

# JI0010: OOP SELF-ASSESSMENT TEST, PART 1\*

R.G. (Dick) Baldwin

This work is produced by OpenStax-CNX and licensed under the  
Creative Commons Attribution License 3.0<sup>†</sup>

## Abstract

Part 1 of a self-assessment test designed to help you determine how much you know about the fundamentals of object-oriented programming using Java.

## 1 Table of Contents

- Preface (p. 1)
- Questions (p. 1)
  - 1 (p. 1) , 2 (p. 2) , 3 (p. 2) , 4 (p. 2) , 5 (p. 2) , 6 (p. 2) , 7 (p. 2) , 8 (p. 3) , 9 (p. 3) , 10 (p. 3)
- Listings (p. 4)
- Miscellaneous (p. 4)
- Answers (p. 4)

## 2 Preface

This module is Part 1 of a self-assessment test designed to help you determine how much you know about the fundamentals of object-oriented programming using Java.

The test consists of a series of questions with answers and explanations of the answers. The answers (p. 4) to the questions, and the explanations of those answers are located ( *in reverse order* ) at the end of module.

The questions and the answers are connected by hyperlinks to make it easy for you to navigate from the question to the answer and back.

## 3 Questions

### 3.1 Question 1 .

**Source File Names:** True or false? All Java source files must end with the extension `.class`  
Answer and Explanation (p. 8)

---

\*Version 1.4: Nov 16, 2012 11:25 am +0000

<sup>†</sup><http://creativecommons.org/licenses/by/3.0/>

### 3.2 Question 2 .

**Separate source File Requirements:** True or false? A source file may contain an unlimited number of non-public class definitions.

Answer and Explanation (p. 7)

### 3.3 Question 3 .

**Class file names:** True or false? If a source file includes a public class, the class name must match the file name (*without the extension*) .

Answer and Explanation (p. 7)

### 3.4 Question 4 .

True or false: Each source file that you compile will produce one file with an extension **.class**

Answer and Explanation (p. 7)

### 3.5 Question 5 .

**Required programming elements:** True or false. If any of the following elements are included in the source file, they must appear in the following order.

- A. package declaration
- B. import declarations
- C. class definitions

Answer and Explanation (p. 7)

### 3.6 Question 6 .

**Import directives:** True or false? The compiler does not require you to use import directives.

Answer and Explanation (p. 6)

### 3.7 Question 7 .

**Import directives:** What output is produced by the program shown in Listing 1 (p. 2) ?

- A. A compiler error
- B. A runtime error
- C. OK

**Listing 1: Code for Question 7.**

```
//File Q001_07.java
import java.awt.*;
public class Q001_07{
    public static void main
        (String args[]){
        Button aButton =
            new Button("Button");
        Label aLabel =
            new Label("Label");
        System.out.println("OK");
    }//end main
}//end class Q001_07
```

Answer and Explanation (p. 5)

### 3.8 Question 8 .

**goto and const:** True or false? As in C and C++, **goto** and **const** are keywords that are actively used in Java.

Answer and Explanation (p. 5)

### 3.9 Question 9 .

Which of the following characters may be used as the first character in a Java identifier (*identify all that may be used*) ?

- A. x (the letter x)
- B. 4 (the number 4)
- C. \$ (the dollar sign character)
- D. \_ (the underscore character)

Answer and Explanation (p. 5)

### 3.10 Question 10 .

**Bonus question.** The following question is considerably more difficult than the previous nine questions, and is included here to challenge you if the previous nine questions have been too easy.

**Access Control:** What output is produced by compiling and running the program shown in Listing 2 (p. 3) ? Note that the instance variable named **x** is declared **private** .

- A. A compiler error
- B. A runtime error
- C. 15

**Listing 2: Code for Question 10.**

```
//File Q001_10.java
class Q001_10{
    public static void main(
        String args[]){
        AClass ref1 = new AClass(5);
        AClass ref2 = new AClass(10);
        System.out.println(
            ref1.add(ref2));
    }//end main()
}//end class definition

class AClass{
    private int x;

    AClass(int x){//constructor
        this.x = x;
    }// end constructor
```

```
int add(AClass ref){
    return x + ref.x;
} //end add()

} //end class AClass
```

Answer and Explanation (p. 4)

## 4 Listings

- Listing 1 (p. 2) . Code for Question 7.
- Listing 2 (p. 3) . Code for Question 10.
- Listing 3 (p. 6) . Listing for Answer 6.
- Listing 4 (p. 7) . Listing for Answer 2.

## 5 Miscellaneous

This section contains a variety of miscellaneous information.

**NOTE: Housekeeping material**

- Module name: Ji0010: OOP Self-Assessment Test, Part 1
- File: Ji0010.htm
- Originally published: August 9, 2000
- Published at cnx.org: November 15, 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.

## 6 Answers

### 6.1 Answer 10 .

The answer is C. The output produced by compiling and running the program is 15.

Back to Question 10 (p. 3)

### 6.1.1 Explanation 10

Even though the instance variable named `x` is private, the variable in the object referred to by `ref2` can be accessed by the object referred to by `ref1`.

The important point is that access modifiers dictate which classes – not which instances of a class – may access features. Thus, a private instance variable can be accessed by *any instance* of the class in which the private instance variable is defined.

## 6.2 Answer 9 .

The following characters may be used as the first character of a Java identifier:

- A. `x` (the letter `x`)
- C. `$` (the dollar sign character)
- D. `_` (the underscore character)

B (*the number 4*) cannot be used as the first character of a Java identifier.

Back to Question 9 (p. 3)

### 6.2.1 Explanation 9

In Java, a legal identifier is a sequence of Unicode letters and digits of unlimited length.

The first character must be a letter.

All subsequent characters must be letters or numerals from any alphabet that Unicode supports.

The underscore character (`_`) and the dollar sign (`$`) are considered to be letters and may be used as any character including the first one.

## 6.3 Answer 8 .

False.

Back to Question 8 (p. 3)

### 6.3.1 Explanation 8

`goto` and `const` are reserved words, but are not actively used in Java. Although they have no meaning in Java, you cannot use them as identifiers. Apparently Sun did this to avoid confusion, but I have never seen an official explanation (*but I admit that I haven't searched very hard for an explanation*).

## 6.4 Answer 7 .

The answer is C. This program will produce the output OK.

Back to Question 7 (p. 2)

### 6.4.1 Explanation 7

Each import directive can specify the package containing a single class file.

Alternatively, an import directive can use the wildcard character (`*`) to "import" all of the class files in a specified package. This program uses the following wildcard version of the import directive to import all of the class files in the package named `awt`.

```
import java.awt.*;
```

This includes both `Button` and `Label`. Therefore, it is not necessary to provide fully-qualified path and package information when referring to either of these classes in the program code.

A word of caution is in order, however. The purpose of packages is to resolve name clashes among class files having the same names. If you find yourself referring to two or more different class files that have the

same name, they must be stored in different packages, and you cannot use import directives for those class files. In that case, you must provide a fully-qualified path and package name each time you refer to one of those classes.

It is possible that you could use the wildcard character to import two different packages that contain one or more duplicate class file names. This could lead to problems.

Therefore, the safest approach is to avoid the use of the wildcard character altogether and to use a separate import directive for each class that you need to refer to in the program. (*This will also cause your program to be more self-documenting.*) I have worked as a Java consultant for a couple of companies whose internal programming standards prohibit the use of the wildcard character in import directives.

## 6.5 Answer 6 .

True

[Back to Question 6 \(p. 2\)](#)

### 6.5.1 Explanation 6

The import directive is provided strictly as a convenience to the programmer. The compiler does not require you to use import directives.

However, you can often save yourself a lot of extra work by using import directives. If you don't use import directives, every reference to a class must provide a fully qualified path and package name for the package that contains the class file.

For example, the program shown in Listing 3 (p. 6) contains references to two classes: **Button** and **Label** . An import directive is used to inform the compiler where it can find the class file for the **Button** class. No such import directive is provided for the **Label** class.

As a result, the code required to refer to the **Label** class must specify the package that contains the class file for the **Label** class. Otherwise, the program cannot be compiled.

As you can see, considerably more typing is required to refer to the **Label** class than is required to refer to the **Button** class.

#### Listing 3: Listing for Answer 6.

```
//File Q001_06.java
import java.awt.Button;

public class Q001_6{
    public static void main
        (String args[]){

        Button aButton =
            new Button("Button");

        java.awt.Label aLabel =
            new java.awt.Label("Label");

        System.out.println(
            aButton.getLabel());
        System.out.println(
            aLabel.getText());
    } //end main
} //end class
```

## 6.6 Answer 5 .

True.

Back to Question 5 (p. 2)

## 6.7 Answer 4 .

False.

Back to Question 4 (p. 2)

### 6.7.1 Explanation 4

One class file is produced for each class definition in your program regardless of the number of source files and regardless of whether they are top-level classes or inner classes. Even each anonymous inner class produces an output class file, and the name given to the class file for the anonymous inner class is created by the system.

## 6.8 Answer 3 .

True. However, see the qualification below.

Back to Question 3 (p. 2)

### 6.8.1 Explanation 3

According to *The Complete Java 2 Certification Study Guide* , by Roberts, Heller, and Ernest, *"This is not actually a language requirement, but is an implementation requirement of many compilers, including the reference compilers from Sun. It is therefore unwise to ignore this convention."*

## 6.9 Answer 2 .

True. A source file may contain an unlimited number of non-public class definitions. In fact, the source file may contain one public class definition and an unlimited number of non-public class definitions. However, the rules change if two or more of the classes are declared to be public.

Back to Question 2 (p. 2)

### 6.9.1 Explanation 2

While it is true that a source file may contain an unlimited number of non-public class definitions, such is not the case if the classes are declared **public** . Each **public** class and interface must be contained in its own source file.

The fact that each **public** class requires its own source file is easy enough to demonstrate. The Java application in Listing 4 (p. 7) fails to compile with a compiler error to the effect that *"Public class Dummy must be defined in a file called "Dummy.java"."*

If the **public** modifier is removed from the definition of the class named **Dummy** , leaving only one public class in the source file, the application will compile and execute successfully.

**Listing 4: Listing for Answer 2.**

```
    public class Q001_2{
    public static void main
                (String args[]){
        System.out.println("OK");
    }//end main
} //end class

public class Dummy{
    //empty class definition
} //end class definition

class AnotherDummy{
    //empty class definition
} //end class definition
```

## 6.10 Answer 1 .

False. Java source files must not end with the extension **.class** . Rather, all Java source files must end with the extension **.java** (except under some advanced circumstances that are beyond the scope of this module).

[Back to Question 1 \(p. 1\)](#)

### 6.10.1 Explanation 1

When you create Java source files, you need to use a text editor that will produce clean character codes (sometimes referred to as *ASCII codes*) .

The output from the editor must not contain any control characters indicating **bold** , *italics* , underline, etc.

Just about any text editor can be used for this purpose. Various text editors are available on the web that can make the creation of source code easier. These editors provide features such as causing the various parts of the program to appear in different colors during the editing process , or automatically indenting each line of source code according to generally accepted indentation standards.

Some of the available editors will allow you to compile and execute the Java program from within the editor program, which can be a real time saver during development. Editors of this type are often referred to as Integrated Development Environments (IDE).

Back to the point of the discussion, as mentioned above, the file name of all source code files must end with the extension **.java** . If your editor doesn't put it there, you will need to rename the file manually to create the correct extension.

When you successfully compile the program, one or more files will be produced by the compiler having the extension **.class**

-end-