Emre KARAGÖZ

ASSIGNMENT 3
JAVA OPERATOS AND DECISION CONSTRUCTS
(HW 3)

Q1-) Option B

Data types supported by switch statements include the following:

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

"`boolean`" and "`long`", floating-point types like "`float`" and "`double`" and their associated wrapper classes, are not supported by switch statements. **Option B** is the answer.

Q2-) Option A

```
meal>6 ? ++tip : --tip
```

`meal` is 5 and it is less than 6. Because of this, "`--tip`" is evaluated. When it declared, "`int tip`" was 2. "`--`" operator is used as a prefix. The value of , "`int tip`" is decreased by 1 and, it changed from 2 to 1. **Option A** is the answer.

Q3-) Option C

"`String john`" and "`String jon`" are different objects. "==" operator compares memory addresses that these objects point out and, returns false. `equals()` method tests the values they refer and returns true because, these values are equals each other. The output of this application is "`false true`", **Option C** is the answer.

Q4-) Option D

The last "`else`" statement (`else System.out.print("Plan C")`); is not connected an if statement. Because of this, this code does not compile. **Option D** is the answer. If we edit the code and connect the second "if" statement and the last "else" statement each other due to "`int plan`" equals to "2" the output of this code becomes "Plan B".

Resource for evaluation order: https://docs.oracle.com/javase/specs/jls/se7/html/jls-15.html

Q5-) Option C

A "`switch`" statement does not have to include a "`default`" statement. It is not mandatory; it is optional, and it can be in any order within the case statements. So**, Option A** and **Option B** are not true. A "`default`" statement can be used even if there is no "`switch`" statement. So, **Option D** is false too. **Option C** is the answer because, unlike a "`case`" statement, the "`default`" statement does not take a value.

Q6-) Option B

"5" always equals "5", therefore the first branch of the ternary operator is evaluated and "3" is assigned to "thatNumber". After this assignment, the "`++`" operator increases the value of "`thatNumber`" by "1" and, makes it "4". The "`If`" statement is skipped because "4<4" is false. After the execution of this code, the value of "`thatNumber`" is 4 so, **Option B** is the answer.

Q7-) Option B

The "`break`" statement exits a "`switch`" statement, skipping all remaining "`case`" or "`default`" branches. **Option B** is the answer.

Q8-) Option C

In a ternary expression only one expression at the right of the conditional operator is evaluated and parentheses are not required. So, **Option A** and **Option B** are incorrect**. Option D** is incorrect too because, ternary expressions do not support "`int`" expressions for the left-most operand. Only "`boolean`" expressions are supported. A ternary expression is a condensed form of an if-then-else statement and we can utilize it instead of an if-then-else statement. **Option C** is the answer.

Q9-) Option C

The code does not compile because, "`candidateA`" and "`candidateB`" are Integers and "`&&`" operation cannot be applied on Integers. It can only be applied on numbers. So, **Option C** is the answer. Even if we edit the code and fix this problem, this app throws a null pointer exception error at runtime because, "`candidateA`" is pointed to null.

Q10-) Option A

The "`if-then`" statement is skipped because, "`pterodactyl`" equals to "6", "(6%3)" equals to 0, and "0" is less than "1". "triceratops" is initialized as 3, "--" operator was used as a postfix here. The value of , "`triceratops`" is decreased by 1 and, it changed from "3" to "2". "2" is the output of this, **Option A** is the answer.

Q11-) Option D

An "if-then" statement does not require to have an "else" statement. Else statements are optional, not mandatory. So, **Option A** is incorrect. If the "Boolean" test is false, target clause is skipped, not evaluated. Namely, **Option B** is also incorrect. "if-then" statements are not required to cast an object, this makes **Option C** incorrect. Answer is **Option D** because, an "if-then" statement can execute a single statement or a code block.

Q12-) Option D

Option D is the answer, none of the other options are the output of this code. "int flair" equals to 15 and, "(flair >= 15 && flair < 37)" is always true. This if statement is evaluated and, prints "Not enough". "(flair==37)" is always false, the second "if" statement is skipped, "else" block is evaluated and prints "Not enough". The final output is "Not enoughToo many".

Q13-) Option B

The values in each "case" statement must be compile-time constant values of the same data type as the switch value. This means you can use only literals, enum constants, or final constant variables of the same data type. So, **Option A**, **Option C**, and **Option C** are true. **Option B** is not true and it is the answer because, a "case" statement does not have to be terminated by a break statement. It can be, but it is not required.

Q14-) Option D

	x = true	**x = false**
y = true	**true**	**false**
y = false	**false**	**false**

"&&" operator corresponds this relationship so, **Option D** is the answer. If the table were as follows, the answer would be "||" operator.

	x = true	**x = false**
y = true	**true**	**true**
y = false	**true**	**false**

Q15-) Option C

The code does not compile because the test part of if statement is an "int". It must be a "boolean". **Option C** is the answer.

Q16-) Option B

The **pre-increment [++v]** operator increases the value of a variable by 1 and returns the new value, while the **post-decrement [v--]** operator decreases the value of a variable by "1" and returns the original value. So, Option B is the answer. "prefix" and "postfix" operators both change the value of the variable. prefix operators return the new value yet postfix operators return the original value.

Q17-) Option B

This code prints winner that equals to "lion+2*(tiger + lion)". If we change the variables with their values in this equation, we get "(3+2*(2+3))" and the result of this is "13". **Option B** is the answer.

Q18-) Option B

Any value that can be implicitly promoted to int will work for the case statement with an int input. "`int`" and "`short`" can be used here. Yet, as explained in Q1 "`long`", and its associated wrapper class are not supported by switch statements. **Option B** is the answer.

Q19-) Option D

In this code snippet, there are two ternary operations. We can write these ternary operations by using parentheses as "`String dinner = time > 10 ? (day ? "Takeout" : "Salad") : "Leftovers";`" but parentheses are not required. Namely, **Option C** is incorrect. This code does not compile so, there is no output. **Option A** and **Option B** are also incorrect. "`day`" variable is the reason why this code is not compiled. Because it is an "`int`" but it must be "`boolean`". **Option D** is the answer.

Q20-) Option C

This code is not compiled because a parenthesis is missing. To fix this we can edit the code such as;

```
int leaders = 10 * (2 + (1 + 2 / 5));
```

After this editing, we can get "`Too many`" as the output.

Q21-) Option B

In Java, '+' operator is used for both;

- Arithmetic Add
- String concatenation

Precedence of evaluation for "+" is from left to right and this is how it works:

- `System.out.println("3" + 3 + 3);`

When you try this, since the first parameter is a String, everything rest will be a String concatenation.

- `System.out.println(3 + "3" + 3);`

Same goes for this one, because you cannot add first int 3 to String 3 and then last 3 will be concatenated to second 3.

- `System.out.println(3 + 3 + "3");`

First left to right expression is evaluated (giving 3+3 = 6) and then string 3 is appended to result of that evaluation giving 63 as output.

```
(5 + 6 + "7" + 8 + 9)
```

In this equation, first left to right expression is evaluated (giving 5+6 = 11), and everything rest will be a String concatenation. So, the output is "11798", and **Option B** is the answer.

** https://stackoverflow.com/questions/38352779/java-operator-between-arithmetic-add-string-concatenation

Q22-) Option B

The **"-" (subtraction)** operator is used to find the difference between two numbers, while the **"%" (modulus)** operator is used to find the remainder when one number is divided by another. **Option B** is the answer.

Q23-) Option B

"int dog = 11" and "int cat = 3" and "3" does not divide 11 evenly. Because of this, int partA is rounded down to 3. "dog % cat" is 2 and when we changed variables with their values we get (2+3*3) which equals to 11. **Option B** is the answer.

Q24-) Option B

There is no break statement in this code so, after a matching case statement the other remaining case statements will be executed. First "`eaten`" which is 0 is increased by 1, after it increased by 2 and will be 3. Finally, it decreased by 1 and will be 2. 2 is the output **Option B** is the answer.

Q25-) Option C

This code does not compile because, "10" is not compatible with a String type of object. Ternary operations are required compatible data types. **Option C** is the answer.

Q26-) - (I think there are typos in this question)

Q27-) Option B

This code compiles and due to "`myTestVariable.equals(null)`" is always false, **Option B** is the answer. If `myTestVariable` was null, Option D would be also true because, the statement still compiles but it will produce a null pointer exception at runtime.

Q28-) Option D

The code does not compile due to the "`else if`" statement. "`&&`" operator is used to compare "`boolean`" types. "`int street`" is not comparable with a "`boolean`", and this prevents this code from compiling.

Q29-) Option B

Option A is incorrect because, the conjunctive operator "&" always evaluates both sides of the expression. Yet, the conditional conjunctive operator "&&" only evaluates the right side of the expression if it is enough to know the result. It is a short circuit operator. So, this operators are not interchangeable and **Option B** is the answer. They may produce different results in runtime. If both operands are true, this operators produce true as a result. This makes **Option C** incorrect.

Q30-) Option C

"`w`" is true, so the right side of the ternary operator is evaluated and the value of "`y`" (5) is assigned to "`x`". After that, post increment operator increased the value of "`y`" by 1 and makes it "6". "`!z`" is always true. Therefore, the value of "`w`" is not change and remains true. "`(x+y)`" is "(5+6)", and this equals to "11". Since "`w`" is still true, "`(w ? 5 : 10)`" returns "5". The output is "11 5", and **Option C** is the answer.

Q32-) Option B

The table below demonstrates operator precedence in Java.

Precedence	Operator	Type	Associativity
15	() [] .	Parentheses Array subscript Member selection	Left to Right
14	++ --	Unary post-increment Unary post-decrement	Right to left
13	++ -- + - ! ~ (*type*)	Unary pre-increment Unary pre-decrement Unary plus Unary minus Unary logical negation Unary bitwise complement Unary type cast	Right to left
12	* / %	Multiplication Division Modulus	Left to right
11	+ -	Addition Subtraction	Left to right
10	<< >> >>>	Bitwise left shift Bitwise right shift with sign extension Bitwise right shift with zero extension	Left to right
9	< <= > >= instanceof	Relational less than Relational less than or equal Relational greater than Relational greater than or equal Type comparison (objects only)	Left to right
8	== !=	Relational is equal to Relational is not equal to	Left to right
7	&	Bitwise AND	Left to right
6	^	Bitwise exclusive OR	Left to right
5	|	Bitwise inclusive OR	Left to right
4	&&	Logical AND	Left to right
3	||	Logical OR	Left to right
2	? :	Ternary conditional	Right to left
1	= += -= *= /= %=	Assignment Addition assignment Subtraction assignment Multiplication assignment Division assignment Modulus assignment	Right to left

** http://www.cs.bilkent.edu.tr/~guvenir/courses/CS101/op_precedence.html

According to this, `"12+6*3%(1+1)"` equals to `"12+6*3%2"`, `"12+18%2"`, `"12+0"`, and `"12"`, respectively. **Option B** is the answer.

Q31-) Option A

"(bob==notBob)" this statement compares values of "String bob" and "String notBob". Their values are the same ("bob") so, this statement returns "true".

"(bob.equals(notBob))" this statement is always "true" because, "String bob" and "String notBob" are the same object.

Namely the final output is "true true", and **Option A** is the answer.

Q33-) Option D

	p = true	**p = false**
q = true	**false**	**true**
q = false	**true**	**false**

This question is about the "exclusive or (^)" operator, and the table above is the truth table of XOR. Missing values are **true** and **false.** **Option D** is the answer.

Q34-) Option C

If data is an empty array, the "if" statement is skipped and nothing is printed. If data[0].equals("sound") or data[0].equals("logic"), if statement is run because in both cases the "data.length" is 1, it is less than 2, and equals to 1. In this cases, output will be "sound" or "logic". So, **Option A, B, and D** are possible outputs. The answer is **Option C.**

Q35-) Option C

As demonstrated in the table in Q32 operators "+" (level 11) , "/" (level 12), "*" (level 12), "%" (level 12), and "++" (level 13 or level 14), are listed in the same or increasing level of operator precedence. **Option C** is the answer.

Q36-) Option D

As we can see from the truth table in Q33, being "true" of one of the operands is not enough for the result to be "true". If both operands of XOR are "true", the result is "false". Therefore, **Option A** and **Option C** is incorrect. **Option B** is also incorrect because, there is no operator such as "^^" in Java. The logical "XOR (^)" can only be applied on Boolean values. **Option D** is the answer.

Q37-) – Option D

This diagram shows us "`x||y`" relationship. z is unused in the diagram so, it is not required in the expression. **Option D** is the answer.

****https://www.allaboutcircuits.com/textbook/digital/chpt-8/boolean-relationships-on-venn-diagrams/**

Q38-) Option D

As explained in Q13, the values in each "`case`" statement must be compile-time constant values of the same data type as the switch value. This means you can use only literals, enum constants, or final constant variables of the same data type. In the code in this question, "final" is missing. Therefore, the code does not compile regardless of variable type. **Option D** is the answer.

Q39-) Option C

<	strictly less than
<=	less than or equal to
>	strictly greater than
>=	greater than or equal to

This question asks about greater than or equal to (>=) and strictly less than (<) operators. **Option C** is the answer.

Q40-) Option B

The code compiles and prints **"Turtle wins!"** as the output. **Option B** is the answer. Respect to operator precedence table from Q32, "`10*(2 +(3+2)/5)`" equals to "`10*(2 +(5)/5`", "`10*(2+1)`", "`10*3`", and "`30`", respectively. "30" is assigned to "int turtle". Since "30" is greater than 5, "25" is assigned to "int hare". Finally, the ternary operator checks if the value of "int turtle" is less than the value of "int hare" and returns **"Turtle wins!"**.

Q41-) Option A

Values of 5, 1, 0 and 2 assigned to the "`threshold`" variable are always less than or equal to 5. So, getResult() method returns 0 in each case. Therefore, this code compiles and prints 0 as the output. **Option A** is the answer.

Q42-) Option A

The value of "`roller`" assigned to "spinner" on this line: "`if(spinner = roller)`". Due to "`roller`" is true, "`runTest()`" method returns "`up`" and it is the output. **Option A** is the answer.

Q43-) Option D

The **"||"** operator is true if either of the operands are true, while the **"!"** operator flips a boolean value. . **Option D** is the answer.

Q44-) Option A

In a ternary expression, parentheses useful but they are not required. There are 2 ternary operations in this code snippet. Due to "`character (5)`" is not less than "4", `(story > 1 ? 2 : 1)` is executed. "`story`" is "3" and it is greater than "1". This ternary operation returns "2.0" and "2.0" is assigned to "`double movieRating`". **Option A** is the answer.

Q45-) Option B

A "switch" statement can have **any number of** "`case`" statements or cannot have any but can have **at most one** "`default`" statement. **Option B** is the answer.

Q46-) Option A

The code compiles but throws "`java.lang.ArrayIndexOutOfBoundsException`" error at runtime. Because, "`&&`" operator cannot test "`null`" arrays. If we ignore this, "`Go Outside`" or "`Stay Inside`" can be the output of this code because ternary operation returns one of them.

Q47-) Option D
Operator '!' cannot be applied to 'int'. Therefore, the code does not compiles. **Option D** is the answer.

Q48-) Option C

	w = true	**w = false**
z = true	**true**	**true**
z = false	**true**	**false**

This question is about the "||" operator, and the table above is the truth table of "||". Missing values are **true** and **true.** **Option C** is the answer.

Q49-) Option A

As demonstrated in the table in Q32 operators "-" (level 11) , "+" (level 11), "/" (level 12), "*" (level 12), and "%" (level 12), are listed in the same or increasing level of operator precedence. **Option A** is the answer.

Q50-) Option C

The code does not compile because "game" is a "String" but the ternary operation in "`String play()`" method returns an "`int`" value but this int value cannot be assigned to "`game`". **Option C** is the answer.