CMSC 202

Inner Classes

Simple Uses of Inner Classes

- **Inner classes** are classes defined within other classes
  - The class that includes the inner class is called the **outer class**
  - There is no particular location where the definition of the inner class (or classes) must be place within the outer class
  - Placing it first or last, however, will guarantee that it is easy to find

Simple Uses of Inner Classes

- An inner class definition is a member of the outer class in the same way that the instance variables and methods of the outer class are members
  - An inner class is local to the outer class definition
  - The name of an inner class may be reused for something else outside the outer class definition
  - If the inner class is private, then the inner class cannot be accessed by name outside the definition of the outer class

Inner/Outer Classes

```
public class Outer
{
        private class Inner
        {
                // inner class instance variables
                // inner class methods

        } // end of inner class definition

        // outer class instance variables
        // outer class methods
}
```

Simple Uses of Inner Classes

- There are two main advantages to inner classes
  - They can make the outer class more self-contained since they are defined inside a class
  - Both of their methods have access to each other's private methods and instance variables
- Using an inner class as a helping class is one of the most useful applications of inner classes
  - If used as a helping class, an inner class should be marked private

Inner and Outer Classes Have Access to Each Other's Private Members

- Within the definition of a method of an inner class:
  - It is legal to reference a private instance variable of the outer class
  - It is legal to invoke a private method of the outer class
  - Essentially, the inner class has a hidden reference to the outer class
- Within the definition of a method of the outer class
  - It is legal to reference a private instance variable of the inner class on an object of the inner class
  - It is legal to invoke a (nonstatic) method of the inner class as long as an object of the inner class is used as a calling object

- Within the definition of the inner or outer classes, the modifiers **`public`** and **`private`** are equivalent

Class with an Inner Class

Display 13.9 **Class with an Inner Class** ***(Part 1 of 2)***

```
1  public class BankAccount
2  {
3      private class Money
4      {
5          private long dollars;
6          private int cents;

7          public Money(String stringAmount)
8          {
9              abortOnNull(stringAmount);
10             int length = stringAmount.length();
11             dollars = Long.parseLong(
12                         stringAmount.substring(0, length - 3));
13             cents = Integer.parseInt(
14                         stringAmount.substring(length - 2, length));
15         }

16         public String getAmount()
17         {
18             if (cents > 9)
19                return (dollars + "." + cents);
20             else
21                return (dollars + ".0" + cents);
22         }
```

The modifier `private` in this line should not be changed to `public`. However, the modifiers `public` and `private` inside the inner class `Money` can be changed to anything else and it would have no effect on the class `BankAccount`.

Class with an Inner Class

Display 13.9 Class with an Inner Class *(Part 1 of 2)* (continued)

```
23      public void addIn(Money secondAmount)
24      {
25          abortOnNull(secondAmount);
26          int newCents = (cents + secondAmount.cents)%100;
27          long carry = (cents + secondAmount.cents)/100;
28          cents = newCents;
29          dollars = dollars + secondAmount.dollars + carry;
30      }

31      private void abortOnNull(Object o)
32      {
33          if (o == null)
34          {
35              System.out.println("Unexpected null argument.");
36              System.exit(0);
37          }
38      }
39  }
```

The definition of the inner class ends here, but the definition of the outer class continues in Part 2 of this display.

Class with an Inner Class

Display 13.9 Class with an Inner Class *(Part 2 of 2)*

```
40      private Money balance;

41      public BankAccount()
42      {
43          balance = new Money("0.00");
44      }

45      public String getBalance()
46      {
47          return balance.getAmount();
48      }

49      public void makeDeposit(String depositAmount)
50      {
51          balance.addIn(new Money(depositAmount));
52      }

53      public void closeAccount()
54      {
55          balance.dollars = 0;
56          balance.cents = 0;
57      }
58  }
```

To invoke a nonstatic method of the inner class outside of the inner class, you need to create an object of the inner class.

This invocation of the inner class method `getAmount()` would be allowed even if the method `getAmount()` were marked as `private`.

Notice that the outer class has access to the private instance variables of the inner class.

This class would normally have more methods, but we have only included the methods we need to illustrate the points covered here.

Referring to a Method of the Outer Class

- If a method is invoked in an inner class
  - If the inner class has no such method, then it is assumed to be an invocation of the method of that name in the outer class
  - If both the inner and outer class have a method with the same name, then it is assumed to be an invocation of the method in the inner class
  - If both the inner and outer class have a method with the same name, and the intent is to invoke the method in the outer class, then the following invocation must be used:

    ```
    OuterClassName.this.methodName()
    ```

Public Inner Classes

- If an inner class is marked **`public`**, then it can be used outside of the outer class
- In the case of a nonstatic inner class, it must be created using an object of the outer class

```
BankAccount account = new BankAccount();
BankAccount.Money amount =
   account.new Money("41.99");
```

  - Note that the prefix ***`account.`*** must come before ***`new`***
  - The new object **`amount`** can now invoke methods from the inner class, but only from the inner class

Public Inner Classes

- In the case of a static inner class, the procedure is similar to, but simpler than, that for nonstatic inner classes

```
OuterClass.InnerClass innerObject =
                        new
    OuterClass.InnerClass();
```

  - Note that all of the following are acceptable

```
innerObject.nonstaticMethod();
innerObject.staticMethod();
OuterClass.InnerClass.staticMethod();
```

Public Money Inner Class

If the Money inner class in the BankAccount example was defined as public, we can create and use objects of type Money outside the BankAccount class.

```
// this is okay in main( )
BankAccount account = new BankAccount( );
BankAccount.Money amt =   // note syntax
                  account.new Money( "41.99" );
System.out.println( amt.getAmount( ) );
// but NOT this - why not??
System.out.println( amt.getBalance( ) );
```

Static Inner Classes

- A normal inner class has a connection between its objects and the outer class object that created the inner class object
  - This allows an inner class definition to reference an instance variable, or invoke a method of the outer class
- There are certain situations, however, when an inner class must be static
  - If an object of the inner class is created within a static method of the outer class
  - If the inner class must have static members

Static Inner Classes

- Since a static inner class has no connection to an object of the outer class, within an inner class method
  - Instance variables of the outer class cannot be referenced
  - Nonstatic methods of the outer class cannot be invoked
- To invoke a static method or to name a static variable of a static inner class within the outer class, preface each with the name of the inner class and a dot

Multiple Inner Classes

- A class can have as many inner classes as it needs.
- Inner classes have access to each other's private members as long as an object of the other inner class is used as the calling object.

The `.class` File for an Inner Class

- Compiling any class in Java produces a `.class` file named ***`ClassName.class`***
- Compiling a class with one (or more) inner classes causes both (or more) classes to be compiled, and produces two (or more) .class files
  - Such as ***`ClassName.class`*** **and** ***`ClassName$InnerClassName.class`***

Nesting Inner Classes

- It is legal to nest inner classes within inner classes
  - The rules are the same as before, but the names get longer
  - Given class **`A`**, which has public inner class **`B`**, which has public inner class **`C`**, then the following is valid:

```
A aObject = new A();
A.B bObject = aObject.new B();
A.B.C cObject = bObject.new C();
```

Inner Classes and Inheritance

- Given an **`OuterClass`** that has an **`InnerClass`**
  - Any **`DerivedClass`** of **`OuterClass`** will automatically have **`InnerClass`** as an inner class
  - In this case, the **`DerivedClass`** cannot override the **`InnerClass`**
- An outer class can be a derived class
- An inner class can be a derived class also

Anonymous Classes

- If: you need to create only a single instance of a new class, that extends some class or implements some interface, an *anonymous class* definition can be used
  - A self-contained subclass definition that is embedded inside the **`new`** expression itself
  - An anonymous class is an abbreviated notation for simultaneously defining a class and instantiating an object "in-line" within any expression
- Restriction: an anonymous class can only be used (i.e., instantiated) once (although that line of code can then be executed multiple times)
- Anonymous classes are usually used when you only need one or two special-purpose methods

Anonymous Classes

- Anonymous classes are sometimes used when they are to be assigned to a variable of another type
  - The other type must be such that an object of the anonymous class is also an object of the other type
  - The other type is usually a Java interface
- Anonymous classes are often, but not always, a good substitute for inner classes
  - The need for very simple "one-use" helper classes is a common enough case that they merit some syntactic shorthand support in the language.

Anonymous Classes

Display 13.11 Anonymous Classes *(Part 1 of 2)*

This is just a toy example to demonstrate the Java syntax for anonymous classes.

```
1  public class AnonymousClassDemo
2  {
3      public static void main(String[] args)
4      {
5          NumberCarrier anObject =
6                  new NumberCarrier()
7                  {
8                      private int number;
9                      public void setNumber(int value)
10                     {
11                         number = value;
12                     }
13                     public int getNumber()
14                     {
15                        return number;
16                     }
17                 };
```

Anonymous Classes

Display 13.11 Anonymous Classes *(Part 1 of 2)*

```
18          NumberCarrier anotherObject =
19                  new NumberCarrier()
20                  {
21                      private int number;
22                      public void setNumber(int value)
23                      {
24                          number = 2*value;
25                      }
26                      public int getNumber()
27                      {
28                          return number;
29                      }
30                  };
31          anObject.setNumber(42);
32          anotherObject.setNumber(42);
33          showNumber(anObject);
34          showNumber(anotherObject);
35          System.out.println("End of program.");
36      }

37      public static void showNumber(NumberCarrier o)
38      {
39          System.out.println(o.getNumber());
40      }

41  }
```

This is still the file AnonymousClassDemo.java.

Anonymous Classes

Display 13.11 Anonymous Classes *(Part 2 of 2)*

SAMPLE DIALOGUE

```
42
84
End of program.
```

This is the file NumberCarrier.java.

```
1  public interface NumberCarrier
2  {
3      public void setNumber(int value);
4      public int getNumber();
5  }
```