Outline

• Inner Classes
Inner Class Rules

- An inner class, or nested class, is defined within the scope of another class
  - An inner class can be used just like a regular class, though it is typically only used by its outer class
  - An inner class can reference the data and methods defined in the outer class
  - An inner class can be defined with a visibility modifier (i.e. `private`
  - If an inner class is defined as `static`, it can be accessed using the outer class name (not an instance)
    - The inner class will not be able to access non-static members of the outer class
    - To create an instance of a `static` inner class from a class other than the outer class, write
      ```java
      OuterClass.InnerClass ic = new OuterClass.InnerClass();
      ```
    - To create an instance of a `non-static` inner class from a class other than the outer class, write
      ```java
      Outerclass oc = new OuterClass();
      OuterClass.InnerClass ic = oc.new InnerClass();
      ```
public class Test {
    public Test() {
        Outer o = new Outer();
        Outer.Inner in = o.new Inner(3);
        System.out.println("num=" + in.getNum());
    }
    public static void main(String args[]) {
        Test t = new Test();
    }
}

class Outer {
    public Outer() {
        System.out.println("Outer constructor");
    }
    class Inner {
        private int num;
        public Inner(int num) {
            System.out.println("Inner constructor");
            this.num = num;
        }
        public int getNum() {
            return num;
        }
    } // ends Inner
} // ends Outer

The class Inner is inside the class Outer
public class Test {
    public Test() {
        Outer.Inner in = new Outer().new Inner(3);
        System.out.println("num=" + in.getNum());
    }
    public static void main(String args[]) {
        Test t = new Test();
    }
}

class Outer {
    public Outer() {
        System.out.println("Outer constructor");
    }
}

class Inner {
    private int num;
    public Inner(int num) {
        System.out.println("Inner constructor");
        this.num = num;
    }
    public int getNum() {
        return num;
    }
} // ends Inner
} // ends Outer
static class Inner {
    private int num;
    public Inner(int num) {
        System.out.println("Inner constructor");
        this.num = num;
    }
    public int getNum() {
        return num;
    }
} // ends Inner

} // ends Outer
Anonymous Inner Classes

- An anonymous inner class is an inner class without a name
  - It combines defining an inner class and creating an instance of the class
  - Anonymous inner classes **always** extend a superclass or implement an interface, but they cannot explicitly do so
  - Anonymous inner classes **must** implement all the abstract methods in the superclass or interface
    - In other words, an **anonymous inner class cannot be abstract**
  - Anonymous inner classes do not have constructors since we don’t know the name of the class
    - Anonymous inner classes automatically inherit the constructors from the parent class (which is different than non-anonymous class inheritance)
    - Obviously if inheriting from an interface, the default constructor is the only constructor an anonymous inner class will have

- Although anonymous inner classes can be used anytime, they are often used with event handling of GUI components
Anonymous Inner Class Example #1

```java
public class Test {
    public Test() {
        showValues(new Printer() {
            public void printNum() {
                System.out.println("num=" + 10);
            }
            public String getName() {
                return name;
            }
        });
    }
    public void showValues(Printer p) {
        p.printNum();
        System.out.println("name=" + p.getName());
    }
    public static void main(String args[]) {
        Test t = new Test();
    }
}

interface Printer {
    public static final int num = 10;
    public static final String name = "CSCI 201";
    public void printNum();
    public String getName();
}
```

This is not instantiating the Printer interface. Instead, it is anonymously instantiating a class that has implemented the Printer interface.

What is the name of the instance?
Anonymous Inner Class Example #2

```java
public class Test {
    public static void foo(C1 c) {
        c.foo();
        c.bar();
    }
    public static void main(String[] args) {
        Test.foo(new C1("a") {
            void bar() {
                System.out.println("b");
            }
        });
    }
}

class C1 {
    private String s;
    public C1(String s) {
        this.s = s;
    }
    void foo() {
        System.out.println(s);
    }
    void bar() {
        System.out.println("d");
    }
}
```

The anonymous inner class inherited the constructor in C1 that takes a String as a parameter, so when we instantiate the anonymous inner class on line 7, it is calling the constructor in the anonymous inner class that takes a String as a parameter.

The constructor in the anonymous inner class just calls `super("hi");`