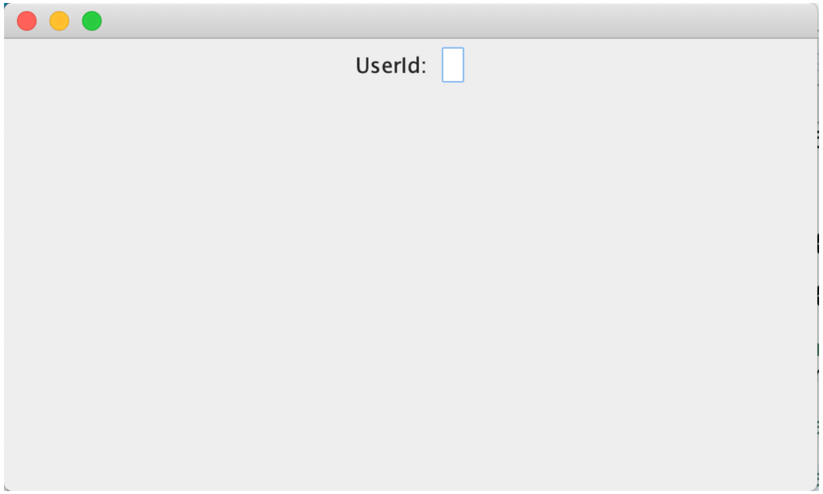
1. Creating an instance of Frame class

2. Extending Frame class

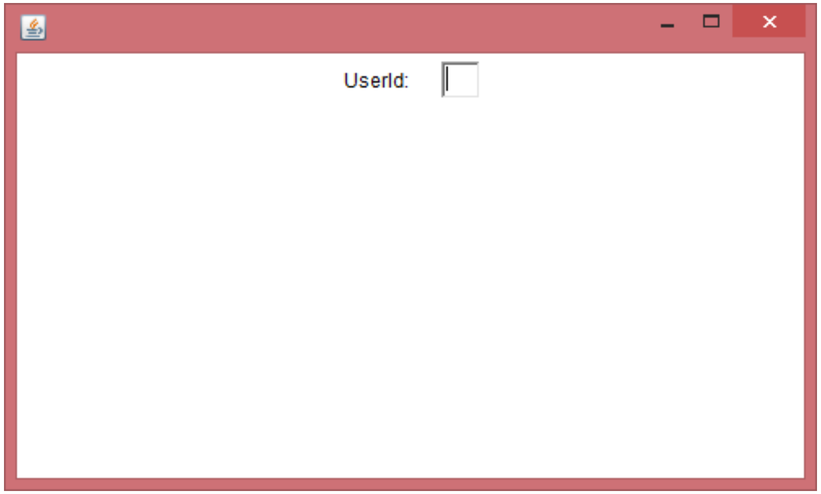
GUI by creating instance of Frame class

```
import java.awt.*;
public class Example1 {

    public Example1()    {
          //Creating Frame
          Frame fr=new Frame();

          //Creating a label
          Label lb = new Label("UserId: ");

          //add some objects to frame
          fr.add(lb);

          //Creating and adding Text Field
          TextField t = new TextField();
          fr.add(t);

          //setting frame size
          fr.setSize(500, 300);

          //Setting the layout for the Frame
          fr.setLayout(new FlowLayout());
          fr.setVisible(true);
     }

   public static void main(String args[])    {

          Example1 ex = new Example1();
   }
}
```

GUI by creating instance of Frame class



UserId:

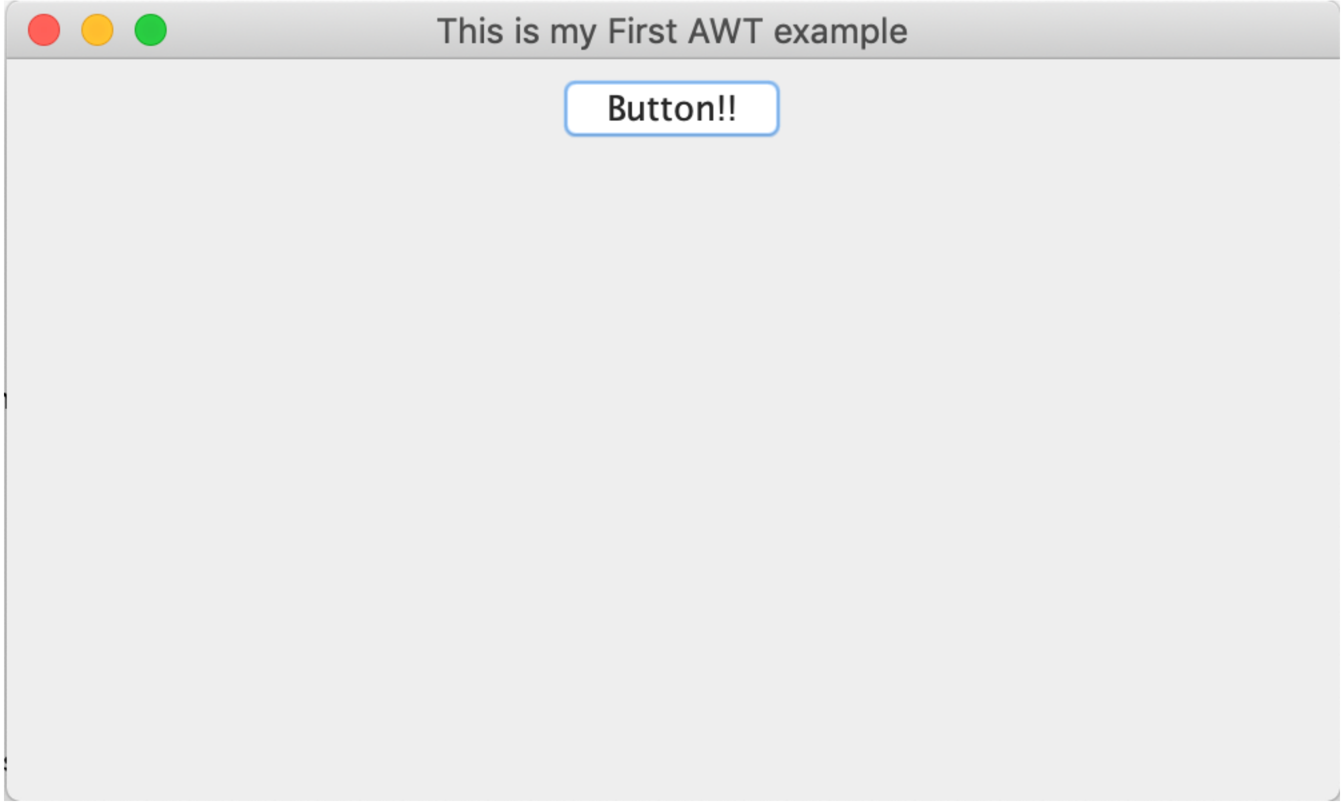


UserId:

GUI by extending Frame class (Inheritance)

```
import java.awt.*;

/* We have extended the Frame class here,  thus our class "SimpleExample"
 * would behave like a Frame  */

public class Example2 extends Frame {

     public Example2() {
         Button b = new Button("Button!!");

         b.setBounds(50,50,50,50);      // setting button position on screen
         this.add(b);                  // adding button into frame

         this.setSize(500,300);                    // Set Frame width and height
         this.setTitle("This is my First AWT example");  // Set title of Frame
         this.setLayout(new FlowLayout());               // Set the layout for the Frame

         /* By default frame is not visible
                              so we are setting the visibility to true*/
         this.setVisible(true);

      }

      public static void main(String args[]) {

           Example2 fr = new Example2();
      }
}
```

GUI by extending Frame class (Inheritance)



Buttons

Constructors:

```
//Construct a Button with the given label
public Button(String btnLabel);

//Construct a Button with empty label
public Button();
```

Some methods:

```
/Get the label of this Button instance
public String getLabel();

//Set the label of this Button instance
public void setLabel(String btnLabel);

//Enable or disable this Button. Disabled Button cannot be clicked
public void setEnable(boolean enable);
```

Text Fields

Constructors:

```
//Construct a TextField instance with the given initial text string with the number of
columns.
public TextField(String initialText, int columns);

//Construct a TextField instance with the given initial text string.
public TextField(String initialText);

//Construct a TextField instance with the number of columns.
public TextField(int columns);
```

Some methods:

```
// Get the current text on this TextField instance
public String getText();

// Set the display text on this TextField instance
public void setText(String strText);

//Set this TextField to editable (read/write) or non-editable (read-only)
public void setEditable(boolean editable);
```

Labels

provides a descriptive text string that is visible on GUI

Constructors:

```
// Construct a Label with the given text String, of the text alignment
public Label(String strLabel, int alignment);

//Construct a Label with the given text String
public Label(String strLabel);

//Construct an initially empty Label
public Label();
```

```
Class also provides 3 constants:
public static final LEFT;      // Label.LEFT
public static final RIGHT;     // Label.RIGHT
public static final CENTER;    // Label.CENTER
```

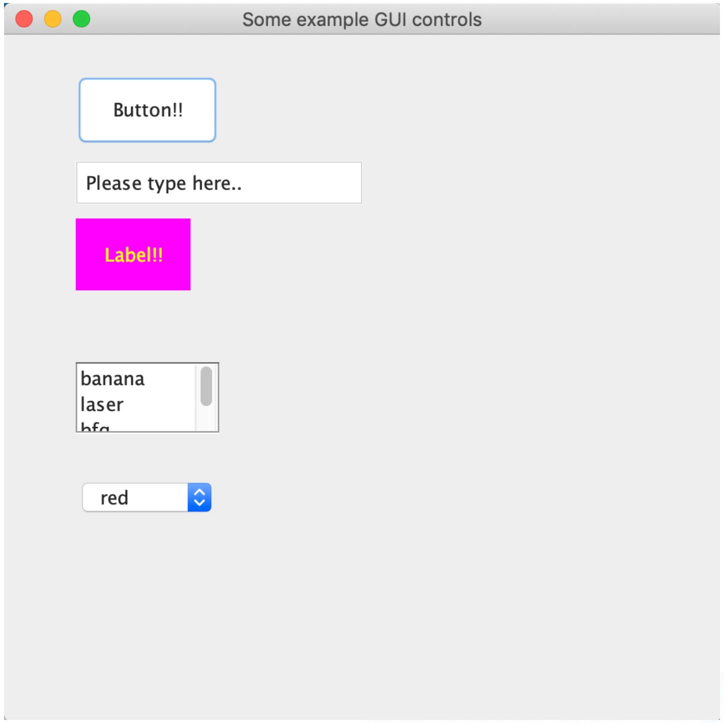
Some methods:

```
public String getText();
public void setText(String strLabel);
public int getAlignment();

//Label.LEFT, Label.RIGHT, Label.CENTER
public void setAlignment(int alignment);
```

Example 2b (different GUI controls)

```
import java.awt.*;

public class Example2b extends Frame {

      public Example2b() {

            this.setSize(500,500); // Set Frame width and height
            this.setTitle("Some example GUI controls"); // Set title of Frame
            this.setLayout(null); // Set the layout for the Frame

            // example GUI controls (positioned and added to Frame)
            Button b = new Button("Button!!");
            b.setBounds(50,50,100,50);                      // setting button position on
            screen
            this.add(b);                                    // adding button into frame

            TextField t = new TextField("Please type here..");
            t.setBounds(50,110,200,30);
            this.add(t);

            Label l = new Label("Label!!", Label.CENTER);
            l.setBounds(50,150,80,50);
            l.setBackground(Color.magenta);
            l.setForeground(Color.yellow);
            this.add(l);


            // ... (next slide)
```

```
    // ...

          List items = new List(2,false);
          items.add("banana");
          items.add("laser");
          items.add("bfg");
          items.add("peanut");
          items.setBounds(50,250,100,50);
          this.add(items);

          Choice c = new Choice();
          c.add("red");
          c.add("green");
          c.add("blue");
          c.setBounds(50, 320, 100, 50);
          this.add(c);


          this.setVisible(true);
    }


    public static void main(String args[]) {

          Example2b fr = new Example2b();

    }
}
```

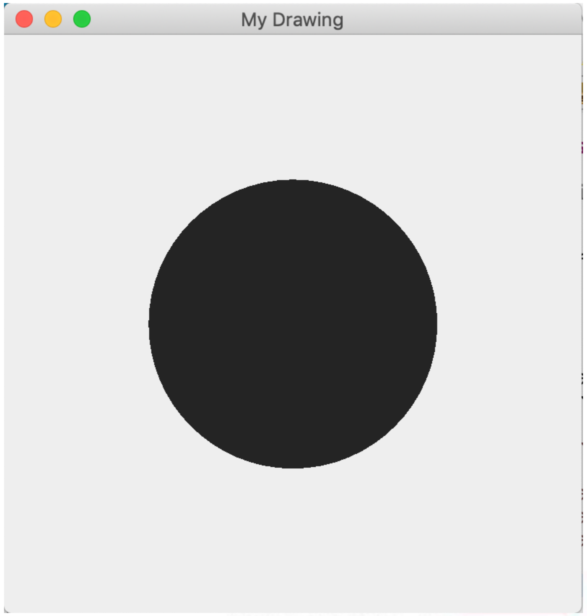


Certain subclasses within Java AWT may be **extended** to develop a unique application

- The application may extend so that it can easily override some of the built-in methods of that class:
  - Example: Extend from **Canvas** class so that a method that automatically paints objects added to the canvas may be overridden

*** we will learn more about the details of methods we can override later*

```
import java.awt.Canvas;
import java.awt.Graphics;
import java.awt.Frame;


public class Example3 extends Canvas {

     // default ctor is implicit

     @Override
     public void paint(Graphics g) {
          g.fillOval(100, 100, 200, 200);
     }

     public static void main(String[] args) {

          Frame frame = new Frame("My Drawing");

          Canvas canvas = new Example3();
          canvas.setSize(400, 400);

          frame.add(canvas);
          frame.pack();
          frame.setVisible(true);

     }
}
```



Key methods in GUI applications..

`repaint(..)`

Method is overloaded (many versions):

the *repaint()* method requests that a component be repainted.

The caller may request that repainting occur as soon as possible, or may specify a period of time in milliseconds.

If a period of time is specified, the painting operation will occur before the period of time elapses.

The caller may also specify that only a portion of a component be repainted. This technique is useful if the paint operation is time-consuming, and only a portion of the display needs repainting.

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

The *update()* method is called in response to a *repaint()* request, or in response to a portion of the component being uncovered or displayed for the first time.

The default implementation provided by the Component class erases the background and calls the *paint()* method

Java AWT/Swing

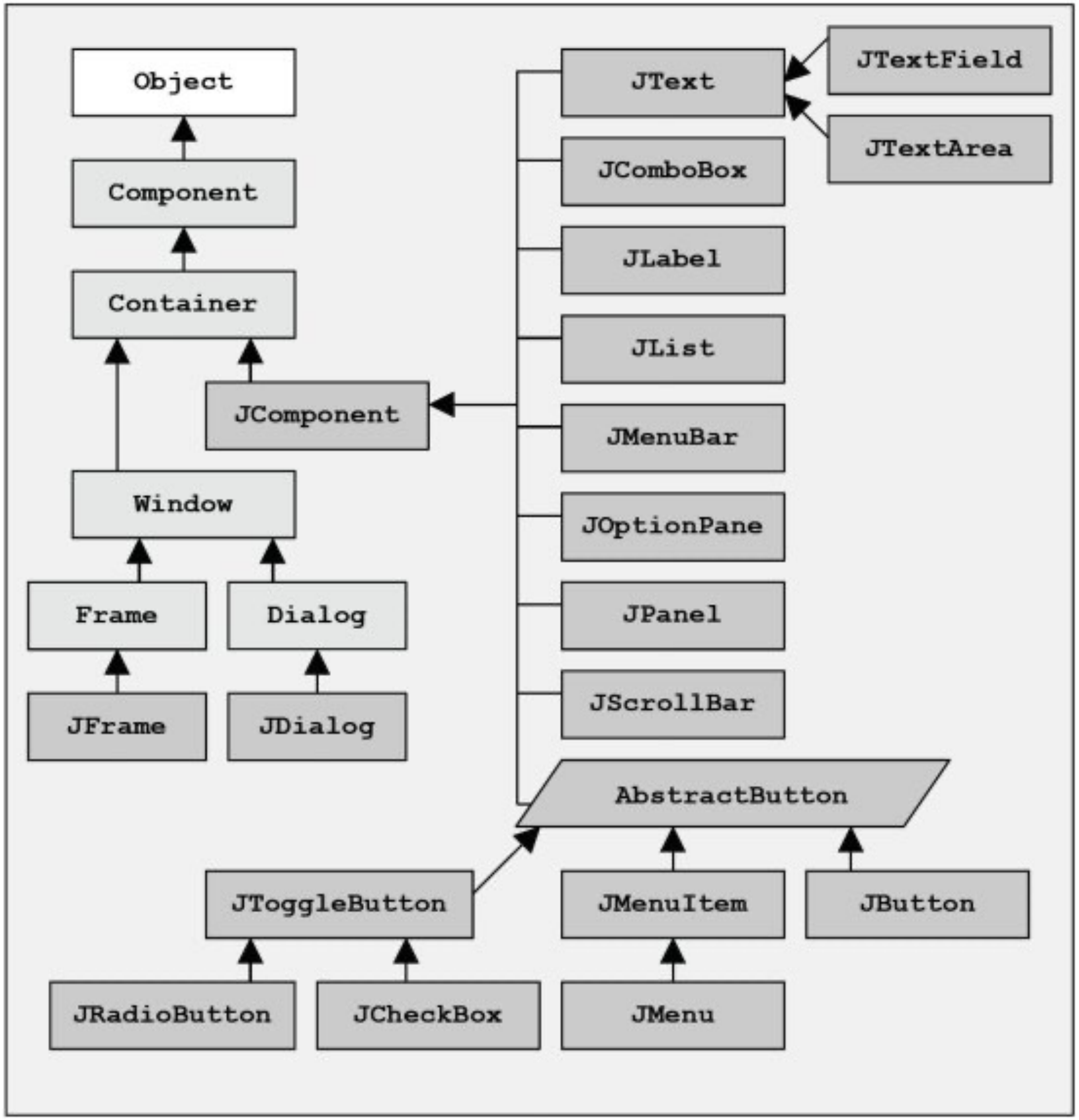
AWT/Swing applications?

- **AWT is platform dependent and heavy-weight nature.**
  - AWT components are considered heavy weight because they are being generated by underlying operating system (OS). For example if you are instantiating a text box in AWT that means you are actually asking OS to create a text box for you.

- **Swing is a preferred API for window based applications because of its platform independent and light-weight nature.**
  - Swing is built upon AWT API however it provides a look and feel unrelated to the underlying platform.
  - It has more powerful and flexible components than AWT.
  - In addition to familiar components such as buttons, check boxes and labels, **Swing provides several advanced components such as tabbed panel, scroll panes, trees, tables, and lists.**

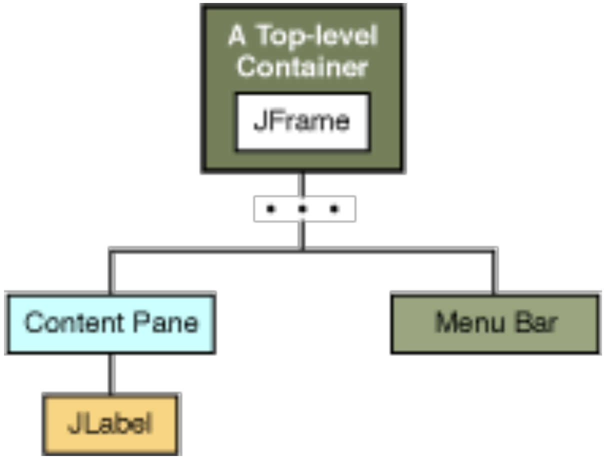
Swing (w.r.t. AWT)

- Platform independent
- Lightweight
- Builds on AWT
- More flexible than AWT



Swing applications

- Everything in a Swing app is contained within a top-level container (usually JFrame)
- Components added to the application form a "containment hierarchy" with the top-level container as the root (like a directory tree)
- The top-level container has a content pane that contains all GUI components – either directly (within top-level container) or indirectly (within sub-containers)
- A GUI component may only be added to a single container (adding to a second container will switch it to the new container)
- Top-level container can also have a menu bar (positioned in container, but separate/outside of content pane)

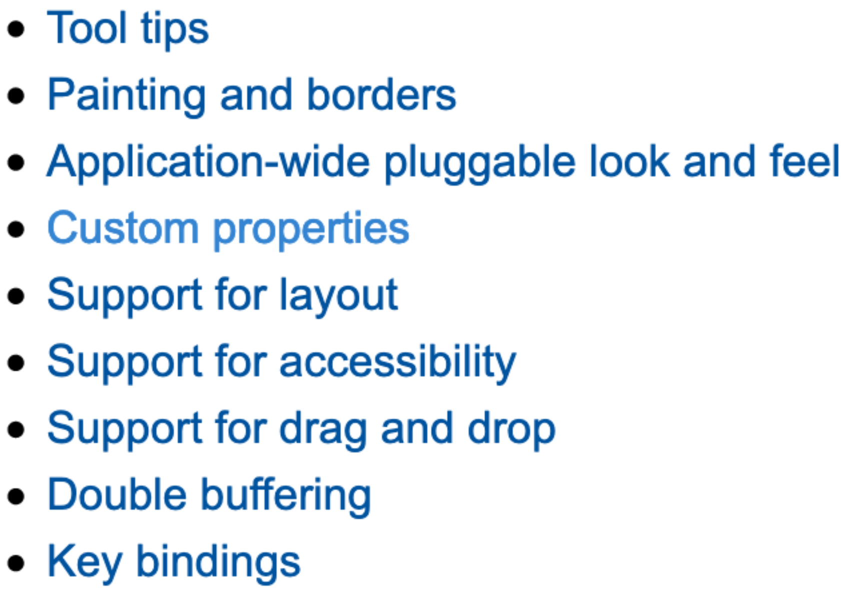


top-level containers:

- *JFrame*
- *JDialog*
- *JComponent*

JComponent

- Provides following functionality:



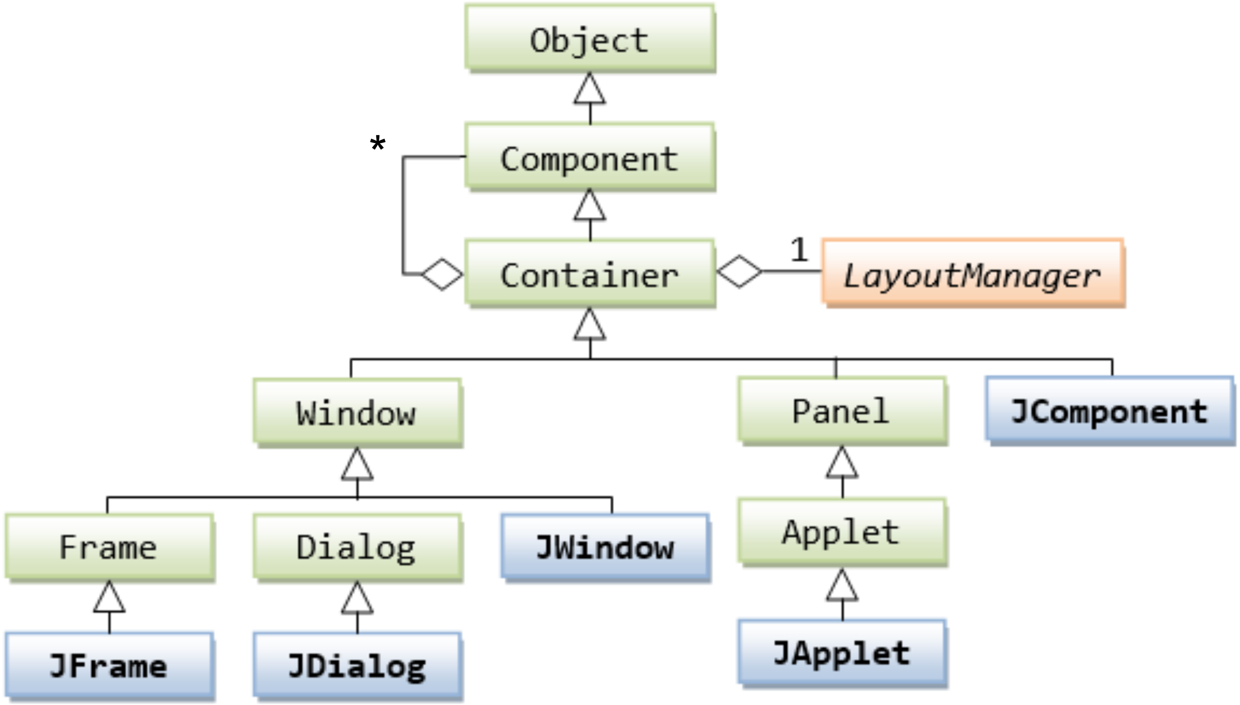
- Tool tips
- Painting and borders
- Application-wide pluggable look and feel
- Custom properties
- Support for layout
- Support for accessibility
- Support for drag and drop
- Double buffering
- Key bindings

JComponent API:
https://docs.oracle.com/javase/tutorial/uiswing/components/jcomponent.html#containmentapi

JComponents ~ Swing version of Components in AWT

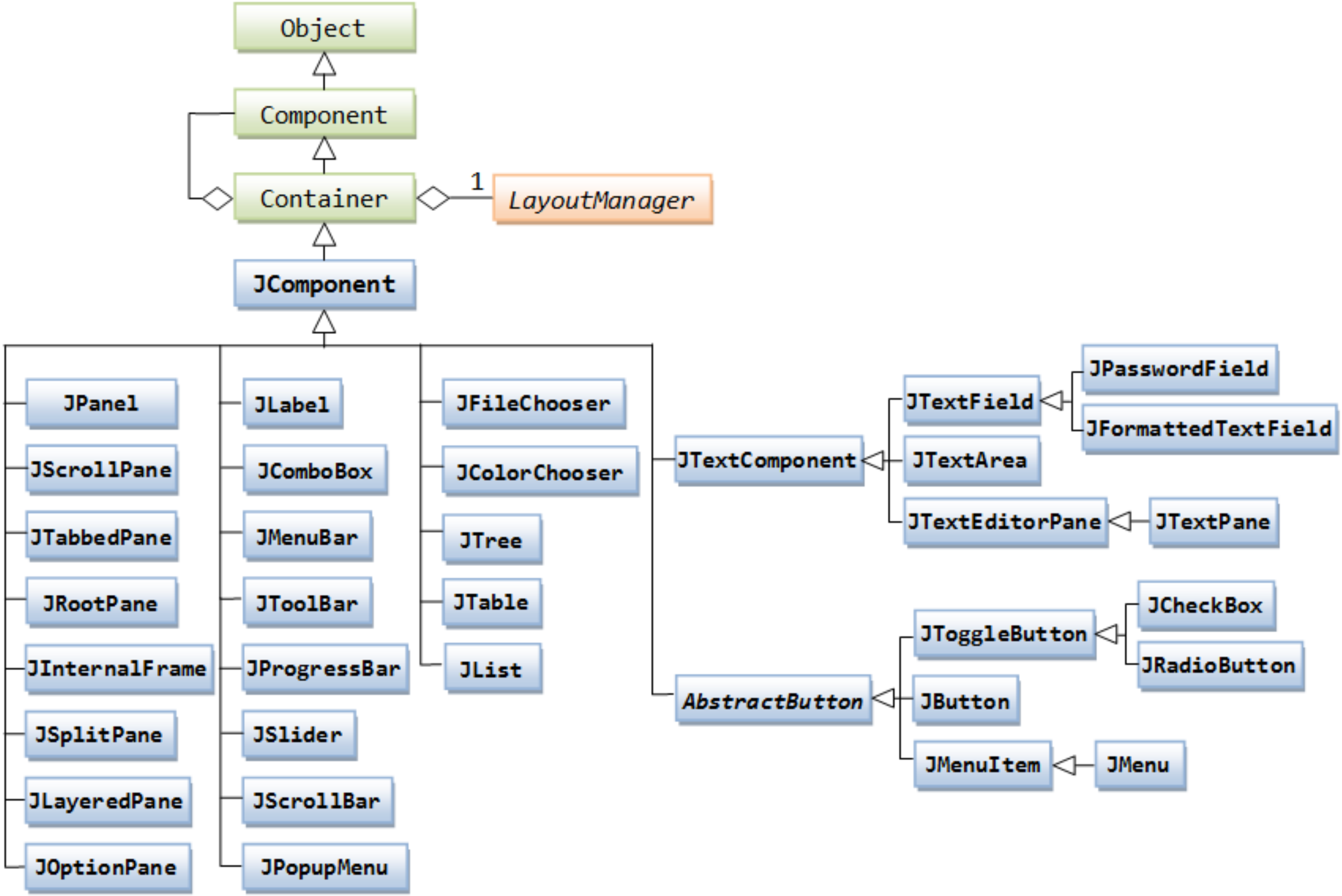
AWT	Swing
Component	JComponent
Frame	JFrame
Dialog	JDialog
Window	JWindow
Panel	JPanel
Button	JButton
Label	JLabel
CheckBox	JCheckBox
Choice	JRadioButton
Text	JText
ScrollBar	JScrollBar
List	JList
	+ many more !

Simplified class hierarchy (Swing):



Container **is-a** Component
Container **has-a** LayoutManager
Container **has-a** Component (in fact can have many)
JFrame, JDialog, JComponent **is-a** Container

Class hierarchy (Swing)



Simple Swing template:

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

public class Example4 extends JFrame {

      // similar to AWT inheritance example (analogous to extending Frame)

      public Example4(String name) {            // constructor creates GUI
            super(name);                        // create parent sub-object
            this.setLayout(null);               // set to not use any layout manager

            JButton button = new JButton("Press me!");
            button.setBounds(100, 50, 100, 50);             // position/size button

            JLabel heading = new JLabel("Hello");
            heading.setBounds(100, 150, 100, 50);           // position/size label

            this.add(button);                               // adds JComponents to content pane
            this.add(heading);

            // setup JFrame object for display
            this.setSize(480,400);                              // frame size 480 width and 400 height
            this.setResizable(true);                            // allow/restrict window resizing
            this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);// close on 'x'
            this.setVisible(true);                              // make frame visible
      }

      public static void main(String[] args) {
            // Create GUI layout
            Example4 frame = new Example4("Hello Swing");
      }
}
```

More to come..