Bünyamin AKTAŞ
01/05/2020

ASSIGNMENT – 3

Q1 **The answer is B.**

Switch statements are complex decision making structures in which a single constant value or a final constant variable is evaluated and when that value is matched with the value in a *case* statement, the code that belongs to that *case* statement is executed. An optional *default* statement can be placed in *switch* block. If none of the *case* statements are executed this *default* statement will be executed. If there is not a *default* statement then all *switch* block will be skipped.

At first, Java was supporting only *int* type values and values that can be converted to *int* type, such as *byte, short* and *char.* Later it started to support *enum* and *String* types, too. So, *String* in option **A,** *int* in option **C** and *char* in option **D** are supported in *switch* statements while ***double* type values are not supported (B).**

Q2 **The answer is A.**

Conditional operator or in another name ***ternary operator,*** *? :,* is a condensed form of an *if-then-else* statement.

booleanExpression ? expression1 : expression2

This operator has 3 operands which the first one is a boolean expression. If the boolean expression returns *true* then *expression1* is executed, if not *expression2* is executed. So for the statement,

(meal>6 ? ++tip : --tip), the compiler checks if *meal* is bigger than 6. *meal* is 5 so that it returns *false*. This means the second expression (third operand) will be executed, which is a pre-unary expression that decrements the variable by 1. **So the variable *tip* will be *2 – 1 = 1***. The answer is **A.**

Q3 **The answer is C.**

Equality operator, ==, check whether the literal values of primitive data types are equal. If reference types are used with the equality operator then it checks if they reference to the same object.

equals() method also checks for object equality for all Objects but if a class override this method it can act different way. *String* class has such a special *equals()* method which checks if the values in *String* objects are equal rather than checking if they reference to same object.

So here we have an object called *john* which has a literal value “john”. The other reference type *jon* reference to a totally different object because the code enforces it by declaring *new String()*. *john==jon* statement will check if these two variables references to the same object which

will return *false* because they reference to totally different objects. But *john.equals(jon)* will check if the values stored in the objects referenced by these two variables are equal which will return *true* because they do have same value "john". So our output will be ***false true,* which is option C**.

Q4 **The answer is D.**

If-then-else statements can have one *if* block, numerous *else-if* blocks and only one *else* block. The code will check the boolean expressions in *if* and *else if* blocks in sequence until it finds a block that evaluates to true. If it cannot find one, then it will execute the *else* block.

In this code snippet there is one *if* block and two *else* blocks that last one is outside of the block of *if-then-else* statement. **So it is an incorrect syntax so that the code will not compile. The answer is D.**

If *else { if … }* statement was *else if { … }* then the code would compile. In that scenario,

plan = plan++ + --plan, would be evaluated.

Post-unary operators, such as *plan++ or plan--*, increment or decrement values of variables by 1 but output the values before the decrement occurred. Pre-unary operators, such as *++plan or --plan*, increment or decrement the values of variables by 1 and output the new value. So here in first expression *plan++* would increment the value of variable *plan* by 1 but would output the value before incrementing. *plan* is initialized to 1, so its value is now 2 but **1** would be used. Now *--plan* would decrement the value of *plan* by 1 and would output the new value which is 2 -1 = **1**. The statement would be like *plan = 1 + 1*. The final value of *plan* would be 2 which would cause the boolean expression in *else-if* block return true and the code in the block would be executed and *Plan B* would be printed.

Q5 **The answer is C.**

As mentioned in **Q1** *switch* statements can have *case* and *default* statements but none are mandatory, which means there can be only *case* statements, or only one *default* statement, or nothing. *default* statement exists to be executed when none of the *case* statements match the value expressed. So by definition, there cannot be two *default* statements in one *switch* block. And again by definition, *default* statements don't take a value because then it would be same as a *case* statement. This makes **option C is true.**

Finally, *case* and *default* statements can appear in any order unless we want to execute more than one *case* statements which can be achieved by not using *break* statement.

Q6 **The answer is B.**

This code snippet has *ternary operator* which checks if literal *5* equals to or is bigger than literal *5*. For that this comparison returns *true* the first expression after the question mark *?* is executed which is 1+2. So the value of the variable *thatNumber* happens to be *3*. Then there is a *if-then* statement that checks if the value of *++thatNumber* is less than 4. *thatNumber'*s value was 3 and it is

incremented by pre-unary operator, ++, and happened to be 4 now. So the comparison will return false and the code that belongs to the *if-then* statement will be skipped. **At the end we have *thatNumber* that equals to 4.**

Q7 **The answer is B.**

A – There is no keyword called *exit* in Java.

B – Only *break* statement makes switch statement quit, skipping all remaining statements.

C – The keyword *goto* is not used in Java actually but still Java developers made it reserved to prevent it from being used. It has no special meaning in Java.

D – The keyword *continue* is allowed only in loops, not in *if* and *switch* statements. So it has no work in switch statements and even it had as it has in loops, *continue* statement does not exit statements instead, it turns them to the starting point.

Q8 **The answer is C.**

A – Because ternary expressions are a replacement for *if-then-else* statement there is no way of that both expressions evaluating at runtime. If one statement is evaluated then the other one will be skipped.

B – While parentheses are allowed and helpful in ternary operators for readability they are not required.

C – *If-then-else* statement is a structure that evaluates a boolean expression and executes some code according to the result of that boolean expression. *Ternary statement* also does the same thing. It evaluates the first operand that is a boolean expression and if it returns true, the second operand will be executed but if it returns false, the third operand will be executed. **So this option is true**.

D – The first operand appearing in ternary statement must be boolean expression. *int* or other type expressions are not allowed.

Q9 **The answer is C.**

Arithmetic variables are not allowed in logical comparisons in Java unlike other programming languages. This means the code will not compile due to the statement in line 4. If this line is fixed someway line 3 would not cause a *NullPointerException* error because *candidateA* is null-checked in that statement. If candidateA is null then the code does not execute *intValue()* method. **So the answer is C.**

Q10 **The answer is A.**

The modulus operator, %, divides the number before it by the number after it and then returns the remaining value. For example *11 % 5 = 1.*

In this code snippet there is an *if-then* statement which checks the remainder of *pterodactyl / 3.* Since *pterodactyl* equals to 6, the remainder will be 0. Then the the *if* statement, which has only one line, will be skipped. The next line after the *if-then* statement is *triceratops--*, which will decrement triceratops by 1 and make it 2. **So the output of the code will be 2**.

Q11 **The answers are D.**

A – *else* statements are optional not required so *if-then* statements may not have an *else* statement.

B – If the boolean test of an *if-then* statement returns *false* then the code clause will be skipped. It would be against to the nature of *if-then* statements if the code clause is still evaluated even when the boolean test returns *false*.

C – *If-then* statements can be used to check if an object is of a particular type but this is not the only use of *if-then* statements.

D – *If-then* statements executes the next code. If that code is in a block it executes all statements inside the block. If not, it executes the next single statement.

Q12 **The answer is D.**

In this code we have one *if-then* statement and one *if-then-else* statement, respectively. First statement checks if the variable *flair* is equal to or bigger than 15 and also less than 37. This boolean expression returns true and execute the code printing “Not enough”. Then the second statement checks if *flair* equals to 37 which is not so that it goes to the *else* statement and prints “Too many”. So the output should be **“Not enoughToo many”**. None of the options indicating some output are true.

Q13 **The answer is B.**

As mentioned in **Q1** *case* statements can take a constant value or final variables which make options **A** and **C** are correct.

case values must match the data type of *switch* variables or be able to be promoted to that type such as from *byte, short* and *int* to *long.* So the option **D** is also correct.

A *case* statement may not be desired to be terminated so that *case* statements may not be terminated by not using *break* statement which is allowed in Java. So the **option B is incorrect.**

Q14 **The answer is D.**

The table presents the **truth table** of AND operator. When AND operator is used, it returns true if and only if both boolean expressions are true.

There are two different but similar AND operators. One of them is **&** and the other one is **&&** which is also called *short-circuit operator.* So here the answer is **option D.**

The other options **A** and **B** are unary operators which decrement and increment variables by 1, respectively. Option **C** is *short-circuit logical operator*.

Q15 **The answer is C.**

Because numerals cannot be used as boolean values in Java the *if-then* statement does not compile. The answer is **C.**

By the way the value of *jumps* is 0 because *[v++]* is a post-increment operator which is used on the variable *hops* and increments the value of *hops* by 1 but outputs the previous value which is 0.

Q16 **The answer is B.**

The operator which increase the value of a variable by 1 and returns the new value is pre-increment [++v] and the operator which decrease the value of a variable by 1 but return the original value is post-decrement [v--] as it is explained in **Q4.** So the answer is **B.**

Q17 **The answer is B.**

short can be promoted to *int* and *int* can be promoted to *long*, which means that all operations stated in this code are valid. Firstly, the most-right operand which is between parentheses is executed which will return 5 that will be multiplied by 2 and then added to the variable *lion* that is 3. The result will be *3+2*(2+3) = 13.* **The answer is B.**

Q18 **The answer is B.**

The *case* statements can take a value only which matches with the data type of *switch* value or a value which can be promoted to the data type of *switch* value. If we assume that the variable of *switch*, dayOfWeek, is *int*, then *case* value, saturday can be of type *byte, short* or *int. long* type is wider then *int* therefore it cannot be promoted to *int.* **The answer is B.**

Q19 **The answer is D.**

The variable *day* which is of *int* type is tried to use as a boolean expression which is not allowed in Java unlike other programming languages. So this code will not compile which makes **option D is true.**

But if *day* will be replaced by a boolean expression then the code would compile. Because the first operand returns *true*, the second operand would be executed, which is a ternary operator actually. So this ternary operator would check its first operand for what it returns and if it is *true* then it would assign "Takeout" to the variable *dinner* but if it returns false then it would assign "Salad" to the variable *dinner.*

If parentheses were used for the inner ternary operator it would increase the readability of the code but they are not required.

Q20 **The answer is C.**

In assignment of the variable *leaders*, there is one missing parenthesis that will cause the code not compile. **So the answer is C.**

If we fix the problem by adding one parenthesis at the end of the assignment the code would compile. In that scenario, because *leaders* is of type *int,* 2/5 would return 0 and the result value of *leaders* would be 30. Naturally the variable *followers* would be 60. In the ternary operator which is put in *System.out.print()* method would add the two variables up and check if they are less than 10. Since they equals to 90 the result would be *false* which would cause the third operand, "Too many", be printed.

There is no 0 used in division operations so the code would not throw a *division by zero* error.

Q21 **The answer is B.**

Concatenation is subject to 3 rules. First one is that if two numeric values are concatenated then it means numeric addition. Second one is that if one of the operands is of type String then it means String concatenation. Third and last one is that the expression is evaluated from left to right.

In the expression *System.out.print(5 + 6 + "7" + 8 + 9)* the compiler will look at *5* and *6* at first. They are both numeric so the compiler will add them up to 11. Then the expression becomes *...11 + "7" + 8 + 9…* Here the compiler will look up the first two values which are numeric *11* and String *"7"*. So here the second rule comes in that because one of the operands is String the String concatenation will be done which will result in "117". After this since the first operand is now "117" of String type the next evaluation will also be String concatenation which results in "1178" and then going on, will result in "11789". **So the answer is B.**

Q22 **The answer is B.**

The difference between two numbers are found by subtracting one from another which can be performed by subtraction (-) operator in Java. The remainder of a division of two numbers are found by using modulus (%) operator between them. **So the answer is B**.

Operator (/) is used for division. It returns the result of division of two numbers, not the remainder or the difference. Operator (<) is a relational operator which checks if first numeric is less than the second numeric which will return a boolean value. Operator (||) is a logical operator which checks two boolean expressions and returns true if one of them is true.

Q23 **The answer is B.**

The variable *partA* must be a whole number, so that part after the decimal point of the result of division of *dog* by *cat* will be thrown away. So *partA* will be 3. The variable *partB* is initialized to the remainder of division of *dog* and *cat* variables, which is 2. The last variable *newDog* is ***2 + 3 * 3 = 11.*** **The answer is B.**

Q24 **The answer is B.**

Once a case statement matches all the remaining *case* statements and the *default* statement will be executed because there is no break statement which would cause the *switch* statement exit. So, the first *case* statement matches the value *30* and *eaten* is incremented by 1. Then the second statement will be executed which adds 2 to *eaten* that makes *eaten* 3. Finally *default* statement will be executed which decrements *eaten* by 1 that makes *eaten* 2. **So the output will be 2.**

Q25 **The answer is C.**

This code will compile because the first method which has a return type, String, contains a return statement which has a ternary operator which has a potential numeric return value. This is not allowed. **So the answer is C.** But if the third operand of the ternary operator was a String value such as “10” then it would compile and the output would be “train”.

Q26 **The answer is A.**

A – If two String objects evaluate to *true* using equality operator, ==, which checks whether two object variables reference to same object, then it must also return *true* when using *equals()* method.

B – If *not equal* operator, !=, returns *true*, then it means the reference names are referenced to two different objects which means they may not contain the same value as *equals()* method checks whether the values stored in String objects are the same so that they may return false using *equals()* method.

C – *equals()* method checks whether two String objects contain same value rather than object equality while equality operator, ==, checks whether they reference to one same object. Two

different String objects can contain same value which will cause *equals()* method return true but the equality operator, ==, return false.

D – As it is explained in option **C,** *equals()* method checks whether two objects contain same value. These two objects can be same single object or different two objects. If they reference to same object then *not equals* operator, !=, would return *false.*

Q27 **The answer is B.**

The method *equals()* which is overloaded in String class checks if two String objects contain exactly same characters. Since the variable *myTestVariable* is not *null*, *equals()* method can be called upon it, which means the program will not produce an exception. The method will try to compare the value stored in *myTestVariable* with the value stored in *null* which does not exist because *null* means no object. **Therefore this will return *false.***

Q28 **The answers is D.**

In *else-if* statement, the variable *streets* which is of *int* type is tried to be used as a boolean expression which is not allowed in Java. **This will cause that the code does not compile.**

But if the boolean expression was changed as *streets > 1000 && intersections > 1000* then it would compile and two 1s will printed.

Q29 **The answer is B.**

Java has two different type logical operators. One type is &, ^ and |. The other type is && and ||. Difference between these types can be explained so that while the first one evaluates two operands in any case, the second one may evaluate only one of the operands if it reaches a conclusion.

For example, if logical operator &&, which is also called *short-circuit operator,* is used it checks the first boolean expression. If it returns *true* then it will check whether the second boolean expression also returns *true.* If both operands return *true* then the operator returns *true.*

But if the first boolean expression returns *false*, then there is no necessary for && to check the second expression because even if it returns *true,* that doesn't means anything since the first expression returned *false* the *short-circuit logical operator, &&,* will return *false*. **So the operator will not evaluate the second operand in this case.**

In conclusion, we can say that these two operators are not interchangeable. As an example if we would like to perform a *null-check*, & will check whether the object is *null* and even if the object is *null* the operator will try to evaluate the second operand which can cause *NullPointerException.* Both operators evaluates to *true* if both operands are *true* which falsifies the **option C. Option D** is also incorrect because the operator && starts to evaluate from left operand, it cannot only evaluate the right operand.

Q30 **The answer is C.**

The code compiles because there is not syntax error or prohibited operation. The variable *x* will be 5 since *w* is true and therefore *y++* is evaluated which has a post-unary operator which increments the variable by 1 but outputs the original value. The variable *z* is *false* and the operator ! negates boolean values which will output *true* when it is used as *!z*. So *w = !z* does not change the original value of the variable *w.* So *(x+y)+" "+(w ? 5 : 10)* will firstly evaluates the first parentheses which is a numeric addition that evaluates to 11. Then, after a white space is added, we have a ternary operator which will evaluate the second operand since *w* is still *true*. **So the output will be *11 5.***

Q31 **The answer is A.**

A String variable, which contains "bob" value, is created with a reference name *bob*. Then a second String variable is created with reference name *notBob* and then it is pointed to the object of *bob.* So in result two different reference names started to point to same object. **Because these two reference names points to the same object, both the object equality operator, ==, and the *equals()* method will return *true.***

Q32 **The answer is B.**

If arithmetic operations contain a section with wrapped by parentheses then that section has a priority. This means at first, (1 + 1) will be evaluated which results in 2. After that, since Java gives priority to *multiplication/division/modulus* operators over *addition/subtraction* operators, *6 * 3 % 2* will be evaluated from left to right. It will become 18 % 2 which is 0. Then the statement becomes 12 + 0. So the value of that statement is **12.**

Q33 **The answer is D.**

The operator ^ is exclusive or which evaluates to true if the operands are different. When both the operands are *true* or *false,* the exclusive or evaluates to *false.* So since the first blank is represent *true^false,* it will evaluate to *true,* while the second blank will evaluate to *false* due to that it represents *false^false*. **The answer is D.**

Q34 **The answer is C.**

The parameter of *main()* method takes an argument which is a String array. The *if-then* statement controls this array if it contains one element. If it contains only one element and if that is *sound* or *logic* the next code will print it. But if it contains more than 1 elements or the element is not *sound* or *logic* it would print nothing. So the options **A, B** and **D** are possible results.

Q35 **The answer is C.**

Order of operators in increasing is like this: ...addition/subtraction operators, multiplication/division/modulus operators and unary operators.

A – This option begins with multiplication operator and continues with a unary operator. So far it is okay but then division operator comes and this is less precedence than unary operator which makes this option incorrect.

B – In this option, subtraction operator comes after modulus operator which has more precedence than subtraction.

C – This option contains division, multiplication and modulus operators respectively, which is in same level of precedence. So this option is correct.

D – This option also has subtraction operator after multiplication operator, similar to the option **B.**

Q36 **The answer does not exist**

As mentioned in **Q33,** exclusive operator ^ evaluates to true if both operands are not same. So one of the operands being *true* is not enough information for that the operator evaluates to *true,* which falsifies **option A.** when both operands are *true* or *false* this operator would evaluate to *false,* which makes **option C** wrong. There is no conditional operator denoted as ^^ because exclusive operator cannot reach a conclusion without evaluating both operands, which makes **option B** wrong.

^ operator can be applied to numerics as bit-wise *operator* which would make bit-wise comparisons between bits that compose those numbers. So **option D** is also wrong.

Q37 **The answer is C.**

When boolean relationships are shown in Venn diagrams OR logical operation is shown as sets are unified while AND is shown as the common part of sets.

In this diagram we see two sets, x and y, are unified while z is unused. So this diagram can represent **x || *y*. The answer is C.**

Q38 **The answer is D.**

For a variable to be able to be used in *case* statements, that variable must be *final* and initialized in declaration statement and also must be of type matching with the *switch* value or promotable to the data type of *switch* value as explained in **Q1**. The variable is initialized in the declaration statement but *long* cannot be promoted to type *int* and *double* cannot be passed into *switch*

statements. **Option C** is a correct data type but is missing *final* keyword which is also not present in the code. So **option C** is also wrong. **The answer is D.**

Q39 **The answer is C.**

Equal to or greater than operator is >=. *Less than* operator is <. The **option C** is the only one that contains both operators.

Q40 **The answer is B.**

The operator precedence goes as *parentheses, multiplication/division/modulus, addition/ subtraction.* So the variable *turtle* will be *10 * (2 + (3 + 2) / 5) → 10 * (2 + 5 / 5) → 10 * (2 + 1) → 10 * 3 → 30.* The variable *hare* will be initialized to the third operand of ternary operator, which is 25, because the first operand evaluates to *false*.

Finally the first operand of the ternary operator in *System.out.print()* method evaluates to *false* since *turtle* is not less than *hare*, **"Turtle wins!" will be printed.**

Q41 **The answer is A.**

The method *getResult()* returns the result of ternary operator which checks if the parameter is bigger than 5. If it is then it evaluates to 1, if not then it evaluates to 0.

For that none of the invokings in *System.out.print()* method take a value bigger than 5, all of them will return 0. Then the last value, 0, will be concatenated as a String with "". Finally String value "0" will be printed.

Q42 **The answer is A.**

The output of an assignment is equal to the assigned value. This means *spinner = roller* will return the value of *roller.* So, because *roller* is *true*, the *runTest()* method returns "up".

Q43 **The answer is D.**

There are two similar operators which evaluates to *true* if either of the operands are true; they are | and ||. So the **options A** and **B** are wrong.

The operator that flips a boolean value is the *logical complement operator (!).* The operator (-) indicates a number is negative when it is put before the number.

So the **option D** which has two valid operators, is correct.

Q44 **The answer is A.**

Since *characters* is not less than or equal to 4, the first operand of the ternary operator will evaluate to *false*, which will cause the third operand will be executed, which is also a ternary operator. This inner ternary operator will check whether *story* is bigger than 1 and for that *story* is 3, it will return *true* and execute the second operand that is 2.

Although the output the outer ternary operator is an integral value, 2, Java automatically promotes this data to the type of *movieRating*. **So the value of *movieRating* is 2.0.**

Q45 **The answer is B.**

As explained in **Q5** a *switch* statement can have any number of *case* statements including zero. Also, a *switch* statement can have one *default* statement if it is desired but not more than 1. Because this is against the nature of *default* statement which exists to be executed when none of the *case* statements match with the value of *switch*. **So the answer is B.**

Q46 **The answer is A.**

weather[0]!=null tries to check whether the first index of the array *weather* points to null or not. In order to be able to perform this check there must be the first index in the array. But the existence of that index is not checked. So if there are no elements in the array the application can throw an *ArrayIndexOutOfBoundsException.*

But if *weather[0]* index exists then the application will run without an issue. If the value stored in that index is “sunny” then the application will print “Go outside”, if not then it will print “Stay inside”. The expression *!false* returns *true* so that it does not affect result of AND expression.

So as a result the application either throws an exception, or prints “Go outside” or “Stay inside”. **The option A is not possible.**

Q47 **The answer is D.**

Logical complement operator, !, can only be applied to boolean values. In this code this operator is tried to be used with numeric values so the code does not compile.

Q48 **The answer is C.**

Logical operator of OR, II, evaluates to *true* if either of the operands is *true.* It evaluates to *false* if and only if both of the operands are *false*. In the table the values for *true* || *false* and *false* || *true* are missing, which would be *true.* **So the answer is *true* and *true.***

Q49 **The answer is A.**

The sentence begins with the *subtraction operator (-).* So it must continue with *addition/subtraction, multiplication/division/modulus* and *unary operators,* respectively in order to have them sorted according to their precedence in increasing level.

A – This option begins with *addition operator* and continues with *division* and *multiplication operators* which meets the requirement stated above. **So this option is correct.**

B – This option begins with a *unary operator* but continues with a *subtraction operator* which has a less precedence than *unary operators.* So this option is incorrect.

C – This option also begins with a *unary operator* but continues with *division operator* which has a less precedence than *unary operators.* This option is also incorrect.

D – This option begins with a *multiplication operator* followed by a *unary operator* which is okay for the requirement but continues with a *modulus operator* which has a less precedence than *unary operators.* This option is incorrect, too.

Q50 The answer is C.

The statement on line *p1* has a ternary operator evaluates to either 1 or 10 which are numeric values that cannot be assigned to a String type variable which is not allowed in Java. So this line causes the code not compile. **The answer is C.**

But if the last two expressions of the ternary operator on line *p1* are changed to String values such as *"1"* and *"10"*, then the code would compile. In that case the boolean expression in *if* statement will evaluate to *false* because *toy* will not be less than 2. So the *else* statement will be evaluated which will execute a ternary operator, too. This ternary operator will check if *age* is bigger than 3 which is not, so the boolean expression will evaluate to *false* the operator will evaluate the third operand which is "Swim".