CSSE 220

Event Based Programming

Check out *EventBasedProgramming* from SVN

Interfaces - Review

- Interfaces are contracts
  - Any class that ***implements*** an interface **<u>MUST</u>** provide an implementation for all methods defined in the interface.
- Interfaces represent the abstract idea (and what it can do):
  - Measurable objects (return a measure)
  - NumberSequences (get the next number, reset)
  - Pet (Can be fed, can tell if eating, can tell name)
- Classes represent the concrete idea:
  - Country, Bank Account
  - AddOne, PowersOfTwo.
  - Dog, Cat, Fish

Polymorphism! (A quick intro)

- Etymology:
  - Poly → many
  - Morphism → shape

- Polymorphism means: An **Interface** can take **many shapes**.
  - A Pet variable could actually contain a Cat, Dog, or Fish

Polymorphic method calls

- pet.feed() **could** call:
  - Dog's feed()
  - Cat's feed()
  - Fish's feed()

- Your code is well designed if:
  - You **don't need to know** which implementation is used.
  - The end result is the same. ("pet is fed")

Q5

Interfaces – Review (continued)

- The specific method to use at runtime is decided by late-binding

```
Sequence sequence = new PowersOfTwo();
System.out.println(sequence.next());
```

The *declared type* of operation is **Sequence**

The *instantiation type* is **PowersOfTwo**

At runtime, Java will use the method implementation of next() from the **PowersOfTwo** class, thanks to late-binding.

Finish the sentence

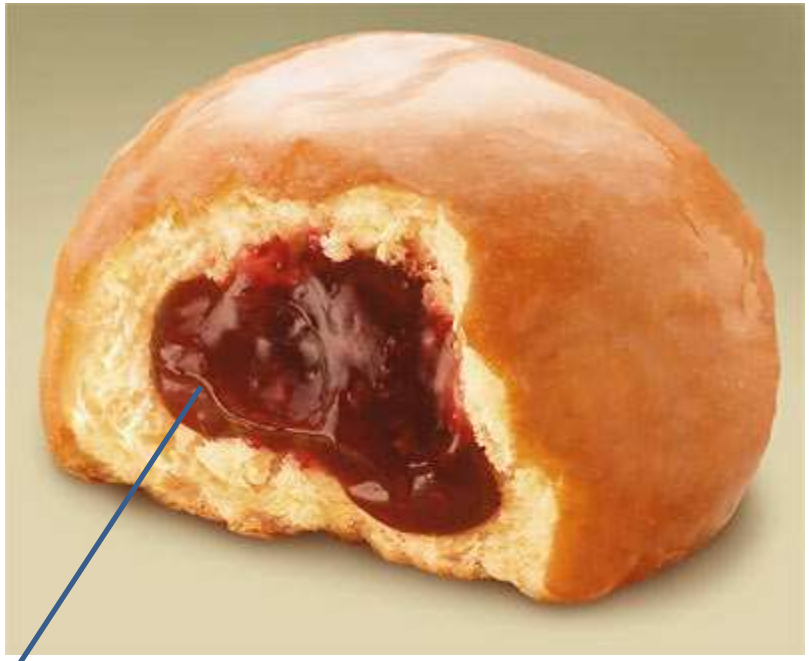
Using interfaces can help reduce _____ between classes.

1. Coupling
2. Cohesion
3. Encapsulation
4. Polymorphism

We need interfaces for event-based programming in Java.

Graphical User Interfaces in Java

- We say what to draw
- Java windowing library:
  - Draws it
  - Gets user input
  - Calls back to us with events
- We handle events



Hmm, donuts

Gooey

Next Assignment Preview

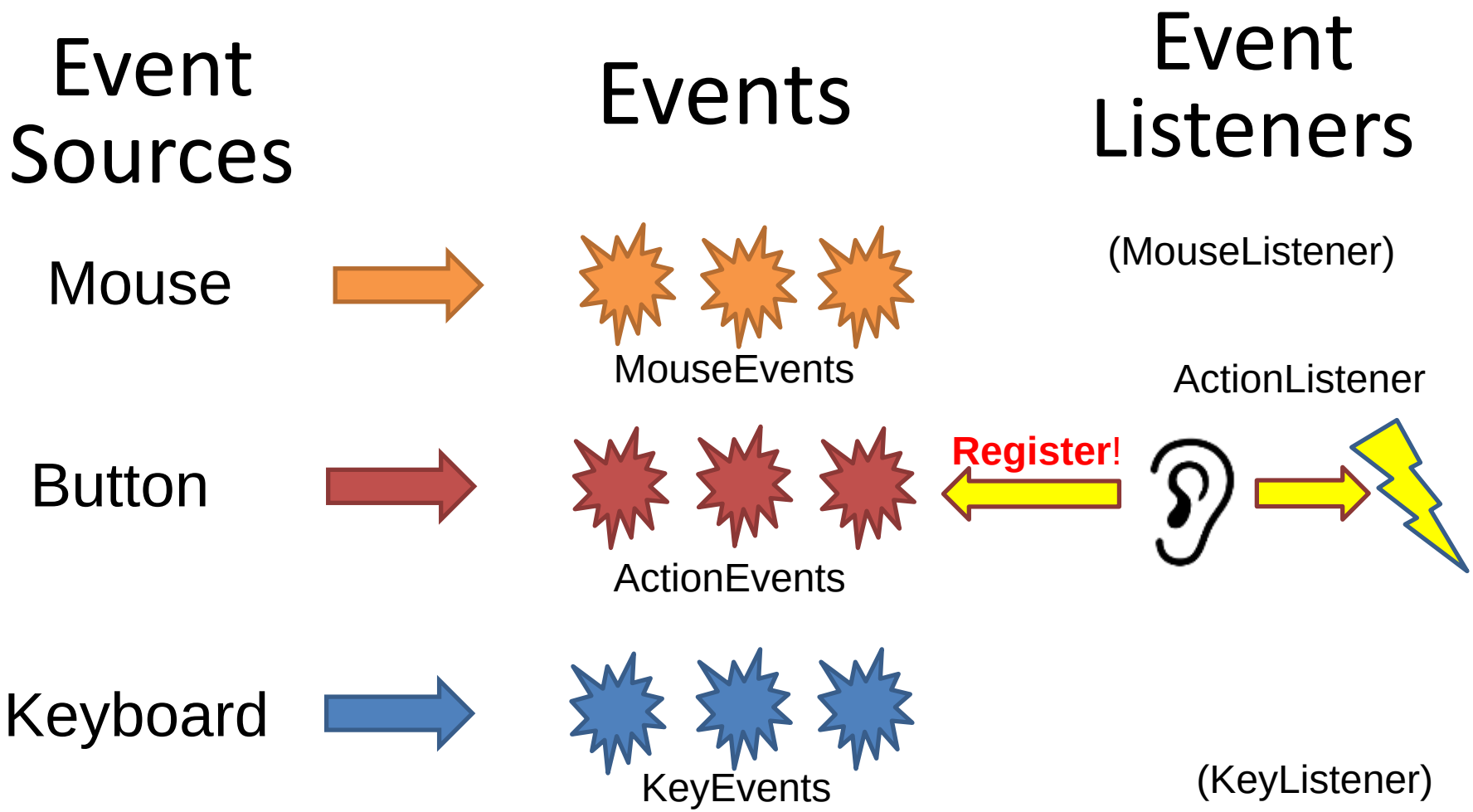
- Two stages
  - Part 1: Ball Strike Counter (individual)
  - Part 2: Optionally work with 1 partner
    - Each list the other's name in javadoc at top of file
    - Both responsible for submitting own code

Handling Events

- Many kinds of events:
  - Mouse pressed, mouse released, mouse moved, mouse clicked, button clicked, key pressed, menu item selected, …
- We create **event listener objects**
  - that implement the right **interface**
  - that handle the event as we wish

- We **register** our listener with an **event source**
  - Sources: buttons, menu items, graphics area, …

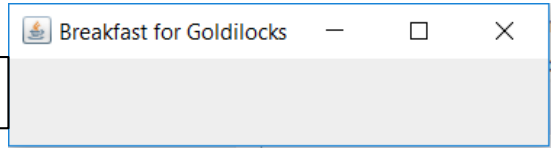
Q1


Event Sources
Events
Event Listeners
Mouse
MouseEvents
(MouseListener)
ActionListener
Button
Register!
ActionEvents
Keyboard
KeyEvents
(KeyListener)

Simple Interactive GUI Workflow

1. Create JFrame *(Needs additional configuration)*

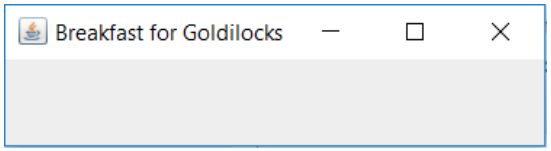
```
JFrame frame = new JFrame("Breakfast for Goldilocks");
```



2. Create JButton

(JButton initially untethered and invisible)

```
JButton button = new JButton("Eat Porridge");
```



3. Add JButton to JFrame (Can also be added to a JPanel)

```
frame.add( button );
```

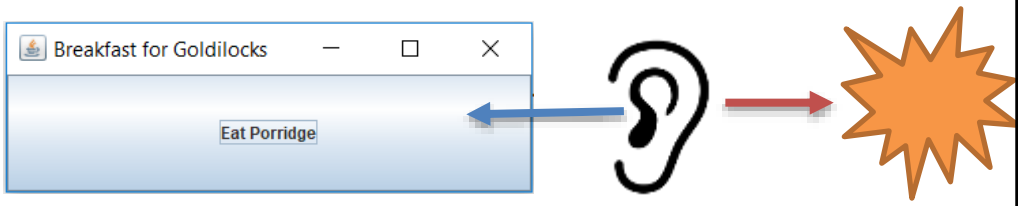
4. Create ActionListener (must code what it does)

(Not connected to JButton, does nothing!)

```
ActionListener ear = new MyListener();
```

5. Attach ActionListener to JButton

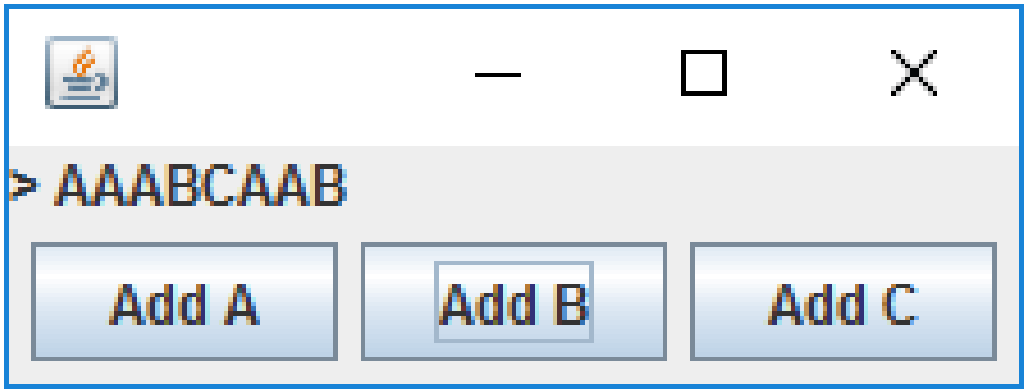
```
button.addActionListener( ear );
```



Live Coding

In Class Activity 1

- In pairs or individually
- Look at the code in the capitalization example
- Then solve the addLettersProblem
- Get buttons and text to show up FIRST!

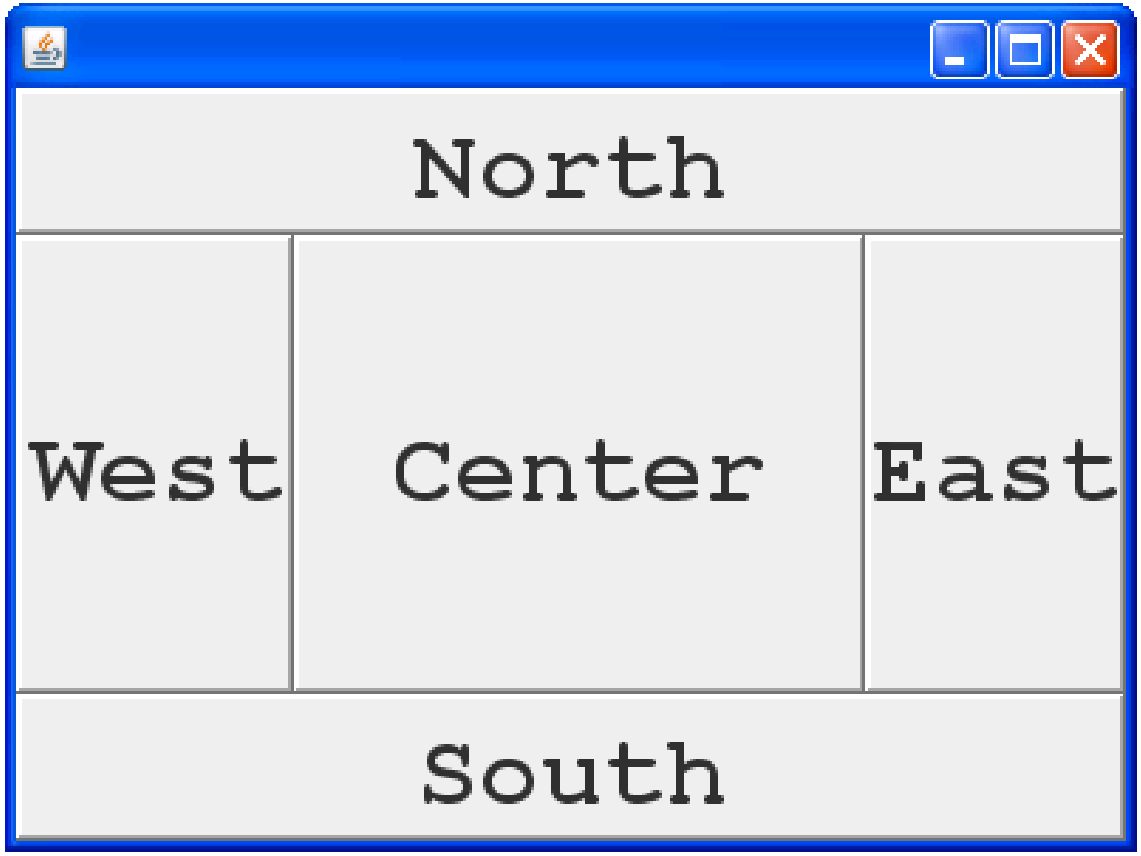


Key Layout Ideas

- JFrame's add(Component c) method
  - Adds a new component to be drawn
  - Throws out the old one!
- JFrame also has method add(Component c, Object constraint)
  - Typical constraints:
    - BorderLayout.NORTH, BorderLayout.CENTER
  - Can add one thing to each "direction", plus center
- JPanel is a container (a thing!) that can display multiple components

Q2,3

JFrame BorderLayout



Advice

Look at the code in the capitalization example

Then solve the addLettersProblem

- Stage 1:
  - Make sure buttons show up
  - Make sure you can get message (JLabel) to appear
- Stage 2: Make sure buttons do ANYTHING
  - Just have them System.out.println(“pressed”)
- Stage 3:
  - Have the buttons perform desired behavior

General GUI Development Workflow

1. Create JFrame (configure!)
2. Create JPanel
3. Put JButtons (or JComponents) into JPanel
4. Add JPanel to JFrame
5. Create ActionListener
   (Might need to create class!)
6. Attach ActionListener to JButton
7. Does ActionListener have what it needs?
   (If not, pass it in the constructor!)

Mouse Listeners

```
public interface MouseListener {
   public void mouseClicked(MouseEvent e);
   public void mouseEntered(MouseEvent e);
   public void mouseExited(MouseEvent e);
   public void mousePressed(MouseEvent e);
   public void mouseReleased(MouseEvent e);
}
```

Q5

Repaint (and thin no more)

- To update graphics:
  - We tell Java library that we need to be redrawn:
    - `drawComponent.repaint()`
  - Library calls `paintComponent()` when it's ready

- **Don't call `paintComponent()` yourself! It's just there for Java's call back.**

Q4

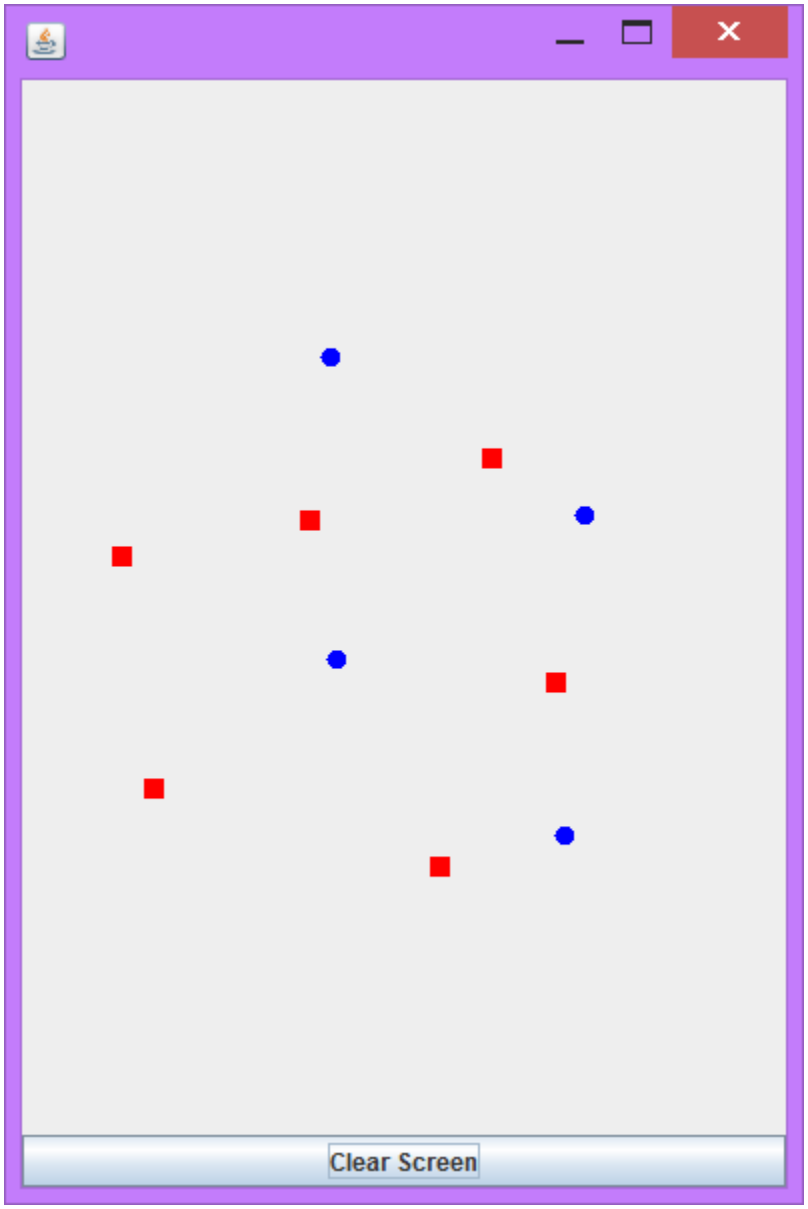
Activity 2

Read the code in the **rectangleExample**, then individually or in pairs solve the **clicksProblem**.

Draw a 20x20 blue circle upon left-click, centered on click

Clear screen button does what it says.

If you have time, make a right click make a red square



Using Inner Classes

- Classes can be defined **inside** other classes or methods
- Used for “smallish” helper classes

- Example: `Ellipse2D.Double`

Outer class

Inner class

- Often used for `ActionListener`s…
- Add to Breakfast program?

Q6

Anonymous Classes

- Sometimes very small helper classes are only used once
  - This is a job for an anonymous class!

- **Anonymous** → no name
- A special case of inner classes

- Used for the simplest `ActionListener`s…

Inner Classes and Scope

- **Inner classes can access any variables in surrounding scope**

- Caveats:
  - Can only use instance fields of surrounding scope if we're inside an instance method

- Example:
  - Prompt user for what porridge tastes like

Work Time

- LinearLightsOut