Jb0190: Java OOP: Using the System and PrintStream Classes[*]

Richard Baldwin

Abstract

Take a preliminary look at the complexity of OOP by examining some aspects of the System and PrintStream classes.

1 Table of Contents

2 Preface

This module takes a preliminary look at the complexity of OOP by examining some aspects of the **System** and **PrintStream** classes.

2.1 Viewing tip

I recommend that you open another copy of this module in a separate browser window and use the following links to easily find and view the listings while you are reading about them.

2.1.1 Listings

- Listing 1 (p. 2) . The program named hello1.
- Listing 2 (p. 3) . Display the string "Hello World".

[*]Version 1.1: Nov 18, 2012 11:06 am -0600
[†]http://creativecommons.org/licenses/by/3.0/

3 Introduction

This lesson introduces you to the use of the **System** and **PrintStream** classes in Java. This is our first introduction to the complexity that can accompany the **OOP** paradigm. It gets a little complicated, so you might need to pay special attention to the discussion.

4 Discussion

What does the main method do?

The **main** method in the controlling class of a Java application controls the flow of the program.

The **main** method can also access other classes along with the variables and methods of those classes and of objects instantiated from those classes.

The hello1 Application

Listing 1 (p. 2) shows a simple Java application named **hello1** .

(By convention, class definitions should begin with an upper-case character. However, the original version of this module was written and published in 1997, before that convention was firmly established.)

Listing 1: The program named hello1.

```
    /*File hello1.java Copyright 1997, R.G.Baldwin
*********************************************************/
class hello1 { //define the controlling class
  //define main method
  public static void main(String[] args){
    //display text string
    System.out.println("Hello World");
  }//end main
}//End hello1 class.  No semicolon at end of Java class.
```

Does this program Instantiate objects?

This is a simple example that does not instantiate objects of any other class.

Does this program access another class?

However, it does access another class. It accesses the **System** class that is provided with the Java development kit. *(The* ***System*** *class will be discussed in more detail in a future module.)*

The variable named *out*

The variable named **out** , referred to in Listing 1 (p. 2) as **System.out** , is a *class variable* of the **System** class. *(A class variable is a variable that is declared to be static.)*

Recall that a class variable can be accessed without a requirement to instantiate an object of the class. As is the case with all variables, the class variable must be of some specific type.

Primitive variables vs. reference variables

A class variable may be a *primitive variable* , which contains a primitive value, or it may be a *reference variable* , which contains a reference to an object.

(I'll have more to say about the difference between primitive variables and reference variables in a future module.)

The variable named **out** in this case is a *reference variable* , which refers to an object of another type.

Accessing class variables

You access class variables or class methods in Java by joining the name of the class to the name of the variable or method with a period.

NOTE: System.out

accesses the class variable named **out** in the Java class named **System** .

The PrintStream class

Another class that is provided with the Java development kit is the **PrintStream** class. The **PrintStream** class is in a package of classes that are used to provide stream input/output capability for Java.

What does the out variable refer to?

The **out** variable in the **System** class refers to *(points to)* an instance of the **PrintStream** class *(a **PrintStream** object),* which is automatically instantiated when the **System** class is loaded into the application.

We will be discussing the **PrintStream** class along with a number of other classes in detail in a future module on input/output streams, so this is not intended to be an exhaustive discussion.

The println method

The **PrintStream** class has an *instance method* named **println** , which causes its argument to be displayed on the standard output device when it is called.

(Typically, the standard output device is the command-line window. However, it is possible to redirect it to some other device.)

Accessing an instance method

The method named **println** can be accessed by joining a **PrintStream** object's reference to the name of the method using a period.

Thus, *(assuming that the standard output device has not been redirected)* , the statement shown in Listing 2 (p. 3) causes the string "Hello World" *(without the quotation marks)* to be displayed in the command-line window.

Listing 2: Display the string "Hello World".

```
System.out.println("Hello World");
```

This statement calls the **println** method of an object instantiated from the **PrintStream** class, which is referred to *(pointed to)* by the variable named **out** , which is a *class variable* of the **System** class.

Read the previous paragraph very carefully. As I indicated when I started this module, this is our first introduction to the complexity that can result from use of the OOP paradigm. *(It can get even more complicated.)* If this is not clear to you, go back over it and think about it until it becomes clear.

5 A word about class variables

How many instances of a class variable exist?

The runtime system allocates a class variable only once no matter how many instances *(objects)* of the class are instantiated.

All objects of the class share the same physical memory space for the class variable.

If a method in one object changes the value stored in the class variable, it is changed insofar as all of the objects are concerned. *(This is about as close to a global variable as you can get in Java.)*

Accessing a class variable

You can use the name of the class to access class variables by joining the name of the class to the name of the variable using a period.

You can also access a class variable by joining the name of a reference variable containing an object's reference to the name of the variable using a period as the joining operator.

Referencing object methods via class variables

Class variables are either primitive variables or reference variables. *(Primitive variables contain primitive values and reference variables contain references to objects.)*

A referenced object may provide methods to control the behavior of the object. In Listing 2 (p. 3) , we accessed the **println** method of an object of the **PrintStream** class referred to by the class variable named **out** .

Instance variables and methods

As a side note, in addition to class variables, Java provides *instance variables* and *instance methods* . Every instance of a class has its own set of instance variables. You can only access instance variables and instance methods through an object of the class.

6 Run the program

I encourage you to copy the code from Listing 1 (p. 2) . Compile the code and execute it. Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

7 Miscellaneous

This section contains a variety of miscellaneous information.

NOTE: **Housekeeping material**

- Module name: Jb0190: Java OOP: Using the System and PrintStream Classes
- File: Jb0190.htm
- Originally published: 1997
- Published at cnx.org: November 18, 2012

NOTE: **Disclaimers:** **Financial** : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.

I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module.

In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-