COMP2026 Problem Solving Using Object Oriented Programming

**Part C Discovery Exercises**

**Task 1: Built-in Type**

Execute the given **BuiltinType.java** program and fill in the following table.

|  |  |  |
| --- | --- | --- |
| Built-in Type | Size (in bits) | Range |
| byte | 8 bits | From -128 to 127 |
| short | 16 bits | From -32768 to 32767 |
| int | 32 bits | From -2147483648 to 2147483648 |
| long | 64 bits | From -9223372036854775808 to 9223372036854775807 |
| float | 32 bits | N/A |
| double | 64 bits | N/A |

**Task 2: Escape Sequences**

A character preceded by a backslash (\) is an ***escape sequence*** and has special meaning to the compiler. The following table shows the Java escape sequences:

|  |  |
| --- | --- |
| Escape Sequence | Description |
| \t | Insert a tab in the text at this point. |
| \b | Insert a backspace in the text at this point. |
| \n | Insert a newline in the text at this point. |
| \r | Insert a carriage return in the text at this point. |
| \f | Insert a formfeed in the text at this point. |
| \' | Insert a single quote character in the text at this point. |
| \" | Insert a double quote character in the text at this point. |
| \\ | Insert a backslash character in the text at this point. |

When an escape sequence is encountered in a print statement, the compiler interprets it accordingly. For example, the statement

System.out.println("Letter \'A\'");

prints

Letter 'A'

Write the print statement(s) to print the pattern on the right.

|  |
| --- |
| System.*out*.println(" \\\\\\\\"); System.*out*.println("+\"\"\"\"\"+"); System.*out*.println("| - \* |"); System.*out*.println("| v |"); System.*out*.println(" \\ /"); System.*out*.println(" ---"); |

**Task 3: Prefix and Postfix operators**

|  |
| --- |
| ...  int i = 3;  i++; // i = 4  System.out.println(i); // prints 4  ++i; // i = 5  System.out.println(i); // prints 5  System.out.println(++i); // i = 6 and then prints 6  System.out.println(i++); // prints 6 and then i = 7  System.out.println(i); // prints 7  ... |

When used in an assignment or print context, the prefix operator **++i** first increments **i** and then returns the value of **i**, whereas the postfix operator **i++** returns the value of **i** and then increments **i**.

Similarly, the prefix operator **--i** first decrements **i** and then returns the value of **i**, whereas the postfix operator **i--** returns the value of **i** and then decrements **i**.

|  |
| --- |
| ...  int i = 3;  i--; // i = 2  System.out.println(i); // prints 2  --i; // i = 1  System.out.println(i); // prints 1  System.out.println(--i); // i = 0 and then prints 0  System.out.println(i--); // prints 0 and then i = -1  System.out.println(i); // prints -1  ... |

After the following code executes, what are the values of ***x*** and ***y***?

|  |
| --- |
| x = 6;  y = ++x; |

***x*** = 7 . ***y*** = 7 .

After the following code executes, what are the values of ***x*** and ***y***?

|  |
| --- |
| x = 6;  y = x++; |

***x*** = 6 . ***y*** = 7 .

Fill in the blanks **with prefix or postfix operator** to produce the output specified.

|  |
| --- |
| ...  int i = 5;  System.out.println(i++); // prints 5  **System.out.println(--i); // prints 5**  **System.out.println(++i); // prints 6**  **System.out.println(i++); // prints 6**  **System.out.println(++i); // prints 8**  **System.out.println(i--); // prints 8**  **System.out.println(--i); // prints 6**  **System.out.println(i++); // prints 6**  System.out.println(i); // prints 7  ... |

**Task 4: Formatted output**

In addition to the **print()** and **println()** methods, the **printf()** method is also frequently used for displaying text that needs to be formatted. Formatting here refers to how data is to be displayed. For example, the number of decimal digits to be displayed, the alignment of the String, etc.

The **printf()** method requires a format String and a set of other arguments whose number depends on the format String. The format String contains specifier(s) that specifies the type of data that is going to be displayed and the formatting details. For example,

|  |
| --- |
| ...  double num1 = 10.0/3;  double num2 = 10.0/6;  System.out.println("The numbers are " + num1 + " and " + num2);  System.out.printf("The numbers are %.2f and %.2f", num1, num2);  ... |

prints



Format specifiers include flags, width, precision, and conversion characters in the following sequence:

**%[flags][width][.precision] conversion-character**

*Note: square brackets denote optional parameters*

**Flags**:

|  |  |
| --- | --- |
| Flag | Description |
| – | Left-justify (default is to right-justify) |
| + | Output a plus (+) or minus (-) sign for a numeral value |
| , | Include locale-specific grouping characters |
| 0 | Forces numerical values to be zero-padded (default is blank padding) |
|  | Space will display a minus sign if the number is negative or a space if it is positive |

**Width** specifies the field width for outputting the argument and represents the ***minimum*** number of characters to be written to the output. Include space for expected commas and a decimal point in the determination of the width for numerical values.

**Precision** specifies the number of digits of precision when outputting floating-point values or the length of a substring to extract from a string. Numbers are rounded to the specified precision.

**Common Conversion-characters**:

|  |  |
| --- | --- |
| Conversion-character | Description |
| c | Display a character |
| d | Display an integer number (base 10) [byte, short, int, long] |
| f | Display a floating-point number [float, double] |
| s | Display a string of characters |

Examples:

|  |  |
| --- | --- |
| Specifier | Result |
| %8d | Integer, right-aligned, 8-space-wide field |
| %–6d | Integer, left-aligned, 6-space-wide field |
| %.2f | Floating-point number, rounded to nearest hundredth |
| %16.3f | Floating-point number, rounded to nearest thousandth, 16-space-wide field |
| %–9s | String, left-aligned, 9-space-wide field |

Complete the following **printf()** method to print the specified output.

|  |
| --- |
| ...  int n = 12345;  double pi = Math.PI;  String str = "Hello";  System.out.printf("[%d]\n", n);  System.*out*.printf("[%9d]\n", n); System.*out*.printf("[%-9d]\n", n); System.*out*.printf("[%09d]\n", n); System.*out*.printf("[%,9d]\n", n); System.*out*.printf("[%+,9d]\n", n);  System.out.printf("[%f]\n", pi);  System.*out*.printf("[%.3f]\n", pi); System.*out*.printf("[%8.3f]\n", pi); System.*out*.printf("[%+8.3f]\n", pi);  System.out.printf("[%s]\n", str);  System.*out*.printf("[%10s]\n", str); System.*out*.printf("[%-10s]\n", str);  ... |

prints

|  |
| --- |
| [12345]  [ 12345]  [12345 ]  [000012345]  [ 12,345]  [ +12,345]  [3.141593]  [3.142]  [ 3.142]  [ +3.142]  [Hello]  [ Hello]  [Hello ] |

**Task 5: ASCII Code**

Computer store data in binary format. Every piece of information, including characters, numbers, and even program instructions, is stored as a sequence of 0’s and 1’s. For example, a lowercase ‘a’ is represented by 1100001 (97 in decimal) and a ‘b’ is encoded as 1100010 (98 in decimal). This encoding is used to identify a character’s ASCII code (American Standard Code for Information Interchange). Every character that appears on your keyboard has its own 7-bit ASCII sequence or code.

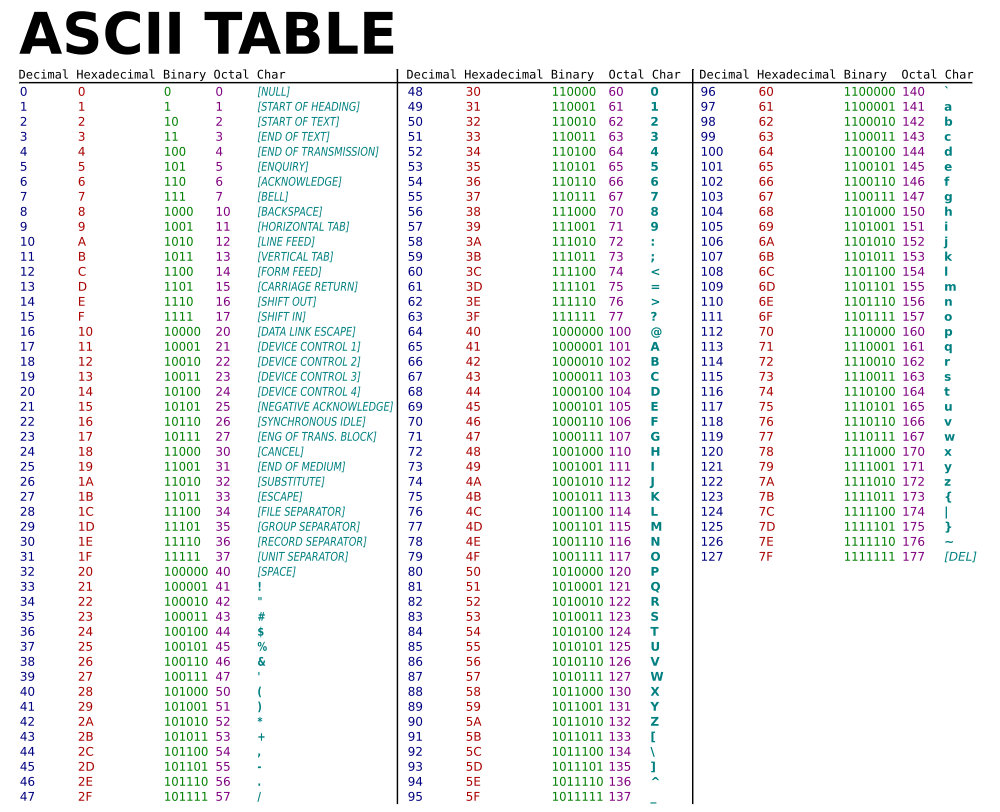


Image source: <https://qph.fs.quoracdn.net/main-qimg-40eb504e356af0832441ef95091eba8e>

The following code fragment can print the decimal ASCII value of a given character, e.g. ‘a’.

|  |
| --- |
| char c = 'a';  System.out.println((int)c); |

The following code fragment can print the character representation of a given decimal ASCII value.

|  |
| --- |
| int asciiCode = 97;  System.out.println((char)asciiCode); |

Using the ASCII Table above, answer the following questions.

Fill in the following table.

|  |  |  |  |
| --- | --- | --- | --- |
| Letter | Lowercase ASCII Value | Uppercase ASCII Value | Lowercase ASCII Value – Uppercase ASCII Value |
| A | 97 | 65 | 97-65=32 |
| B | 98 | 66 | 98-66=32 |
| C | 99 | 67 | 99-67=32 |
| D | 100 | 68 | 100-68=32 |

What is the different between the decimal ASCII values of lowercase letter and uppercase letter?

Ans: The **difference** between the **lower** **case** and the **upper** **case** **letter** is always 32.

What is the range of the decimal ASCII values of lowercase letter a to z?

Ans: 97 to 122 .

What is the range of the decimal ASCII values of uppercase letter A to Z?

Ans: 65 to 90.

What is the range of the decimal ASCII values of digit 0 to 9?

Ans: 48 to 57.

**References**

1. Bravaco, R., & Simonson, C. (2009). *Java programming: From the ground up*. Dubuque, IA: McGraw-Hill.
2. Dean, J., & Dean, R. (2008). *Introduction to programming with Java: A problem solving approach*. Boston: McGraw-Hill.
3. Farrell, J. (2012). *Java programming. Boston, MA: Course Technology Cengage Learning*
4. Forouzan, B. A., & Gilberg, R. F. (2007). *Computer science: A structured programming approach using C (3rd ed.)*. Boston, MA: Thomson Course Technology.
5. Gaddis, T. (2016). *Starting out with Java (6th ed.)*. Pearson.
6. Liang, Y. D. (2013). *Introduction to Java programming: Comprehensive version*. (8th ed.). Pearson.
7. Schildt, H. ( 2006). *Java a beginner's guide*. New York: McGraw Hill.
8. Schildt, H., & Skrien, D. J. (2013). Java programming: A comprehensive introduction. New York: McGraw-Hill.
9. Wu, C. T. (2010). *An introduction to object-oriented programming with Java*. Boston: McGraw Hill Higher Education
10. Xavier, C. (2011). *Java programming: A practical approach*. New Delhi: Tata McGraw Hill.
11. yet another insignificant Programming Notes. (n.d.). Retrieved from https://www3.ntu.edu.sg/home/ehchua/programming
12. Zakhour, S., Kannan, S., & Gallardo, R. (2013). *The Java tutorial: A short course on the basics (5th ed.)*.