

Homework 3

Q1) (B)

Data types supported by switch statements include the following:

- int and Integer
- byte and Byte
- short and Short
- char and Character
- String
- enum values

Q2) (D)

We can change the order of operation explicitly by wrapping parentheses around the sections we want evaluated first. Unless overridden with parentheses, Java operators follow order of operation.

Q3) (C)

Determining equality in Java can be a nontrivial endeavor as there's a semantic difference between "two objects are the same" and "two objects are equivalent." For object comparison, the equality operator is applied to the references to the objects, not the objects they point to. Two references are equal if and only if they point to the same object, or both point to null. equals() standard method checks the values inside the String rather than the String object itself. If a class doesn't have an equals method, Java determines whether the references point to the same object—which is exactly what == does.

Q4) (D)

Code does not compile because two **else** have been used in the code. The **if-then-else** statement provides a secondary path of execution when an "if" clause evaluates to false. **else if** statement is used to specify a new condition if the first condition is false.

Q5) (C)

There is no requirement that the case or default statements be in a particular order. **default** statement is also optional in switch statements and it can be used without any **case** statements although it will always execute. The **default** section handles all values that are not explicitly handled by one of the case sections, that is to say it does not take a value like case statements.

Q13) (B)

The values in each case statement must be **compile-time constant** values of the **same data type as the switch value**. This means we can only use **literals**, enum constants, or **final** constant variables of the same data type. **break** statements terminate the switch statement and return flow control to the enclosing statement. Having the break statement leaved out, flow **will continue** to the next proceeding case or default block automatically.

Q14) (B)

Given table describes the **&&** (and) logical operator. **AND** is only true if both operands are true.

Q18) (B)

A switch statement has a target variable that is not evaluated until runtime. Prior to Java 5.0, this variable could only be int values or those values that could **be promoted to int**, specifically byte, short, char, or int. When enum was added in Java 5.0, support was added to switch statements to support enum values. In Java 7, switch statements were further updated to allow matching on String values. Finally, the switch statement also supports any of the primitive numeric wrapper classes, such as Byte, Short, Character, or Integer.

Q24) (B)

Break statements terminate the switch statement and return flow control to the enclosing statement. If we leave out break statement, flow will continue to the next proceeding case or default block automatically.

Q38) (D)

The values in each case statement must be compile-time constant values of the same data type as the switch value. This means we can use only literals, enum constants, or final constant variables of the same data type. Final constant means that the variable must be marked with the final modifier and initialized with a literal value in the same expression in which it is declared.

Q39) (C)

For comparing a value if it is “**greater or equal to**” another value **>=** operator used and for comparing if a value is “**strictly less than**” another value **<** operator is used. These relational operators are applied to numeric primitive data types only. If the two numeric operands are not of the same data type, the smaller one is promoted to the bigger type.

Q40) (B)

The code compiled without any error and does not raise an exception at runtime. Operator precedence is overridden when evaluating the **turtle** variable by parentheses. **turtle** variable is assigned 30 as a result of the ternary operation.

hare variable's assignment is based upon whether **turtle** variable is smaller than **5** or not. As the result of the ternary operation **hare** variable is assigned **25** and since **turtle** is bigger than **hare** variable last ternary operation in `print` statement returns "Turtle wins!".

Q41) (A)

getResult() method simply check if the given parameter is bigger than 5 with. If it is returns **1** and if it is not returns **0**. Since in the main method **getResult()** method is invoked with actual parameters all smaller or equal to **5** they all evaluate to **0** and hence the sum is **0**.

Q42) (A)

Code compiles without any issue. The key point here is that the assignment operator is used here instead of comparison, right hand side is assigned to left hand side and newly assigned value is returned, so they are not compared. That is to say **roller** variable is the enactor and decides the if statement and because it is **true**, if statement evaluates to **true**, and '**up**' is printed.

Q43) (A)

The **&&** (and) operator is true if either of the operand are true, in all other cases it evaluates to false. Logical complement operator (**!**) flips a boolean value.

Q44) (A)

For the sake readability parentheses would be a good idea but it is not mandatory in ternary operations. **movieRating** can't be 3 because **character <= 4** evaluates to false, so the right side is evaluated and returned in which there is another embedded ternary operation.

Q45) (D)

Switch statements can have any number of case statements and since the **default** statement handles all values that are not explicitly handled by one of the case sections, switch statement can have at most one **default** statement.

Q46) (D)

If the array's size is zero code can throw `indexOutOfBoundsException` exception at runtime, if the first element is null, left side of the ternary operator evaluates to false and "stay inside" is printed. Lastly if the first element in array is "sunny" all three boolean expressions evaluates to true and hence ternary operator returns "Go Outside" and it is printed.

Q47) (D)

The logical complement operator, `!`, flips the value of a boolean expression. For example, if the value is true, it will be converted to false, and vice versa. The code does not compile because logical complement operator is undefined for the type `int`. If we omit logical complement operator the expression on the right hand side of the assignment, after having considered the precedence rules, evaluates to 34.

Q48) (C)

The logical **OR** operator is only evaluated as true when one of its operands evaluates true. If either or both expressions evaluate to true, then the result is true.

Q49) (A)

Decrement operator has precedence over subtraction. Increment operator has precedence over division and modulus operators. Only true order of precedence is presented at option A. Addition operator has a lower precedence over division and division and multiplication are on the same level of precedence.

Q50) (C)

The code does not compile due to p1, because of the type mismatch error. Ternary operator return and integer and we are trying to assign it to an String reference type.