Today …

- Quiz 5
- Inner classes
- Exercise 7
- Lab 4
- Readings:
  - Ch. 6 pages 269-271
Panel with two buttons

```java
public class MyPanel extends JPanel implements ActionListener {
    private JButton button1 = new JButton("Don't Panic");
    private JButton button2 = new JButton("Panic");

    public MyPanel() {
        createGUI();
    }

    private void createGUI() {
        setLayout(new BoxLayout(this, BoxLayout.X_AXIS));
        button1.addActionListener(this);
        button2.addActionListener(this);
        add(button1);
        add(button2);
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame("MyPanel in a JFrame");
        frame.add(new MyPanel());
        frame.pack();
        frame.setVisible(true);
    }
}
```

Event handling

```java
public class MyPanel extends JPanel implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        if (event.getSource() == button1) {
            // do don't panic stuff
        }
        if (event.getSource() == button2) {
            // do panic stuff
        }
    }
}
```

- We have **two** buttons and **one** action event
  - This works but isn’t very “object oriented”
  - One method doing many things
  - Changing how one source is handled requires changing everyone’s event handler
Another approach

Create two separate ActionListener classes

```java
public class Button1Listener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        // do don’t panic stuff
    }
}
public class Button2Listener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        // do panic stuff
    }
}
```

Any issues here?
- The classes may not have access to the variables they need
- e.g., if panic changes the panel to red
- Causes a proliferation of classes
- Each one with a specific, narrow task

Wouldn’t it be nice …

If we could have two different listener classes

… but if each class had access to the instance variables of the main GUI class

… and we didn’t have a proliferation of classes

• We can with Inner Classes!
**Inner classes to the rescue!**

An inner class is a class nested within another class

```java
public class MyOuterClass {
    class MyInnerClass {
    }
}
```

- Inner classes can use **private** variables and methods of the **outer** class
  ... as if they were defined in the **inner** class!

```
inner.callInnerMe();    // sets x to 42!
```

---

**Inner Classes**

- This works:
  ```java
  public class MyOuterClass {
      private int x;
      public void callOuterMe() {
          MyInnerClass inner = new MyInnerClass();
          inner.callInnerMe();
      }
      private class MyInnerClass {
          public void callInnerMe() {
              x = 42;
          }
      }
  }
  MyOuterClass outer = new MyOuterClass();
  outer.callOuterMe(); // sets x to 42!
  ```
Inner Classes

An inner class can use all the methods and variables of the outer class

– … even the private ones!
– the inner class can use these as if they were declared within the inner class

• When we say an inner class accesses something in the outer class
  – We really mean an instance of the inner class
  – Having access to an instance of the outer class

Can any arbitrary instance of the inner class access the methods and variables of any instance of the outer class?

NO!

• An inner object must be tied to a specific outer object (on the heap)

1. Make an instance of the outer class
2. Make an instance of the inner class using the instance of the outer class
3. The outer and inner objects are now linked!
Inner Classes

• This works:

```java
public class MyOuterClass {
    private int x;
    public void callOuterMe() {
        MyInnerClass inner = new MyInnerClass();
        inner.callInnerMe();
    }
    private class MyInnerClass {
        public void callInnerMe() {
            x = 42;
        }
    }
}
```

The outer object creates the inner object here

```java
MyOuterClass outer = new MyOuterClass();
outer.callOuterMe();    // sets x to 42!
```

Local Inner Classes

You can also define inner classes within methods ...

```java
public void myMethod() {
    class InnerListener implements ActionListener {
        public void actionPerformed(ActionEvent e) { ... }
    }
    button.addActionListener(new InnerListener());
}
```

• These are called “Local Inner Classes”
  – Only the method has access to the inner class
  – Unlike regular inner classes, you can access local variables
  – However, these have to be defined final (const)
  – Ensures values of local variables can be copied into inner class
  – Local variable out of scope once method ends (must be copied)
Anonymous Inner Classes

You can go one step further with local inner classes

• If you only need a single object of the class

... you can create the object “on the fly”

... without having to name the class

button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) { ... }
});

You should use these “anonymous” inner classes judiciously
– they can make your code hard to read
– but concise for short (simple) inner classes

Panel with two buttons – Revised

public class MyPanel extends JPanel {
    private JButton button1 = new JButton("Don't Panic");
    private JButton button2 = new JButton("Panic");
    public MyPanel() {
        createGUI();
    } 
    private void createGUI() {
        setLayout(new BoxLayout(this, BoxLayout.X_AXIS));
        button1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) { // don't panic stuff }
        });
        button2.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) { // panic stuff }
        });
        add(button1);
        add(button2);
    }
    public static void main(String[] args) {
        JFrame frame = new JFrame("MyPanel in a JFrame");
        frame.add(new MyPanel());
        frame.pack();
        frame.setVisible(true);
    }
}
Inner classes are (almost) plain-old classes

• After you compile your outer classes

… you end up with a bunch of these files:

    Outer$Inner.class

*Inner classes are translated by the compiler to class files*

• You can “see” what these look like using:

    javap –private Outer$Inner

• Note you don’t see the method implementations

• See the textbook for more on the translation

---

Exercise 6
Exercise 6

1. The compile error:
   button.addActionListener(); // listener object missing!
   Fix: button.addActionListener(new ButtonListener());

2. As anonymous inner class:
   button.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
         if(button.getText().equals("A"))
            button.setText("B");
         else
            button.setText("A");
      }
   });

Lab 4

• Finish up Lab 3 (if you haven’t already)
• Modify Lab 3 …
  1. Remove the text field
  2. Add a JComboBox
  3. Use the value in the combo box to display in the dialog
     Note that combo boxes use action listeners
  4. Make your action listener an anonymous inner class!
  5. Play with editable and un-editable combo boxes (if time)