5.6: Understanding static and final keywords

IT1406 - Introduction to Programming

Level I - Semester 1

Understanding static

- When a member is declared **static**, it can be accessed before any objects of its class are created, and without reference to any object. You can declare both methods and variables to be **static**. The most common example of a **static** member is **main( )**. **main( )** is declared as **static** because it must be called before any objects exist.
- Instance variables declared as **static** are, essentially, global variables. When objects of its class are declared, no copy of a **static** variable is made. Instead, all instances of the class share the same **static** variable.
- Methods declared as **static** have several restrictions:
  - They can only directly call other **static** methods.
  - They can only directly access **static** data.
  - They cannot refer to **this** or **super** in any way.

Understanding static

- The following example shows a class that has a **static** method, some **static** variables, and a **static** initialization block:

```
class UseStatic {
    static int a = 3;
    static int b;
    static void meth(int x) {
        System.out.println("x = " + x);
        System.out.println("a = " + a);
        System.out.println("b = " + b);
    }
    static {
        System.out.println("Static block initialized.");
        b = a * 4;
    }
    public static void main(String args[]) {
        meth(42);
    }
}
```

Understanding static

- As soon as the **UseStatic** class is loaded, all of the **static** statements are run. First, **a** is set to **3**, then the **static** block executes, which prints a message and then initializes **b** to **a*4** or **12**. Then **main( )** is called, which calls **meth( )**, passing **42** to **x**. The three **println( )** statements refer to the two **static** variables **a** and **b**, as well as to the local variable **x**.
- Here is the output of the program:

```
Static block initialized.
x = 42
a = 3
b = 12
```

Understanding static

- if you wish to call a **static** method from outside its class, you can do so using the following general form:

  classname.method( )

- Here, *classname* is the name of the class in which the **static** method is declared. As you can see, this format is similar to that used to call non-**static** methods through object reference variables. A **static** variable can be accessed in the same way—by use of the dot operator on the name of the class. This is how Java implements a controlled version of global methods and global variables.
- Here is an example. Inside **main( )**, the **static** method **callme( )** and the **static** variable **b** are accessed through their class name **StaticDemo**.

Understanding static

```
class StaticDemo {
        static int a = 42;
        static int b = 99;
        static void callme() {
                System.out.println("a = " + a);
        }
}
class StaticByName {
        public static void main(String args[]) {
                StaticDemo.callme();
                System.out.println("b = " + StaticDemo.b);
        }
}
```

- Here is the output of this program:

```
a = 42
b = 99
```

Introducing final

- A field can be declared as **final**. Doing so prevents its contents from being modified, making it, essentially, a constant. This means that you must initialize a **final** field when it is declared. You can do this in one of two ways: First, you can give it a value when it is declared. Second, you can assign it a value within a constructor. The first approach is the most common. Here is an example:

```
final int FILE_NEW = 1;
final int FILE_OPEN = 2;
final int FILE_SAVE = 3;
final int FILE_SAVEAS = 4;
final int FILE_QUIT = 5;
```

Introducing final

- Subsequent parts of your program can now use **FILE_OPEN**, etc., as if they were constants, without fear that a value has been changed. It is a common coding convention to choose all uppercase identifiers for **final** fields, as this example shows.
- In addition to fields, both method parameters and local variables can be declared **final**. Declaring a parameter **final** prevents it from being changed within the method. Declaring a local variable **final** prevents it from being assigned a value more than once.
- The keyword **final** can also be applied to methods, but its meaning is substantially different than when it is applied to variables.