Jb0200r: Review: Variables*

Richard Baldwin

This work is produced by The Connexions Project and licensed under the
Creative Commons Attribution License †

Abstract

This module contains review questions and answers keyed to the module titled Jb0200: Java OOP:
Variables.

1 Table of Contents

2 Preface

This module contains review questions and answers keyed to the module titled Jb0200: Java OOP: Variables
[1] .

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 again.

3 Questions

3.1 Question 1 .

Write a Java application that reads characters from the keyboard until encountering the # character. Echo
each character to the screen as it is read. Terminate the program when the user enters the # character.
Answer 1 (p. 12)

---

*Version 1.4: Nov 23, 2012 11:02 am -0600
†http://creativecommons.org/licenses/by/3.0/
[1]http://cnx.org/content/m45150/latest/

3.2 Question 2

What is the common name for the Java program element that is used to contain data that changes during
the execution of the program?
    Answer 2 (p. 12)

3.3 Question 3

What must you do to make a variable available for use in a Java program?
    Answer 3 (p. 12)

3.4 Question 4

True or false? In Java, you are required to initialize the value of all variables when they are declared.
    Answer 4 (p. 12)

3.5 Question 5

Show the proper syntax for declaring two variables and initializing one of them using a single Java statement.
    Answer 5 (p. 12)

3.6 Question 6

True or false? The Java compiler will accept statements with type mismatches provided that a suitable type
conversion can be implemented by the compiler at compile time.
    Answer 6 (p. 12)

3.7 Question 7

Show the proper syntax for the declaration of a variable of type **String[]** in the argument list of the **main**
method of a Java program and explain its purpose.
    Answer 7 (p. 12)

3.8 Question 8

Describe the purpose of the type definition in Java.
    Answer 8 (p. 11)

3.9 Question 9

True or false? Variables of type **int** can contain either signed or unsigned values.
    Answer 9 (p. 11)

3.10 Question 10

What is the important characteristic of type definitions in Java that strongly supports the concept of
platform independence of compiled Java programs?
    Answer 10 (p. 11)

3.11 Question 11

What are the two major categories of types in Java?
    Answer 11 (p. 11)

3.12 Question 12

What is the maximum number of values that can be stored in a variable of a *primitive* type in Java?
    Answer 12 (p. 11)

3.13 Question 13

List the *primitive* types in Java.
    Answer 13 (p. 11)

3.14 Question 14

True or false? Java stores variables of type **char** according to the 8-bit extended ASCII table.
    Answer 14 (p. 11)

3.15 Question 15

True or false? In Java, the name of a *primitive* variable evaluates to the value stored in the variable.
    Answer 15 (p. 11)

3.16 Question 16

True or false? Variables of *primitive* data types in Java are true objects.
    Answer 16 (p. 11)

3.17 Question 17

Why do we care that variables of *primitive* types are not true objects?
    Answer 17 (p. 10)

3.18 Question 18

What is the name of the mechanism commonly used to convert variables of *primitive* types to true objects?
    Answer 18 (p. 10)

3.19 Question 19

How can you tell the difference between a *primitive* type and a *wrapper* for the primitive type when the two are spelled the same?
    Answer 19 (p. 10)

3.20 Question 20

Show the proper syntax for declaring a variable of type **double** and initializing its value to 5.5.
    Answer 20 (p. 10)

3.21 Question 21

Show the proper syntax for declaring a variable of type **Double** and initializing its value to 5.5.
    Answer 21 (p. 10)

3.22 Question 22

Show the proper syntax for extracting the value from a variable of type **Double** .
    Answer 22 (p. 9)

3.23 Question 23

True or false? In Java, the name of a reference variable evaluates to the address of the location in memory where the variable is stored.

Answer 23 (p. 9)

3.24 Question 24

What is a *legal identifier* in Java?

Answer 24 (p. 9)

3.25 Question 25

What are the rules for variable names in Java?

Answer 25 (p. 9)

3.26 Question 26

What is meant by the *scope* of a Java variable?

Answer 26 (p. 9)

3.27 Question 27

What are the four possible *scope* categories for a Java variable?

Answer 27 (p. 9)

3.28 Question 28

What is a member variable?

Answer 28 (p. 8)

3.29 Question 29

Where are *local variables* declared in Java?

Answer 29 (p. 8)

3.30 Question 30

What is the scope of a local variable in Java?

Answer 30 (p. 8)

3.31 Question 31

What defines a *block* of code in Java?

Answer 31 (p. 8)

3.32 Question 32

What is the scope of a variable that is declared within a block of code that is defined within a method and which is a subset of the statements that make up the method?

Answer 32 (p. 8)

3.33 Question 33

What is the scope of a variable declared within the initialization clause of a *for* statement in Java? Provide an example code fragment.

Answer 33 (p. 8)

3.34 Question 34

What are *method parameters* and what are they used for?

Answer 34 (p. 8)

3.35 Question 35

What is the scope of a *method parameter* ?

Answer 35 (p. 8)

3.36 Question 36

What are *exception handler parameters* ?

Answer 36 (p. 7)

3.37 Question 37

Write a Java application that illustrates member variables *(class and instance)* , local variables, and method parameters.

Answer 37 (p. 7)

3.38 Question 38

True or false? Member variables in a Java class can be initialized when the class is defined.

Answer 38 (p. 7)

3.39 Question 39

How are *method parameters* initialized in Java?

Answer 39 (p. 7)

4 Listings

What is the meaning of the following two images?

This image was inserted here simply to insert some space between the questions and the answers to keep them from being visible on the screen at the same time.

The image is also an example of the kinds of things that we do in my course titled ITSE 2321, Object-Oriented Programming.

This image was also inserted for the purpose of inserting space between the questions and the answers.

5 Answers

5.1 Answer 39

Method parameters are initialized by the values passed to the method.
   Back to Question 39 (p. 5)

5.2 Answer 38

True.
   Back to Question 38 (p. 5)

5.3 Answer 37

See the application named **member1** in this module [2] for an example of such an application.
   Back to Question 37 (p. 5)

5.4 Answer 36

Exception handler parameters are arguments to exception handlers, which will be discussed in a future module.
   Back to Question 36 (p. 5)

---

[2]http://cnx.org/content/m45150/latest/#Listing_8

5.5 Answer 35

The scope of a method parameter is the entire method for which it is a parameter.

Back to Question 35 (p. 5)

5.6 Answer 34

 Method parameters are the formal arguments of a method. Method parameters are used to pass values into and out of methods.

Back to Question 34 (p. 5)

5.7 Answer 33

Java treats the scope of a variable declared within the initialization clause of a *for* statement to be limited to the total extent of the *for* statement. A sample code fragment follows where **cnt** is the variable being discussed:

NOTE:

```
    for(int cnt = 0; cnt < max; cnt++){
  //do something
}//end of
```

Back to Question 33 (p. 5)

5.8 Answer 32

In Java, the scope can be reduced by placing it within a block of code within the method. The *scope* extends from the point at which it is declared to the end of the block of code in which it is declared.

Back to Question 32 (p. 4)

5.9 Answer 31

A block of code is defined by enclosing it within curly brackets as shown below

   { ... } .

Back to Question 31 (p. 4)

5.10 Answer 30

The *scope* of a local variable extends from the point at which it is declared to the end of the block of code in which it is declared.

Back to Question 30 (p. 4)

5.11 Answer 29

In Java, *local variables* are declared within the body of a method or constructor, or within a block of code contained within the body of a method or constructor.

Back to Question 29 (p. 4)

5.12 Answer 28

A *member variable* is a member of a class (*class* variable) or a member of an object instantiated from that class (*instance* variable). It must be declared within a class, but not within the body of a method or constructor of the class.

Back to Question 28 (p. 4)

5.13 Answer 27

The *scope* of a variable places it in one of the following four categories:

- member variable
- local variable
- method parameter
- exception handler parameter

Back to Question 27 (p. 4)

5.14 Answer 26

The *scope* of a Java variable is the block of code within which the variable is accessible.
    Back to Question 26 (p. 4)

5.15 Answer 25

The rules for Java variable names are as follows:

- Must be a legal Java identifier consisting of a series of *Unicode* characters.
- Must not be the same as a Java *keyword* and must not be *true* or *false.*
- Must not be the same as another variable whose declaration appears in the same scope.

Back to Question 25 (p. 4)

5.16 Answer 24

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. In addition, the underscore character (_) and the dollar sign ($) are considered letters and may be used as any character including the first one.
    Back to Question 24 (p. 4)

5.17 Answer 23

False. The name of a reference variable evaluates to either null, or to information that can be used to access an object whose reference has been stored in the variable.
    Back to Question 23 (p. 4)

5.18 Answer 22

Later versions of Java support either syntax shown in Listing 1 (p. 9) .

Listing 1: Listing for Answer 22.

```
class test{
  public static void main(String[] args){
    Double var1 = 5.5;
    double var2 = var1.doubleValue();
    System.out.println(var2);

    double var3 = var1;
```

```
    System.out.println(var3);
  }//end main
}//end class test
```

Back to Question 22 (p. 3)

5.19 Answer 21

The proper syntax for early versions of Java is shown below. Note the upper-case **D** . Also note the instantiation of a new object of type **Double** .

NOTE:

```
Double myWrappedData = new Double(5.5);
```

Later versions of Java support the following syntax with the new object of type **Double** being instantiated automatically:

NOTE:

```
Double myWrappedData = 5.5;
```

Back to Question 21 (p. 3)

5.20 Answer 20

The proper syntax is shown below. Note the lower-case **d** .

NOTE:

```
double myPrimitiveData = 5.5;
```

Back to Question 20 (p. 3)

5.21 Answer 19

The name of the *primitive* type begins with a lower-case letter and the name of the *wrapper* type begins with an upper-case letter such as **double** and **Double** . Note that in some cases, however, that they are not spelled the same. For example, the **Integer** class is the wrapper for type **int** .

Back to Question 19 (p. 3)

5.22 Answer 18

Wrapper classes

Back to Question 18 (p. 3)

5.23 Answer 17

This has some ramifications as to how variables can be used *(passing to methods, returning from methods, etc.)* . For example, all variables of *primitive* types are passed by value to methods meaning that the code in the method only has access to a copy of the variable and does not have the ability to modify the variable.

Back to Question 17 (p. 3)

5.24 Answer 16

False. Primitive data types in Java *(int, double, etc.)* are not true objects.
Back to Question 16 (p. 3)

5.25 Answer 15

True.
Back to Question 15 (p. 3)

5.26 Answer 14

False. The **char** type in Java is a 16-bit Unicode character.
Back to Question 14 (p. 3)

5.27 Answer 13

- byte
- short
- int
- long
- float
- double
- char
- boolean

Back to Question 13 (p. 3)

5.28 Answer 12

Primitive types contain a single value.
Back to Question 12 (p. 3)

5.29 Answer 11

Java supports both *primitive* types and *reference* (or object) types.
Back to Question 11 (p. 2)

5.30 Answer 10

In Java, a variable of a specified type is represented exactly the same way regardless of the platform on which the application or applet is being executed.
Back to Question 10 (p. 2)

5.31 Answer 9

False. In Java, all variables of type **int** contain signed values.
Back to Question 9 (p. 2)

5.32 Answer 8

All variables in Java must have a defined *type* . The definition of the *type* determines the set of values that can be stored in the variable and the operations that can be performed on the variable.
Back to Question 8 (p. 2)

5.33 Answer 7

The syntax is shown in boldface below:

NOTE: public static void main(**String[] args**)

In this case, the type of variable declared is an array of type **String** named **args** *(type String[])* . The purpose of the **String** array variable in the argument list is to make it possible to capture arguments entered on the command line.

Back to Question 7 (p. 2)

5.34 Answer 6

False. Fortunately, Java provides very strict type checking and generally refuses to compile statements with type mismatches.

Back to Question 6 (p. 2)

5.35 Answer 5

NOTE:

```
int firstVariable, secondVariable = 10;
```

Back to Question 5 (p. 2)

5.36 Answer 4

False: In Java, it is possible to initialize the value of a variable when it is declared, but initialization is not required. *(Note however that in some situations, the usage of the variable may require that it be purposely initialized.)* .

Back to Question 4 (p. 2)

5.37 Answer 3

To use a variable, you must notify the compiler of the name and the type of the variable *(declare the variable)*.

Back to Question 3 (p. 2)

5.38 Answer 2

variable

Back to Question 2 (p. 2)

5.39 Answer 1

Listing 2: Listing for Answer 1.

```
    /*File simple4.java
This application reads characters from the keyboard until
encountering the # character and echoes each character to
the screen. The program terminates when the user enters
the # character.
*********************************************************/
```

```
class simple4 { //define the controlling class
  public static void main(String[] args)
                                throws java.io.IOException {
    int ch1 = 0;
    System.out.println(
                     "Enter some text, terminate with #");
    while( (ch1 = System.in.read() ) != '#')
      System.out.print((char)ch1);
    System.out.println("Goodbye");
  }//end main
}//End simple4 class.
```

Back to Question 1 (p. 1)

6 Miscellaneous

This section contains a variety of miscellaneous information.

NOTE: **Housekeeping material**

- Module name: Jb0200r: Review: Variables
- File: Jb0200r.htm
- Published: November 23, 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-