Import Statements, Instance Members, and the Default Constructor

Introduction

In this article from my free Java 8 course, I will be discussing `import` statements, instance members, and the default constructor.

Instance Members vs. Static Members

As you can see below, I have created the `Person` class, and within that class we have an instance variable called `‘personName’` and an instance method called `helloWorld()`. Our variables and methods are called the **instance members** of the class.

```
package com.marcusbiel.java8course;

import com.marcusbiel.java8course.attributes.Name;

public class Person {

     private Name personName;

     public String helloWorld() {
          return “Hello World”;
     }
}
```

Example 1

So what is an **instance**? An instance is a single occurrence of an object. While there is only one class, there are many instances of the class: objects. For example, we can have hundreds and hundreds of different `Person` objects. Each Person object has its own instance of the `personName` object, each with its own value and its own version of the `helloWorld()` method. Besides instance variables and methods, there can also be static ones. All instances of a class **share** the `static` variables or `static` methods of the class.

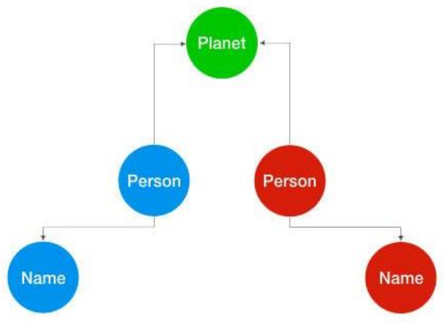
Variables and method are also called “**members**”. A member that belongs to the instance, is called instance member; a member that belongs to the class (a static variable or method) is called class member.

Besides instance members there are also static members, or class members. Class variables and methods are created by adding the `static` keyword to a method or variable.

```
package com.marcusbiel.java8course;

import com.marcusbiel.java8course.attributes.Name;

public class Person {

     private Name personName;
     private static Planet homePlanet;

     public static String helloWorld() {
          return “Hello World”;
     }
}
```

Example 2

If we change the `homePlanet` value for one instance of `Person`, all of the other instances will have their values changed as well. This is an advanced topic that will be discussed later in more detail. For now, just know that without the `static` keyword, our methods and variables are instance variables and methods.



Example 3

Constructors

The next thing we will do in our program is write a **constructor** for our Person Object. I’ve mentioned constructors before, but to go into more detail, constructors are the code that is used to create objects. Constructors start with the short name of the class, followed by parentheses, that contain any parameters, and then an opening curly brace and a closing curly brace. In between the curly braces you assign values to instance variables and call any methods that need to be called when an object of this class type is created. Constructors may look like methods, but they neither return a value nor do they have the tag `void`. You can have as many constructors as you want in a class, but each constructor needs a unique set of parameters.

If you don't write a constructor for your class, the compiler will automatically include a constructor that contains no parameters. This constructor is called the **default constructor**. This is a bit problematic in my opinion. If you do (at a later time) add a constructor that does contain one or more parameters, the default constructor will not be added by the compiler. I refer to such automatic compiler actions as "magic", as they are not always clear and easily lead to confusion.

Therefore, I generally recommend that you always write out your default constructor in code, if you need it. Never rely on the compiler to add it for you. However, this may lead to further confusion about why you added a redundant default constructor. Another developer might even just "extend" it later, adding parameters to the default constructor- effectively removing it from your class.
So I suggest you use the following approach by default: if you need a default constructor, add it explicitly. Then leave a comment documenting *why* you explicitly added the default constructor in this case. To give a concrete example: default constructors are often needed when certain frameworks are in use, like Hibernate, for instance.

For our Person class, I'm going to create a constructor that sets the value of `personName` given a value. If you look at the example below, I've created two different variables called `personName`. One of them is the instance variable in the class, and one of them is an argument in our constructor. To differentiate them, we have to add "`this.`" to the instance variable `personName`. This way `this.personName` is set to the `personName` received as an argument in the constructor.

```java
package com.marcusbiel.java8course;

import com.marcusbiel.java8course.attributes.Name;

public class Person {

     private Name personName;
     private static Planet homePlanet;

     public Person(Name personName) {
          this.personName = personName;
     }

     public static String helloWorld() {
          return "Hello World";
     }
}
```
Example 4

Since we've created a constructor, the compiler won't automatically use a default constructor. Now whenever someone calls the constructor to create an object of type person, they have to include the `personName` since that is the only constructor available to them.

If you remember from our test class from last article, we constructed our object without any arguments. If we try to execute this test again, it will fail.

Test Case Creation

As I've talked about before, a test method is annotated with `@Test`. This annotation invokes the Test class from the JUnit framework and sets up a test environment in our Java Virtual Machine (JVM). Let's create a second method in our test class so we can see how how our test class works a little more in depth. In our test class, I'm going to create a second method, `shouldReturnHelloMarcus()`.

```java
package com.marcusbiel.java8course;

import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class PersonTest {

     @Test
     public void shouldReturnHelloWorld() {
          Person person = new Person();
          assertEquals("Hello World", person.helloWorld() );
     }

     @Test
     public void shouldReturnHelloMarcus() {
          Person marcus = new Person();
          assertEquals("Hello Marcus", marcus.hello(“Marcus”));
    }
}
```

Example 5

Once I've created this method, I'm going to create an object of type `Person` with an instance variable name of `marcus`. If we try to run this, we will get a compilation error as I noted earlier, because our `shouldReturnHelloWorld()` method tries to create a Person object with a default constructor. To get rid of this error, we'll simply create a default constructor in our class.

```
package com.marcusbiel.java8course;

import com.marcusbiel.java8course.attributes.Name;

public class Person {

     private Name personName;
     private static String homePlanet;

     public Person() {
           /*
           * default constructor, required by Hibernate
           */
     }
}
```
Example 6

Comments

As I said before,there might be confusion later as to why you created this default constructor. You may even forget the reason yourself, further down the line. Adding comments to the code will prevent the confusion. They're ignored by the compiler so besides denoting the comment itself, little syntax is required. Do remember though to use comments sparingly, and never comment on the obvious. A variable person doesn't need a comment that says "this is a person"!

Instead, you should always aim to express your intent in the code itself. For example, a method that adds two numbers that is called `add()` clearly portrays what it does and probably doesn't require a comment to describe the method. There are some few exceptions to this rule, but generally, a comment is used as a fix for bad code that doesn't properly express its intent. Comments can never be as precise as well written code. Also, while code is always up to date, a comment will usually end up out of sync with the code. Obsolete comments are really dangerous, they are often the cause for severe confusion. Therefore, I usually say "comments lie" and never completely trust a comment. Code can't lie. One will always be One and Zero will always be zero. Code always does exactly what it claims it does.

There are two types of comments you can define in Java: multi-line comments and single line comments. Multi-line comments start with '`/*`' and end with '`*/'`. Between these two symbols, you can write as many lines of comments as you'd like.

A single line comment starts with '`//`' and lasts for the remainder of the line in the code. In general, I recommend that you always use multi-line comments, because if a single line comment is too long, it will be broken up when the code gets auto formatted, which might happen quite often in a development team. Now we create the default constructor for the class Person. Inside the default constructor I've added a comment. I'm also going to create the `hello()` method that we called in our test class. It receives one argument, which is the name of person we are 'saying' hello to.

Concatenating Strings

In our `hello()` method we are returning a `String`. However, the `String` we return will not always be the same. It will always start with "Hello", followed by the name of the person. We have to turn these two separate Strings into one `String` that we return. This process of adding Strings together is called **Concatenation**. To do this, we put a '`+`' between the two Strings, creating one larger `String`. Now our method will return "Hello" followed by the person's name. I've left a hidden bug in Example 6 that will cause our test to fail. See if you can find it.

Testing our Code and Fixing Bugs

Now let's execute our Test class. It failed as expected. The reason is that there is a mismatch between our expected value which was "`Hello Marcus`" and the actual value, "`HelloMarcus`". When we concatenated our two Strings we forgot to include a space between them. To fix this, all we have to do is add a space after the word "`Hello`".

```
public static String hello(String name) {
     return “Hello ” + name;
}
```
Example 7

Now when we execute our test, it's green and we've successfully completed this article!

Thanks for reading!