Ji0020: OOP Self-Assessment Test, Part 2 - Old version*

Richard Baldwin

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

Abstract

Part 2 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. 3) , 8 (p. 3) , 9 (p. 3) , 10 (p. 3)
- Listings (p. 4)
- Miscellaneous (p. 4)
- Answers (p. 4)

2 Preface

This module is one part 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 .

Primitive Data Types: The following list purports to show the primitive data types and their representative sizes in bits. Identify which lines, if any, contain errors.

1. boolean, 1

*Version 1.2: Dec 1, 2012 12:34 pm -0600
†http://creativecommons.org/licenses/by/3.0/

2. char, 8
3. byte, 8
4. short, 16
5. int, 32
6. long, 64
7. float, 32
8. double, 64

Answer and Explanation (p. 8)

3.2 Question 2 .

boolean states: Which of the following are the possible states of a **boolean** variable *(list all)* ?

- A. true
- B. uncertain
- C. false

Answer and Explanation (p. 8)

3.3 Question 3 .

Range of integer types: The following list purports to show the numeric range of the integer types in terms of a power of two. Identify which lines, if any, contain errors. *(The syntax 2eX means 2 raised to the X power.)*

1. byte, min = -2e7, max = (2e7)-1
2. short, min = -2e15, max = (2e15)-1
3. int, min = -2e31, max = (2e31)-1
4. long, min = -2e63, max = (2e63)-1

Answer and Explanation (p. 7)

3.4 Question 4 .

Signed versus unsigned: True or false? All of the integral types, including **char** are signed.
Answer and Explanation (p. 7)

3.5 Question 5 .

Character representation: True or false? Characters in Java are represented by the 8-bit Extended ASCII character set.
Answer and Explanation (p. 6)

3.6 Question 6 .

Unicode to ASCII relationship: What is the relationship between the 16-bit Unicode character set and the 7-bit ASCII character set.
Answer and Explanation (p. 6)

3.7 Question 7 .

Invalid numeric values: As a result of certain arithmetic operations *(such as division by zero)* , certain primitives can take on bit patterns that do not represent valid numeric values. These possibilities are represented by symbolic constants in the various classes. Which, if any, of the following are not valid symbolic constants?

- A. Integer.NaN
- B. Integer.NEGATIVE_INFINITY
- C. Integer.POSITIVE_INFINITY
- D. Float.NaN
- E. Float.NEGATIVE_INFINITY
- F. Float.POSITIVE_INFINITY
- G. Double.NaN
- H. Double.NEGATIVE_INFINITY
- I. Double.POSITIVE_INFINITY

Answer and Explanation (p. 6)

3.8 Question 8 .

Literal values: Which, if any, of the following are not valid literal values for type **char** ?

- A. '\u1632'
- B. '\t'
- C. "a"
- D. 'end'

Answer and Explanation (p. 5)

3.9 Question 9 .

Escape characters: Which of the following are valid escape characters in Java?

- A. '/n'
- B. '/r'
- C. '/t'
- D. '/b'
- E. '/f'
- F. '/"
- G. '/"'
- H. '//'

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.

Bit manipulations: What output is produced by the program shown in Listing 1 (p. 4) ?

- A. 1
- B. -1
- C. 2

- D. -2

Listing 1: Listing for Question 10.

```
 class Q002_10{
public static void main(String args[]){
  int x = 1;
  int y = ~x + 1;
  System.out.println(y);
}//end main()
}//end class definition
```

Answer and Explanation (p. 4)

4 Listings

- Listing 1 (p. 4) . Listing for Question 10.
- Listing 2 (p. 7) . Listing for Answer 3.
- Listing 3 (p. 8) . 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 2
- File: Ji0020.htm
- Originally published: February 20, 2001
- Published at cnx.org: November 29, 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 B, -1.

Back to Question 10 (p. 3)

6.1.1 Explanation 10

From http://docs.oracle.com/javase/tutorial/java/nutsandbolts/op3.html [1] ,

The unary bitwise complement operator " ~ " inverts a bit pattern; it can be applied to any of the integral types, making every "0" a "1" and every "1" a "0". For example, a byte contains 8 bits; applying this operator to a value whose bit pattern is "00000000" would change its pattern to "11111111" .

If you invert all of the bits in the binary twos complement representation of the decimal value 1, the decimal value of the resulting bit pattern is -2. If you then add decimal 1 to that, the result is -1.

In fact, this is how you convert a positive twos complement binary value to a negative twos complement binary value *(or vice versa)* . First you invert all of the bits. Then you add 1.

6.2 Answer 9 .

None are valid escape characters.

Back to Question 9 (p. 3)

6.2.1 Explanation 9

Escape characters are constructed with a backslash character instead of a forward slash character. Replace the forward slash characters with backslash characters and they would all be valid.

Here is a chart showing the usage of each of the escape characters for the chart in Question 9 (p. 3)

- A. Newline
- B. Return
- C. Tab
- D. Backspace
- E. Formfeed
- F. Apostrophe or single quote
- G. Double quote
- H. Backslash

6.3 Answer 8 .

C and D are not valid literal values for type **char** .

Back to Question 8 (p. 3)

6.3.1 Explanation 8

A. '\u1632' is a valid literal value of type **char** . This is how you can express Unicode characters in hexadecimal format. For example, this is how you can express Unicode characters that do not appear on the keyboard that you use to write your source code.

B. '\t' is also a valid literal value of type **char** . This is the so-called *escape character* that represents a *Tab* . There are about eight such escape characters supported by Java that represent various things like *Tab* , *Newline* , *Return* , *Backspace* , etc.

C. "a" is not a valid literal of type **char** . Rather, this is a **String** literal.

In Java, unlike some other languages, quotation marks (") and apostrophes (') have entirely different meanings.

Zero or more characters surrounded by quotation marks are interpreted by the compiler to constitute a **String** literal.

Any single character surrounded by apostrophes is interpreted by the compiler to be a **char** literal. When confronted with such a literal, the compiler produces the numeric value that represents that character.

[1] http://docs.oracle.com/javase/tutorial/java/nutsandbolts/op3.html

For example, the code fragment in the following note box will cause the value 65 to be displayed because the numeric value that represents the upper-case A in the Unicode character set is 65.

NOTE:

```
    int data = 'A';
System.out.println(data);
```

D. 'end' has no meaning in Java. Unlike some other languages, such as Python, you cannot interchange quotation marks and apostrophes when creating strings. For example, the following statement will produce the compiler error "Invalid character constant"
System.out.println('end');

6.4 Answer 7 .

A, B, and C are not valid symbolic constants.
Back to Question 7 (p. 3)

6.4.1 Explanation 7

As a result of arithmetic operations *(such as division by zero)* , both **float** and **double** can take on bit patterns that do not represent valid numeric values. These possibilities are represented by symbolic constants in the **Float** and **Double** classes. However, no such possibility exists for the **Integer** class.

An object of the **Integer** class is intended to encapsulate an **int** value. There are no operations that you can perform with an **int** that results in a bit pattern that does not represent a valid numeric value.

While it is possible to end up with erroneous **int** values as a result of some arithmetic operations, they are still valid numeric values and so there is no need for the kind of symbolic constants described for **float** and **double** in this question.

6.5 Answer 6 .

The Unicode character set matches the 7-bit ASCII set for the least significant seven bits. In other words, if the nine most significant bits of a Unicode character are all zero, then the encoding is the same as 7-bit ASCII.
Back to Question 6 (p. 2)

6.5.1 Explanation 6

This is very convenient for those of us who have been writing programs in other languages for some time and who are making the transition to Unicode. For example, the character whose decimal value is 65 is an upper case A in both ASCII and Unicode.

As it turns out, for those of us who program using the U.S./English alphabet, the change from Extended ASCII to Unicode isn't very significant. We can pretty much continue doing what we have been doing for years.

However, those who program using alphabets from other spoken languages now have the ability to reflect those languages in the character representations contained in their programs.

6.6 Answer 5 .

False
Back to Question 5 (p. 2)

6.6.1 Explanation 5

For many years, programming languages have used the 7-bit ASCII character set, and more recently, the 8-bit Extended ASCII character set. *(To learn more about the ASCII character set and the Extended ASCII character set, visit the following URL [2] .)*

However, Java does not use the ASCII character set. Java characters are expressed as 16-bit Unicode characters. *(To learn more about Unicode characters, visit the following URL [3] .)*

6.7 Answer 4 .

False

Back to Question 4 (p. 2)

6.7.1 Explanation 4

All of the integral types other than **char** are signed. However, although the **char** type is integral, it is unsigned. Its range is from 0 to 65,535.

6.8 Answer 3 .

None of the lines are in error *(unless I made a typing error)* .

Back to Question 3 (p. 2)

6.8.1 Explanation 3

Each of the four primitive types identified in this question are signed integers. Signed integers in Java are represented in *two's complement binary notation.*

In summary, this means that a variable containing **n** bits has the numeric range shown in the following note box *(where 2eX means 2 raised to the X power):*

NOTE:

```
    Minimum = -2e(n-1)
Maximum = (2e(n-1))-1
```

Consider the simple example in the following note box. Assume that **n** is equal to 3. Then:

NOTE:

```
Minimum = -2e(3-1) = -4
Maximum = (2e(3-1))-1 = 4-1 = 3
```

A three-bit variable can represent eight different combinations of bits. In two's complement notation, it can represent the range of integer values from -4 to +3. Listing 2 (p. 7) shows how the bits would be organized to represent those values in two's complement binary notation.

Listing 2: Listing for Answer 3.

[2] http://www.telacommunications.com/nutshell/ascii.htm
[3] http://www.unicode.org/

```
000,  0
001,  1
010,  2
011,  3
100, -4
101, -3
110, -2
111, -1
```

6.9 Answer 2 .

true and **false** are the possible states of a **boolean** variable.

Back to Question 2 (p. 2)

6.9.1 Explanation 2

A **boolean** variable can have only two states, **true** and **false** .

The **boolean** type is often used in conditional expressions to make a choice between two possibilities. For example, the construct shown in Listing 3 (p. 8) would not work properly if the **boolean** were allowed to have the third value *uncertain* . In that case, you might find yourself stranded on the roadside with an empty gas tank.

Listing 3: Listing for Answer 2.

```
if(gasInTank < 1)
  buyGas(); //need to buy gas
else
  keepDriving();//don't need to buy gas
```

6.10 Answer 1 .

Line 2, **char** is in error. This line should show that the size of a **char** type is 16 bits.

Back to Question 1 (p. 1)

6.10.1 Explanation 1

Whereas many earlier programming languages, such as C and C++, have used 8-bit characters, the **char** or character type in Java is based on the 16-bit Unicode character set.

You may wonder why this is the case. One of the primary reasons is the need to internationalize computer technology. An 8-bit character set can represent only 255 different characters. During that part of history when most computer technology was based on the U.S./English alphabet, this was sufficient. However, computer technology is no longer so confined, and there is a need to be able to represent more than 255 characters.

The 16-bit Unicode set makes it possible to represent more than 65,000 different characters. This should go a long way towards representing the alphabets of all countries in the international computer technology arena.

You may also wonder about the correctness of line 1 which indicates that the **boolean** type is 1 bit in size. I personally don't know exactly how a **boolean** variable is represented in memory, but I would have guessed that it is stored in an 8-bit byte. I also don't know how to write any code that will expose the manner in which a **boolean** variable is stored in memory.

The size of one bit for a **boolean** is based on information from the book entitled *The Complete Java 2 Certification Study Guide* by Roberts, Heller, and Ernest. A **boolean** certainly could be stored in a single bit since only two different values must be represented. This assumes, of course, that the processor has the ability to address memory down to the single-bit level.

In the final analysis, it probably doesn't matter. Unlike C and C++, you cannot do arithmetic with a **boolean** , and I can't think of any other operation that you can perform on a **boolean** where the manner in which it is stored in memory makes any difference *(except possibly for speed)* .

-end-