CSSE 220

Event Based Programming

Import out *EventBasedProgramming* from repo

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
  i.e., *what it does*:
  - Weighable objects (return a weight)
  - NumberSequences (get the next number, reset)
  - Pet (Can be fed, can tell if eating, can tell name)
- Classes represent the concrete idea
  i.e., *how it does it*:
  - Country, Bank Account
  - AddOne, PowersOfTwo.
  - Dog, Cat, Fish

Polymorphism! (A quick intro)

- Etymology:
  - Poly → many
  - Morphism → shape

  Skip

- 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()

Skip

- 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
  - Detects user input
  - Dispatches that input to our program by a callback to our program passing to our progra... the input event

- We handle events
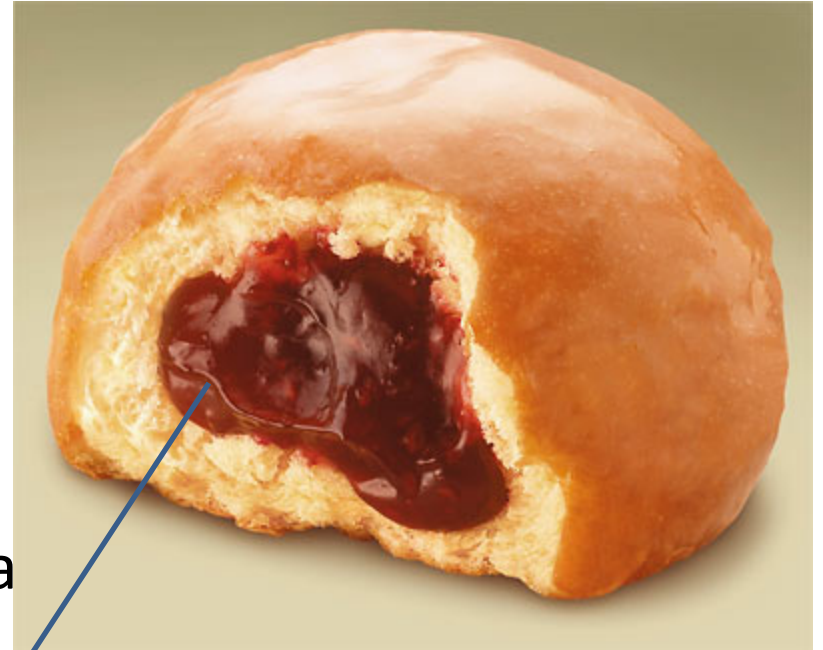
Hmm, donuts

Gooey

# The Next Graphics Assignment Preview

- Linear Lights Out
- 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

Mouse
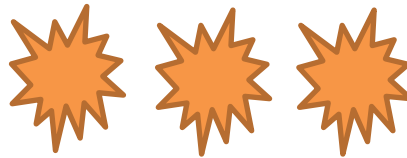
Button

Keyboard

Events

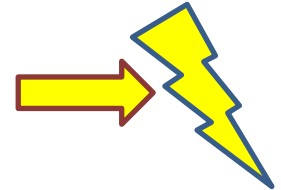MouseEvents

ActionEvents

KeyEvents

Event Listeners

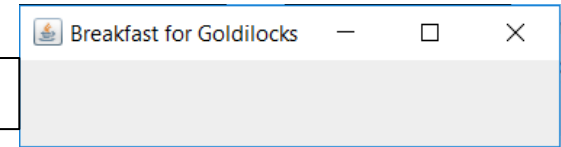(MouseListener)

ActionListener

Register!

(KeyListener)

Simple Interactive GUI Workflow

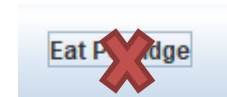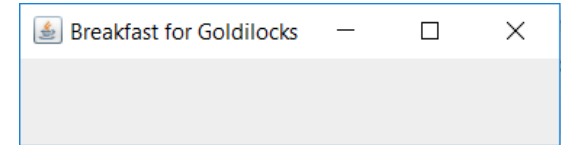### 1. Create JFrame *(Needs additional configuration)*

```
JFrame frame = new JFrame("Breakfast for Goldilocks");
```
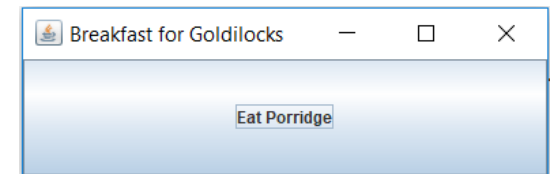
2. Create JButton

(JButton initially untethered and invisible)

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

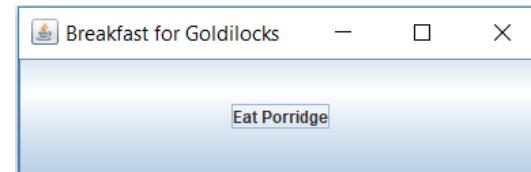### 3. Add JButton to JFrame (Can also be added to a JPanel)

```
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 );
```
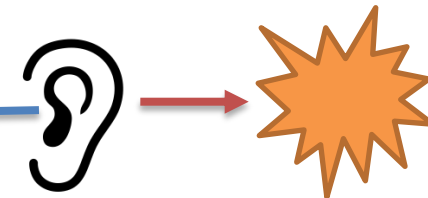
Live Coding

1. Set up a Window
2. Add a Button
3. Add an Event Listener to the Button
   - Create Listener – ActionListener from Java Swing
   - Register Listener with Button
4. Add a Panel, put Button on Panel, add Panel to Frame
5. Add a Label, put Label on Panel
6. Modify Label text when Button clicked

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!)

Key Layout Ideas

- JFrame's add(Component c) method
  - Adds a new component to be drawn
  - Throws out the old one!

Q2,3

JFrame BorderLayout
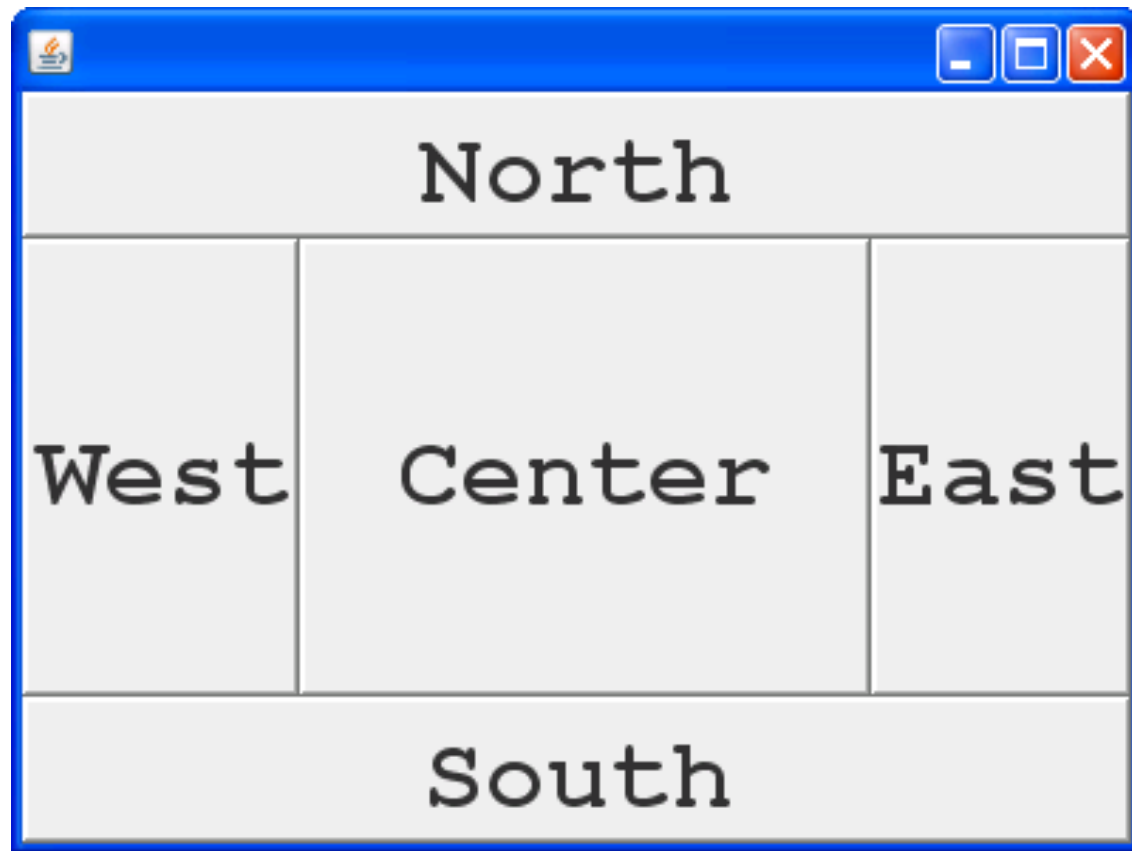
Key Layout Ideas – add to area

- JFrame also has method
  `add(Component c, Object constraint)`
  - Typical constraints:
    - `BorderLayout.NORTH`
      `BorderLayout.CENTER`
  - Can add one thing to each "direction", plus center

Q2,3

Key Layout Ideas - JPanel

- JPanel is a container component that can display multiple components

- Whereas there are only 5 areas in a Jframe North, South, East, West, and Center

Q2,3

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!

JFrame w/ 2 components added
1) Label to Center
2) Panel (w/ 3 Buttons) to South
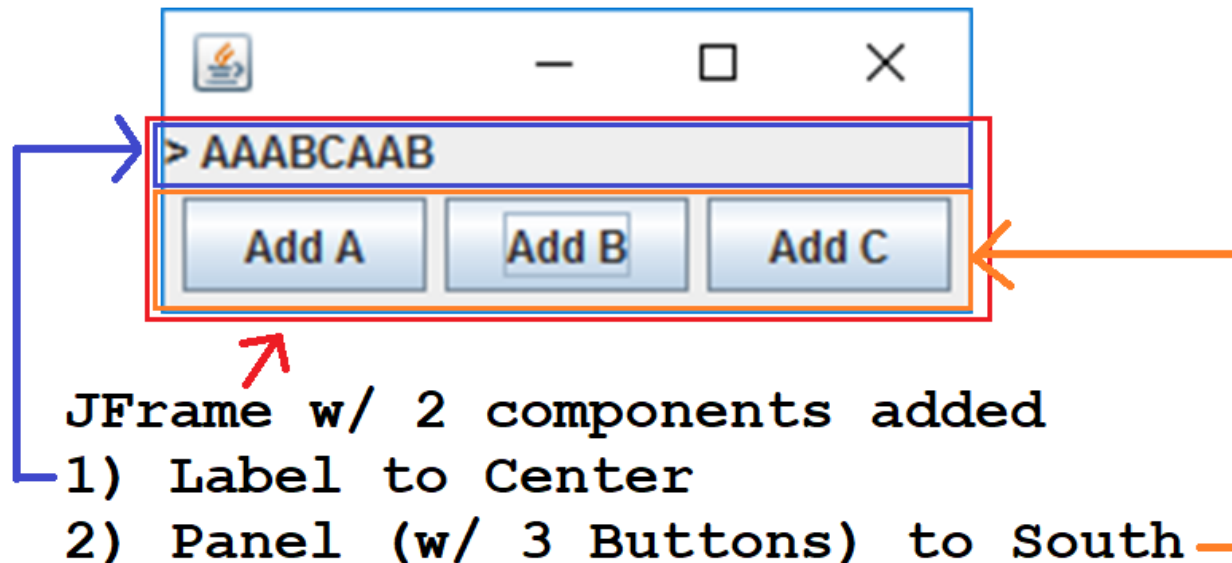
Advice

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

```java
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);
}
MouseEvent class
```

- **Method: getButton()**
- **Constants: BUTTON1 and BUTTON2**

```java
if(e.getButton() == MouseEvent.Button1) {…}
```

Repaint (and then 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

1) Read the code in the **rectangleExample**
2) 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
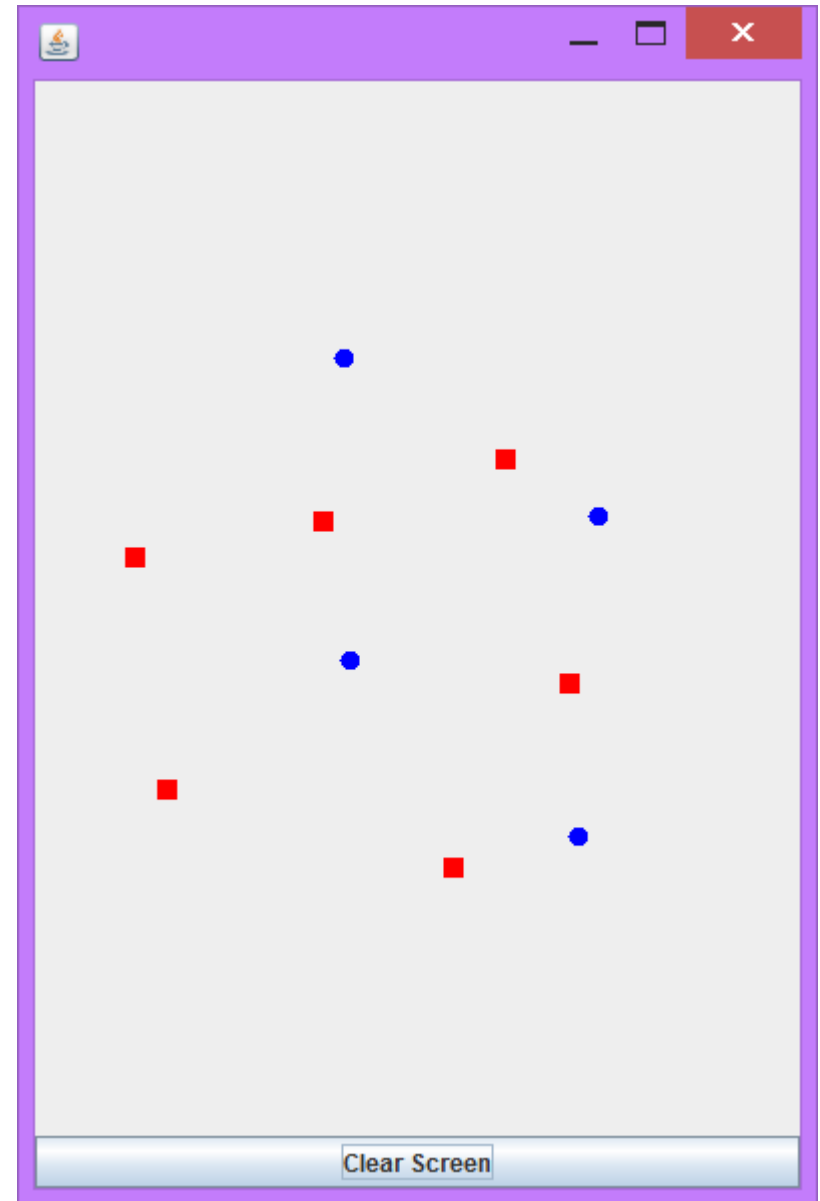
Using Inner Classes

- Classes can be defined **inside** other classes or methods
  - Used for "smallish" *helper* classes
  - Usually made *private* inside the outer class
  - Because it is inside the outer class, the inner class's methods have full access to outer class's data fields

- 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 by a client once
  - This is a job for an anonymous class!

- **Anonymous** → nameless

- A special case of inner classes

- E.g., 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