Inner Classes

CSCI 201
Principles of Software Development

Jeffrey Miller, Ph.D.
jeffrey.miller@usc.edu

USC Viterbi
School of Engineering

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

```
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

```
Outerclass oc = new OuterClass();
OuterClass.InnerClass ic = oc.new InnerClass();
```

Inner Class Example 1

```
1   public class Test {
2     public Test() {
3       Outer o = new Outer();
4       Outer.Inner in = o.new Inner(3);
5       System.out.println("num=" + in.getNum());
6     }
7     public static void main(String args[]) {
8       Test t = new Test();
9     }
10  }
11
12  class Outer {
13    public Outer() {
14      System.out.println("Outer constructor");
15    }
16
17    class Inner {
18      private int num;
19      public Inner(int num) {
20        System.out.println("Inner constructor");
21        this.num = num;
22      }
23      public int getNum() {
24        return num;
25      }
26    } // ends Inner
27  } // ends Outer
```

The class `Inner` is inside the class `Outer`

Inner Class Example 2

```
1   public class Test {
2     public Test() {
3       Outer.Inner in = new Outer().new Inner(3);
4       System.out.println("num=" + in.getNum());
5     }
6     public static void main(String args[]) {
7       Test t = new Test();
8     }
9   }
10
11  class Outer {
12    public Outer() {
13      System.out.println("Outer constructor");
14    }
15
16    class Inner {
17      private int num;
18      public Inner(int num) {
19        System.out.println("Inner constructor");
20        this.num = num;
21      }
22      public int getNum() {
23        return num;
24      }
25    } // ends Inner
26  } // ends Outer
```

Anonymous instantiation of `Outer`

Static Inner Class Example

```
1   public class Test {
2     public Test() {
3       Outer.Inner in = new Outer.Inner(3);
4       System.out.println("num=" + in.getNum());
5     }
6     public static void main(String args[]) {
7       Test t = new Test();
8     }
9   }
10
11  class Outer {
12    public Outer() {
13      System.out.println("Outer constructor");
14    }
15
16    static class Inner {
17      private int num;
18      public Inner(int num) {
19        System.out.println("Inner constructor");
20        this.num = num;
21      }
22      public int getNum() {
23        return num;
24      }
25    } // ends Inner
26  } // ends Outer
```

Since the class `Inner` is static, you do not need a reference to the class `Outer` to instantiate it

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

```
1   public class Test {
2     public Test() {
3       showValues(new Printer() {
4         public void printNum() {
5           System.out.println("num=" + 10);
6         }
7         public String getName() {
8           return name;
9         }
10      });
11    }
12    public void showValues(Printer p) {
13      p.printNum();
14      System.out.println("name=" + p.getName());
15    }
16    public static void main(String args[]) {
17      Test t = new Test();
18    }
19  }
20
21  interface Printer {
22    public static final int num = 10;
23    public static final String name = "CSCI 201";
24    public void printNum();
25    public String getName();
26  }
```

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

```
1  public class Test {
2    public static void foo(C1 c) {
3      c.foo();
4      c.bar();
5    }
6    public static void main(String [] args) {
7      Test.foo(new C1("a") {
8        void bar() {
9          System.out.println("b");
10       }
11     });
12   }
13 }
14 class C1 {
15   private String s;
16   public C1(String s) {
17     this.s = s;
18   }
19   void foo() {
20     System.out.println(s);
21   }
22   void bar() {
23     System.out.println("d");
24   }
25 }
```

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 cCalls super("hi");