Inner Classes

```
public class DrawingExample extends JFrame {
   public boolean drawLine = false;
   private DrawingPanel drawingPanel;
   public DrawingExample() {
       super("Drawing Example");
       drawingPanel = new DrawingPanel(this);
   }
}
class DrawingPanel extends JPanel {
   private DrawingExample owner;
   public DrawingPanel (DrawingExample p) { owner = p; }
   public void paintComponent(Graphics gc) {
        super.paintComponent(gc);
        if (owner.drawLine) {
            gc.drawLine(10, 10, 100, 100);
        }
   }
}
```

Without Inner classes

Each class has a reference to the other

Needs to access frame's field

With Inner classes

```
public class DrawingExample extends JFrame {
   private boolean drawLine = false;
   private DrawingPanel drawingPanel;
   public DrawingExample() {
       super("Drawing Example");
       drawingPanel = new DrawingPanel();
   }

   class DrawingPanel extends JPanel {
    public DrawingPanel () {  }
    public void paintComponent(Graphics gc) {
        super.paintComponent(gc);
        if (drawLine) {
            gc.drawLine(10, 10, 100, 100);
        }
   }
}
```

drawLine is private

No explicit reference to frame from panel

Inner class can access frame's private members directly

Basic Example

Key idea: Classes can be *members* of other classes…

```java
public class Outer {
  private int outerVar;
  public Outer () {
    outerVar = 6;
  }
  public class Inner {
    private int innerVar;
    public Inner(int z) {
      innerVar = outerVar + z;
    }
  }
}
```

Name of this class is Outer.Inner
(which is also the static type of objects that this class creates)

Reference from inner class to instance variable bound in outer class

Object Creation

- Inner classes can refer to the instance variables and methods of the outer class
- Inner class instances usually created by the methods/constructors of the outer class

```
public Outer () {
   Inner b = new Inner ();
}
```

Actually   this.new

- Inner class instances *cannot* be created independently of a containing class instance.

```
Outer.Inner b = new Outer.Inner()
```
✗

```
Outer a = new Outer();
Outer.Inner b = a.new Inner();
```
✓

```
Outer.Inner b = (new Outer()).new Inner();
```
✓

Anonymous Inner class

- New *expression* form: define a class and create an object from it all at once

New keyword

```
new InterfaceOrClassName() {
    public void method1(int x) {
        // code for method1
    }
    public void method2(char y) {
        // code for method2
    }

}
```

Normal class definition, no constructors allowed

Static type of the expression is the Interface/superclass used to create it

Dynamic class of the created object is anonymous! Can't really refer to it.

Like first-class functions

- Anonymous inner classes are the Java equivalent of Ocaml first-class functions

- Both create "delayed computation" that can be stored in a data structure and run later
  - Code stored by the event / action listener
  - Code only runs when the button is pressed
  - Could run once, many times, or not at all
- Both sorts of computation can refer to variables in the current scope
  - OCaml: Any available variable
  - Java: only instance variables (fields) and variables marked final