CS 2113
Software Engineering

Lecture 10: GUIs

```
 Import the code to intelliJ
https://github.com/cs2113f16/lec-10-guis.git
```

Professor Tim Wood - The George Washington University

Last Time...

- Class Hierarchies
- Abstract Classes
- Interfaces

This Time

- Hash maps and sorting
- GUIs in Java
  - AWT vs Swing
  - Swing Basics
- Zombies

Student Roster

- Need a way to find and print a student's data based on their ID number
  - What data structure can help us with that?
- Need a way to sort a list of students by name?
  - Do we need to implement bubble sort ourselves?
- Go back and reread chapter 16!

HashMap

- A type of **`Map`** - stores {**Key, Value**} pairs
  - Lookup a **Value** object by presenting a **Key** object
- Need to be careful if value is an int, float, or double
  - These are basic types in Java, not objects!
  - Need to use Integer, Float, or Double as type

```
HashMap<String, Integer> hmap = new HashMap<String, Integer>();
hmap.put("Rahul", 20);
hmap.put("Chen", 15);
int c = hmap.get("Chen");
c++
hmap.put("Chen", c); // replaces old value
```

- Example usage:
  - http://beginnersbook.com/2013/12/hashmap-in-java-with-example/

Interfaces for Sorting

- Sorting is a very common operation
- How do you sort:
  - Numbers
  - Letters
  - Names
  - Animals
  - Customers
- Basic operation in any sorting algorithm:
  - Is element **A** higher or lower than element **B**?

Comparable Interface

- Implement the **`Comparable`** Interface to define how to compare instances of a class
- Allows you to use a generic sorting function

```
List<Name> names = new ArrayList<Name>();

// add elements to list

Collections.sort(names);
// list is magically sorted!
```

- Must implement the CompareTo(b) function
  - Return 0 if identical
  - Less than 0 if `this < b` or greater than 0 if `this > b`

Roster Lab

- Get the code in intelliJ **https://github.com/cs2113f16/lec-10-guis.git**
- ALWAYS use the VCS->Checkout From Version Control menu option!
- Let's solve it together...

What is a GUI library?

- A way to:
  - Open windows
  - Display **widgets** on screen
  - Process **events**
- Widgets:
  - Buttons, images, Menu bars, tabs, popups, etc
- Events:
  - Mouse clicks, keyboard interactions, windows being moved/resized/minimized/closed, etc

GUIs in Java

- Two main approaches:
- **A**bstract **W**indow **T**oolkit (AWT)
  - Java library to interact with the OS's **native** graphical interface tools
- Swing
  - Interface library relying (almost) purely on Java
- Pros and Cons?

Swing vs AWT

- Code is similar:

```
import java.awt.*;

public class TestAwt1 {
  public static void main (String[] argv)
  {
    Frame f = new Frame ();
    f.setSize (200, 100);
    f.setVisible (true);
  }
}
```

```
import javax.swing.*;

public class TestSwing1 {
  public static void main (String[] argv)
  {
    JFrame f = new JFrame ();
    f.setSize (200, 100);
    f.setVisible (true);
  }
}
```

Swing vs AWT

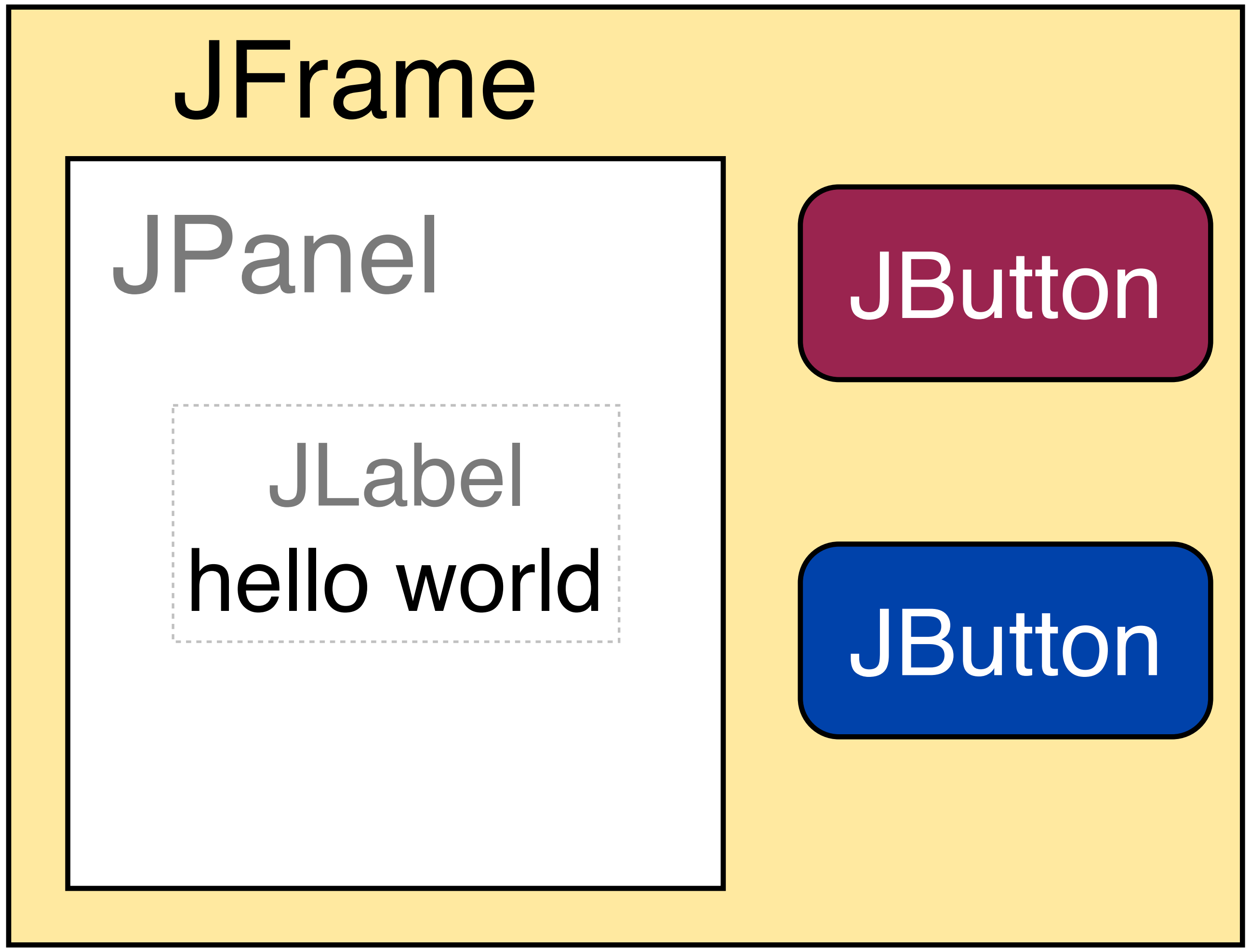
- AWT relies on **native** libraries to draw graphics
  - A java program written with AWT would look different depending on the OS
  - Less control within AWT library since OS does most work

- Swing:
  - Gives a consistent look across platforms
  - Supports a wider range of widgets
  - More customizable

- We will use Swing

GUIs are made up of:

- Containers
  - Holds other widgets
- Components
  - A widget to interact with or display something
- Common examples:
  - Frame: basic window
  - Panel: an area to group other objects or draw images/art
  - TextField/TextArea: allows text input
  - Simple widgets: Checkbox, List Button, Label, Scrollbar and Scrollpane.
- Swing widget classes all start with "J"



Our First Window

- Is this code enough?

```
import javax.swing.*;

public class TestSwing1 {
  public static void main (String[] argv)
  {
    JFrame f = new JFrame ();
  }
}
```

Our First Window

- Is this code enough?

```
import javax.swing.*;

public class TestSwing1 {
  public static void main (String[] argv)
  {
    JFrame f = new JFrame ();
  }
}
```

- Nope!
- Also need to:
  - Give the window a size and make itself visible

Open a Window

- Get the code for today from the class site
- Look at the **guis.HelloSwing.java** file
  - What happens when you run it?
  - What happens when you try to close the window?
- Can you figure out how to set the title of the window to "Hello World"?

```
More fun:
 - have the window appear at a specific location
 - open five windows instead of one
```

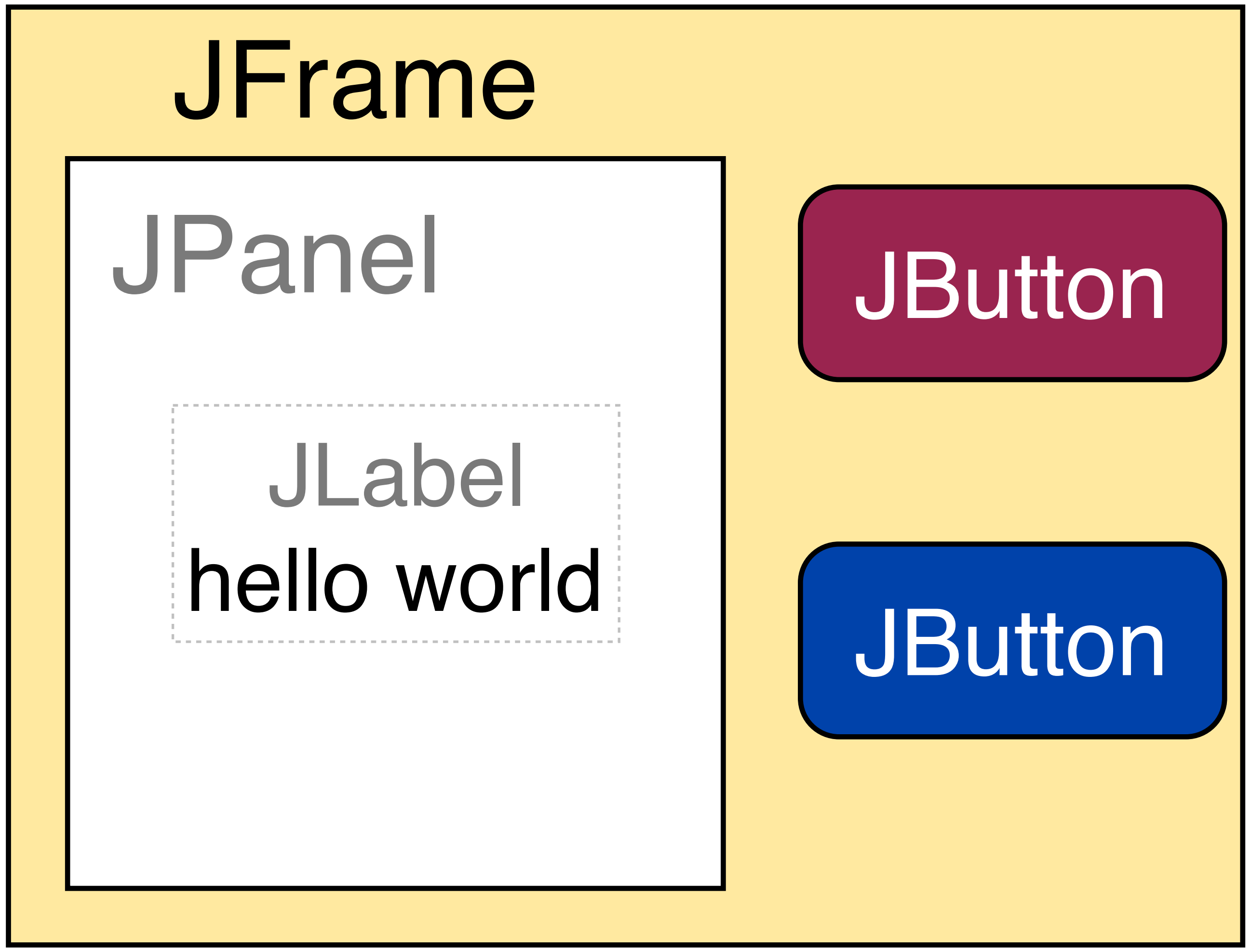
We haven't said "hello" in a while...

- What if we want to draw text or simple line art in our program?

- We can draw on a component by getting access to a **graphics context**
  - An instance of the **Graphics** class
  - Has methods for drawing lines/points/shapes
  - The functions we used from StdDraw were all calling methods of a graphics object

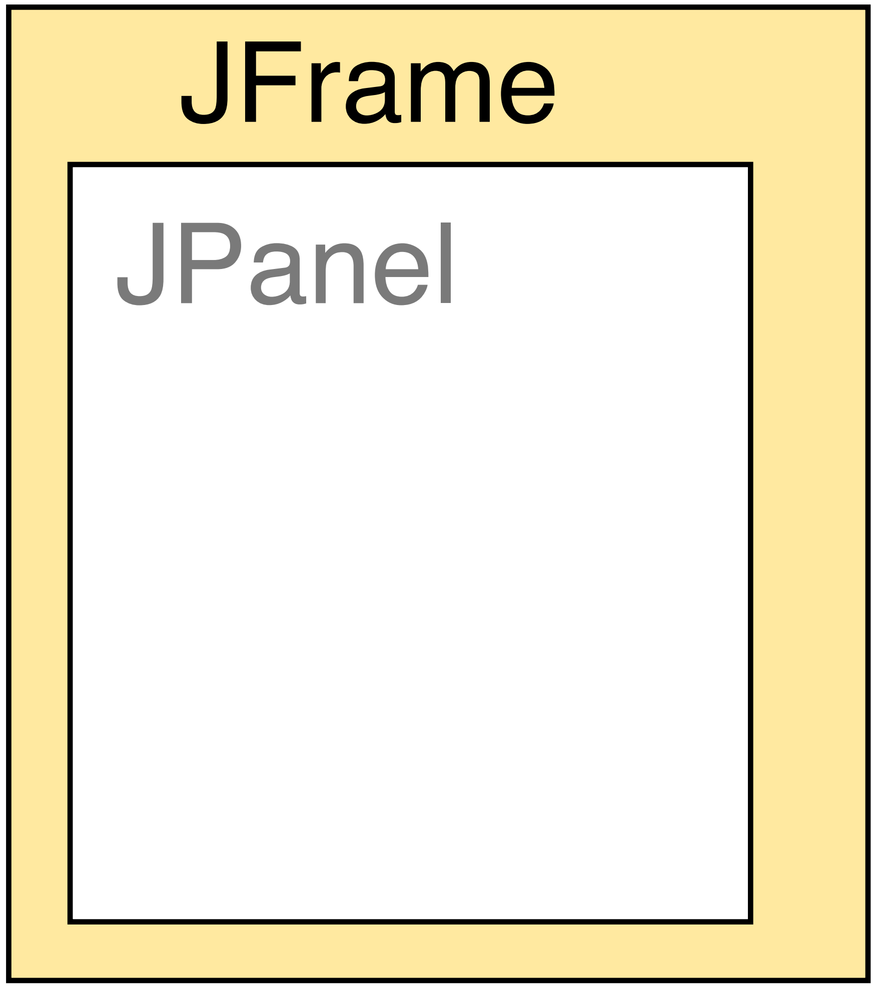
Content Pane

- A JFrame has a **ContentPane** to hold widgets
  - but the contentPane by itself doesn't know how to draw...
- Need the graphics context of a **JPanel**
  - Can add a JPanel to the JFrame's ContentPane
  - Then we can draw / add more objects



How a GUI draws itself

- When a widget needs to display itself it must call:
- **paintComponent(Graphics g)**



- A look at JPanel's family tree:

```
java.lang.Object
  java.awt.Component
      java.awt.Container
          javax.swing.JComponent
              javax.swing.JPanel
```

- The `javax.swing.JComponent` class defines **paintComponent(Graphics g)**

We must extend JPanel

- To create your own **paintComponent()** method, must **extend JPanel** in a custom class
- Gives you direct access to the Graphics object
  - Can draw, change colors, go nuts
  - **`g.drawString ("Hello World!", 100, 100)`**

`Graphics`: Drawing Shapes

- **drawRect**(int topleftx, int toplefty, int width, int height):
  - The first two integers specify the topleft corner.
  - The next two are the desired width and height of the rectangle.
- **drawOval**(int topleftx, int toplefty, int width, int height):
  - The first two integers specify the topleft corner.
  - The next two are the desired width and height of the enclosing rectangle.
- Also have filledRect and filledOval equivalents
- **drawLine**(int x1, int y1, int x2, int y2):
  - Unfortunately, the line thickness is fixed at one pixel.
  - To draw thicker lines, you have to "pack" one-pixel lines together yourself.

Draw me a picture

- Draw a pretty picture
  - Edit the **guis.PrettyPicture.java** file

- **drawRect**(int topleftx, int toplefty, int width, int height):
  - The first two integers specify the topleft corner.
  - The next two are the desired width and height of the rectangle.
- **drawOval**(int topleftx, int toplefty, int width, int height):
  - The first two integers specify the topleft corner.
  - The next two are the desired width and height of the enclosing rectangle.
- Also have filledRect and filledOval equivalents
- **drawLine**(int x1, int y1, int x2, int y2):
  - Unfortunately, the line thickness is fixed at one pixel.
  - To draw thicker lines, you have to "pack" one-pixel lines together yourself.

How it Works

- We create a JFrame (window)
- We fill the JFrame with a JPanel (or a child of JPanel)
- JFrames call **paintComponent** on every component inside them
- Our custom **paintComponent()** method draws our perty picture

Drawing

- When is **paintComponent()** called?

Another way to say "hello"

- It doesn't always make sense to use drawString()
  - Low level function
  - What if we want to change the text dynamically?
  - Does not feel very "object oriented"

- Can also use the **JLabel** component

```
Container cPane = f.getContentPane();
JLabel helloLabel = new JLabel("Hello!");
cPane.add(helloLabel);
```

- Gives us an object to store a message
- Add it to a panel/frame and it will be drawn automagically!

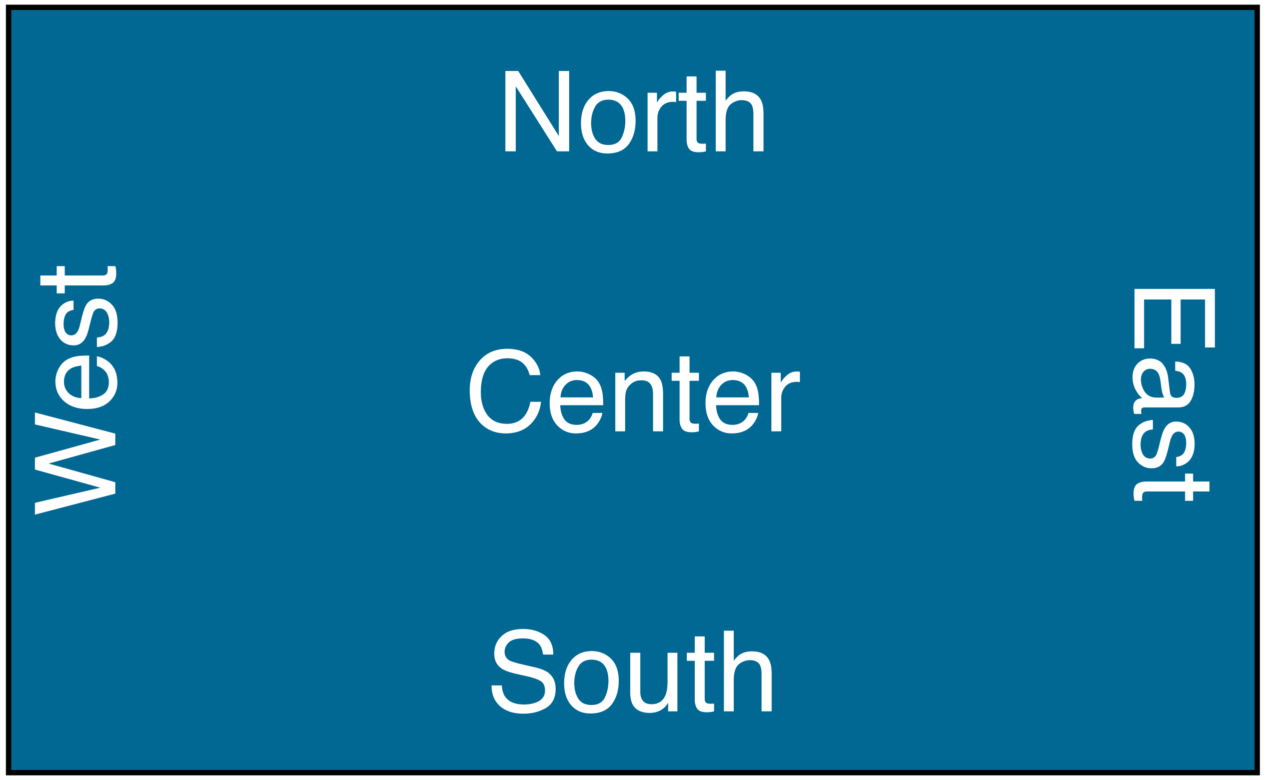
JLabel

- Try out **guis.HelloSwing2.java**
- At a low level, how do you think JLabel works?
- Where is the message displayed?
- What happens if you create another JLabel and add it to the frame as well?

Java Layout Managers

- Swing (and AWT) use Layout Managers to control where components are placed
  - You (generally) do not have precise control over placement
  - Simplifies automated GUI creation
  - Makes hand designed GUIs trickier

- Default layout manager: BorderLayout



JLabel take two

- You can specify (approximately) where to add a component with:

```
cPane.add(helloLabel, BorderLayout.WEST);
// or .EAST, NORTH, SOUTH, CENTER
```

- Add a second JLabel so that it does NOT replace the first one

```
More fun:
 - go back to PrettyPicture.java and make it add
 three PrettyPanels to the window next to each
 other
```

More Layouts

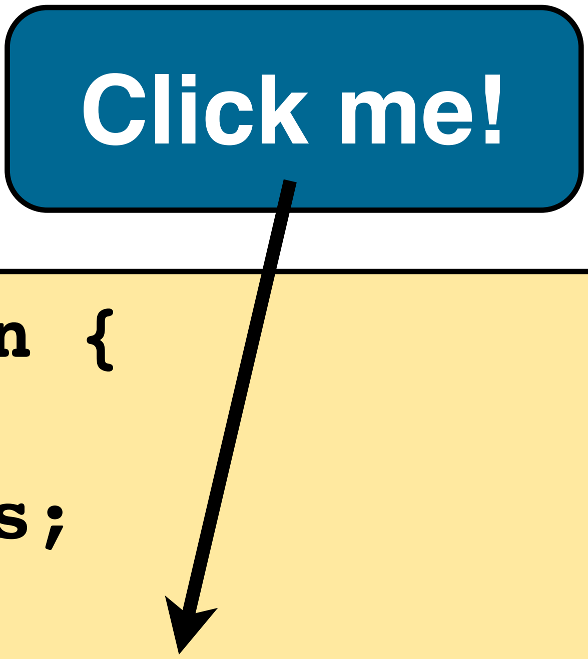
- Commonly used layouts managers:
  - **BorderLayout**: tries to place components either in one of five locations: North, South, East, West or Center (default).
  - **FlowLayout**: places components left to right and row-by-row.
  - **CardLayout**: displays only one component at a time, like a rolodex.
  - **GridLayout**: places components in a grid.
  - **GridBagLayout**: uses a grid-like approach that allows for different row and column sizes.
- Change a container's layout with:

```
Container cPane = f.getContentPane();
cPane.setLayout(new FlowLayout());
```

Events and Listeners

- Clicking a button is an event
- What happens if a tree falls in a forest and nobody is there to hear it?
    - Same idea with buttons
- How do you think buttons should work code-wise?

Inside a Button



```
public class JButton extends AbstractButton {

private ArrayList<ActionListener> listeners;

protected void fireActionPerformed(ActionEvent event) {
  for(ActionListener al: listeners) {
     al.actionPerformed(event);
  }

protected addActionListener(ActionListener L) {
  listeners.add(L);
}
```

What's an ActionListener?

- It's just an Interface!

http://download.oracle.com/javase/1.4.2/docs/api/java/awt/event/ActionListener.html

- Only requires one method:
  - actionPerformed(ActionEvent e)

Button Events

- Something must implement ActionListener

- One option: have the JFrame do it

```
class NewFrame extends JFrame implements ActionListener {
  public NewFrame (int width, int height)
  {
   // ...
   button.addActionListener(this);
   // ...
  }
  public void actionPerformed (ActionEvent a)
  {
    System.out.println ("ActionPerformed!");
  }
```

Mouse/Keyboard Interfaces

- Sometimes you want to detect keyboard and mouse events other than interactions with buttons

- **MouseListener**

```
public void mouseClicked(MouseEvent m);

public void mouseEntered(MouseEvent m);

public void mouseExited(MouseEvent m);

public void mousePressed(MouseEvent m);

public void mouseReleased(MouseEvent m);
```

- **KeyListener**

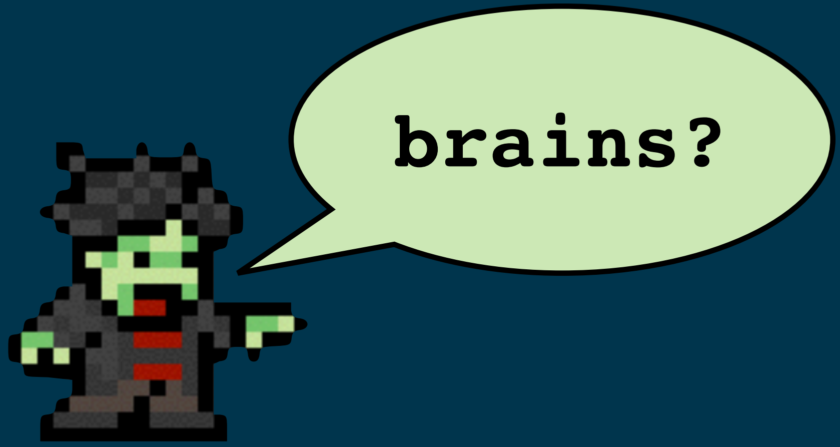
```
public void keyTyped(java.awt.event.KeyEvent arg0);

public void keyPressed(java.awt.event.KeyEvent arg0);

public void keyReleased(java.awt.event.KeyEvent arg0);
```

Project 2...

Zombie Infestation Simulator



Zombie Sim Structure

- ZombieSim
  - main()
  - instantiates city
  - loop: update city and draw
- City
  - private Walls[][]
  - update
  - draw
  - populate()
  - what else to add???

Tips/Best Practices:

- Think carefully about class structure and the data and functions in each one
- Think carefully about the "is a" versus "has a" relationship when designing your classes
- It is better to have a class interact with another using an API (functions) instead of directly accessing data
- Use classes to encapsulate both data and functions. A City class should be responsible for everything to do with the city and a Cat class would be responsible for everything to do with cats, etc.