RADFORD UNIVERSITY

ITEC 120

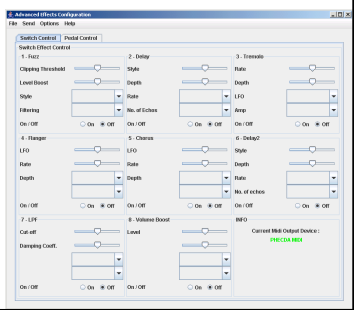
Lecture 38
GUI Interactivity

Review

- GUI basics
- JFrame
- JPanel
- JLabel, JButton, JTextField
- Layout managers
  - Flow / Box
  - Border / Grid

Objectives

- Add interactivity



Need

- When you click on a button…
- How does your code get called
  - Push
  - Pull
  - Both

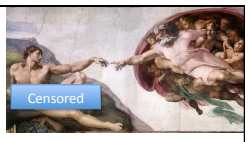
Pipe dream

Problem: this isn't how java works

```
public void buttonPressed(JButton button)
{
    if (button.getText().equals("+")
        System.out.println("You pressed the + key");
}
```

Interactivity

Problem



- Java doesn't know how to call your code
- Have to connect the two worlds
- When java was written vs when your program was written

Interactivity

Solution

- Have a specific function that java knows it can call
- Requires that you guarantee you have written the function

```
public class ButtonHandler implements ActionListener
{
    public void actionPerformed(ActionEvent event)
    {
        //Code that handles a button press here
    }
}
```

Interactivity

Rationale

- Interface
  - Guarantees that certain functions exist in a class
  - Useful for GUIs, architects
- Example

No implementation

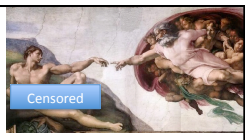
```
public interface simplemath
{
    int add(int a, int b);
    int subtract(int a, int b);
}
```

Will not compile

```
public class s implements simplemath
{
    int add(int a, int b) { return a+b;}
}
```

Interactivity

How?



- One line

```
public void actionPerformed(ActionEvent event)
{

        if (event.getSource() == button)

}
```

JButton that got clicked

Variable in your class

Compares memory address of button with what was clicked

Interactivity

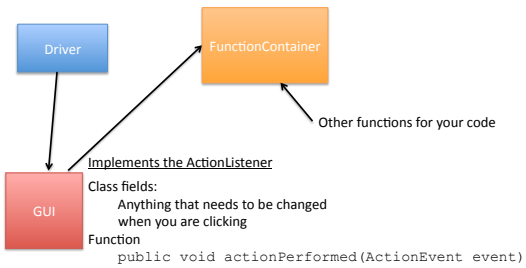
Circle.complete()

- Button needs to know who to notify it was pressed

```
public class GUI implements ActionListener
{
   JButton example;
   public GUI() { example = new JButton(); }
   public void actionPerformed(ActionEvent event)
   {
      if (event.getSource() == example)
         System.out.println("You pressed the button");
   }
}
```

Interactivity

Design



Interactivity

Code

```
public class GUI implements ActionListener
{
   private JButton button;
   public GUI()
   {
         button = new JButton();
         button.addActionListener(this);
   }
   public void actionPerformed(ActionEvent event)
   {
      if (event.getSource() == button)
      {
         System.out.println("Button pressed");
      }
   }
}
```

Associate button with listener

Interactivity

Create the GUI in a static void main and you are finished.

Code

```
public class Driver
{
    GUI theGUI = new GUI();
    theGUI.funcs = new FunctionContainer();
}
```

Program doesn't stop until GUI windows are closed

Hands off control to

References on button presses

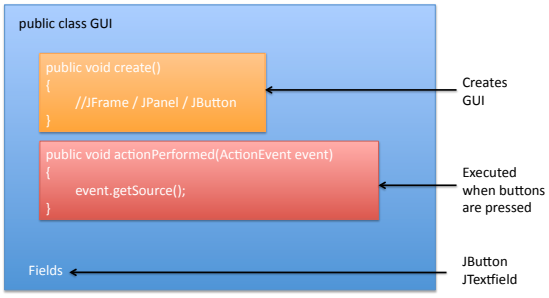
Interactivity

Problem

- Write a GUI
- Don't think about interactivity
- Variables are in functions, not classes
- Have to refactor your code
  - Hack to make it easier

Interactivity

Operation



Interactivity

Online examples

Inner class

- Syntax you might find in tutorials

```
public class GUI{
   private class Action implements ActionListener {
      public void actionPerformed(ActionEvent event){
         if (event.getSource() == button)
           System.out.println("You pressed the button");}
   }
   public GUI()
   {
      Action jackson = new Action();
      button = new JButton();
      button.addActionListener(jackson);
   }
   public JButton button;
}
```

Interactivity

Pro/Con

- Prevents anyone other than Java from pressing your buttons
- Easy to find bugs (1 source)
- Requires some head scratching
- Rarely used "feature"

Interfaces

- Meetings to figure out what to design
- Delegating tasks
- Enforcing decisions
- Hand checking vs javac checking

Summary

- Interfaces
  - ActionListener
- GUI design
  - How to create your classes