Anonymous Classes

If you wish to create an object but have no need to name the object's class, then you can embed the class definition inside the expression with the new operator. These sorts of class definitions are called **anonymous classes** because they have no class name. An expression with an anonymous class definition is, like everything in Java, inside of some class definition. Thus, an anonymous class is an inner class. Before we go into the details of the syntax for anonymous classes, let's say a little about where one might use them.

The most straightforward way to create an object is the following:

```java
YourClass anObject = new YourClass();
```

If `new YourClass()` is replaced by some expression that defines the class but does not give the class any name, then there is no name `YourClass` to use to declare the variable `anObject`. So, it does not make sense to use an anonymous class in this situation. However, it can make sense in the following situation:

```java
SomeOtherType anObject = new YourClass();
```

Here `SomeOtherType` must be a type such that an object of the class `YourClass` is also an object of `SomeOtherType`. In this situation you can replace `new YourClass()` with an expression including an anonymous class instead of `YourClass`. The type `SomeOtherType` is usually a Java interface.

Here's an example of an anonymous class. Suppose you define the following interface:

```java
public interface NumberCarrier
{
    public void setNumber(int value);
    public int getNumber();
}
```

Then the following creates an object using an anonymous class definition:

```java
NumberCarrier anObject = new NumberCarrier()
{
    private int number;
    public void setNumber(int value)
    {
        number = value;
    }
    public int getNumber()
    {
        return number;
    }
};
```
The part in the braces is the same as the part inside the main braces of a class definition. The closing brace is followed by a semicolon, unlike a class definition. (This is because the entire expression will be used as a Java statement.) The beginning part, repeated below, may seem strange:

```java
new NumberCarrier()
```

The `new` is sensible enough but what's the point of `NumberCarrier()`? It looks like this is an invocation of a constructor for `NumberCarrier`. But, `NumberCarrier` is an interface and has no constructors. The meaning of `new NumberCarrier()` is simply

```java
implements NumberCarrier
```

So what is being said is that the anonymous class implements the `NumberCarrier` interface and is defined as shown between the main braces.

Display 13.11 shows a simple demonstration with two anonymous class definitions. For completeness we have also repeated the definition of the `NumberCarrier` interface in that display.

---

Display 13.1 Anonymous Classes (part 1 of 2)

```java
public class AnonymousClassDemo {
  public static void main(String[] args) {
    NumberCarrier anObject =
      new NumberCarrier() {
        private int number;
        public void setNumber(int value) {
          number = value;
        }
        public int getNumber() {
          return number;
        }
      };

    NumberCarrier anotherObject =
      new NumberCarrier() {
        private int number;
        public void setNumber(int value) {
          number = 2 * value;
        }
      };
  }
}
```

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

(continued)
Display 13.11 Anonymous Classes (part 2 of 2)

```java
26 public int getNumber()
27 {
28     return number;
29 };
31
32     anObject.setNumber(42);
33     anotherObject.setNumber(42);
34     showNumber(anObject);
35     showNumber(anotherObject);
36     System.out.println("End of program.");
37 }
38
39 public static void showNumber(NumberCarrier o)
40 {
41     System.out.println(o.getNumber());
42 }
```

This is the file AnonymousClassDemo.java.

Sample Dialogue

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

This is the file NumberCarrier.java.

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

TIP: Why Use Inner Classes?

Most simple situations do not need inner classes. However, there are situations for which inner classes are a good solution. For example, suppose you want to have a class with two base classes. That is not allowed in Java. However, you can have an outer class derived from one base class with an inner class derived from the other base class. Since the inner and outer classes have access to each other’s instance variables and methods, this can often serve as if it were a class with two base classes.