Basic Java Keywords Explained – Debriefing

Introduction

In this article from my free Java 8 course, I will review some of the basic Java keywords.

Debriefing

For review purposes, let's go back to our development environment and take a look at what we've learned already. Firstly, when we write our Java Class, the first line of our program is the package declaration. Usually it's the domain name in reverse order. While it's not mandatory to declare the package, I recommend that you always do it.

```
package com.marcusbiel.java8course;
```

Example 1

After the package declaration, we have our import statements. In this article, we're going to create a new `Name` class inside the subpackage '`java8course.attributes`'. Since this class is in a different package and we want to use it in our main class, we'll import it using the statement shown below in Example 2. Please note, I'm creating the `Name` class in a separate package for the sole purpose of demonstrating import statements.

I'd like to take some time to discuss import statements in a bit more detail. We've already discussed that we need an import statement is when we are using an Object from a class. When we import the class, we import the full class name, which includes the package name, followed by the name of the class. Here's an example:

```
package com.marcusbiel.java8course;

import com.marcusbiel.java8course.attributes.Name;

public class Person {

     private Name personName;

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

Example 2

There's also another way that we can create instance variables of class `Name`. When we create our instance variable, we can include the full class name appended by the name of our variable. We wouldn't need to import the class if we did this:

```
package com.marcusbiel.java8course;

public class Person {

     private com.marcusbiel.java8course.attributes.Name.personName;

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

Example 3

Firstly, from a readability standpoint, Example 3 is barely readable. On top of that, while you may not see any difference in this one line, having to repeatedly use the full class name to create instance variables is much longer to type out repeatedly.

The only time when you *have to* use the full class name rather than an import statement is when you have two different classes with the same short name, but the two classes are from different packages. For example if you had our class "com.marcusbiel.java8course.**attributes**.Name", but you also needed to use a class "com.marcusbiel.java8course.Name", you wouldn't be able to import both classes. So you can import one and use the full class name for the other.

Next we have our class definition, `public class Person`, followed by opening and closing brackets. The opening bracket defines the beginning of the code for the class, and the closing one marks the end. The keyword `public` means that any class in the same package, or in any other package for that matter, can see this class. Just like this `Person` class can see the `Name` class, that `Name` class is able to see the `Person` class.

Now, inside the `Person` class, we'll define a reference variable `personName` which is of type Name. To make this reference variable inaccessible to other classes, we define it as `private`. But since we want this field to be accessible to other classes, we're also going to create a public method `name()` which is easily accessible and will return this reference variable `personName`. Any class that wants to access the `personName` variable will have to call the `name()` method. There is no naming convention for this method, so we can name it whatever we want. The `name()` method's implementation should contain a return statement that returns the `personName` variable.

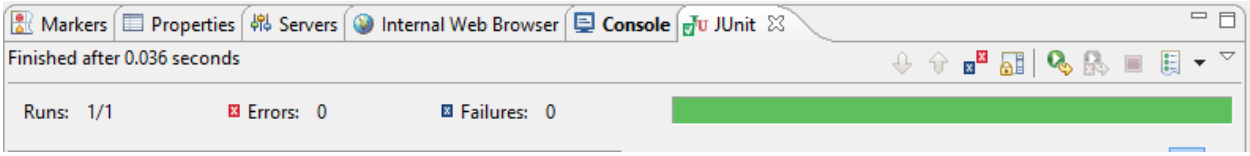
```
package com.marcusbiel.java8course;
```

```
import com.marcusbiel.java8course.attributes.Name;

public class Person {

	private Name personName;

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

	public Name name() {
		return personName;
	}
}
```

Example 4

We also have the *`helloWorld()`* method from our last iteration of the `Person` class. If you remember, we already have a class called `PersonTest` from our last article, where we defined the `helloWorld()` method as one of the arguments in an `assertEquals()` method.

Defining this test method gives us an opportunity to think about a good design for our program. Once we execute the Test class and the code has been correctly implemented, we should get a green bar that means our test case executed successfully.



Example 5

While testing, we are always focused on the green and red bars. The green bar means that our test passed and the red means that our test failed. You should also note that in the `PersonTest` class, I used the `@Test` annotation and the static keyword in the import statement of the `assertEquals()` method from the JUnit library. I won't go into much detail about this now, but it will be covered in a later article. For your reference, here is the `PersonTest` class:

```
package com.marcusbiel.java8course;

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

public class PersonTest {

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

Example 6

Thanks for reading!