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)

Ternary operator, is the only operator that takes three operands and is of the form:

- booleanExpression ? expression1 : expression2

The first operand must be a boolean expression, and the second and third can be any expression that returns a value. The ternary operation is really a condensed form of an if-then-else statement that returns a value.

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.

Q6) (B)

Ternary operation used to assign variable return 3. Pre increment operator inside if boolean expression, first increases variable and then the comparison is made. As the result, variable is the value of 4 after the execution of the code snippet.

Q7) (B)

As you saw when working with switch statements, a break statement transfers the fl ow of control out to the enclosing statement. The same holds true for break statements that appear inside of while, do-while, and for loops, as it will end the loop early.

break statement terminates the switch statement and return flow control to the enclosing statement. If it is left out the, flow will continue to the next proceeding case or default block automatically.

Q8) (C)

- Only boolean expressions are allowed on the leftmost operand of the ternary operator.
- Only one of the right-hand side expression gets evaluated in ternary operation, decided by the boolean value of the first ternary operand.
- Ternary expressions only support boolean expressions as the leftmost operand of the operator

The ternary operation is really a condensed form of an if-then-else statement that returns a value.

Q9) (C)

The code does not compile because, the && operator is undefined for the type Integer. We can use logical operator & instead of the shot-circuit version but then the returned value from the operator causes type mismatch.

Q10) (A)

Often, we only want to execute a block of code under certain circumstances. The **if-then** statement, accomplishes this by allowing our application to execute a particular block of code if and only if a boolean expression evaluates to true at runtime.

This boolean expression evaluates to false since the remainder from **6 % 3** expression is 0 and it is not greater than or equal to 1. Triceratops won't get incremented, but it is certainly get decremented because the post-decrement statement is not in the if-then block.

Q11) (D)

- Else statement is optional in **if-then** statements.
- There is no casting required in if-then statements. But it is true that if-then statements are used to check the objects type for the purpose of casting it to another type.
- If the boolean test of an if-then statement evaluates to false, target clase won't be evaluated, that would be the very opposite of the objective of if-then statements.
- It is true that if-then statement can execute a single statement or block, first block in which it's test evaluated to true gets executed and then the if-then or if-then-else statement is exited.

Q12) (D)

We should point out that there is two distinct if statements. Since, **flair** variable is 15, first if-then boolean check evaluates to **true** and "Not enough" will be printed. In the second if-then-else statement however, boolean check returns false but then the else block gets executed and prints "Too many".

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.

Q15) (C)

The structure of an if-then statement:

- `if(`**`booleanExpression`**`) { // Branch if true }`

The code does not compile because if checks only accepts expressions which returns and boolean value.

Q16) (B)

Increment and decrement operators, ++ and --, respectively, can be applied to numeric operands and have the higher order or precedence, as compared to binary operators. In other words, they often get applied first to an expression.Increment and decrement operators require special care because the order they are applied to their associated operand can make a difference in how an expression is processed. If the operator is placed before the operand, referred to as the pre-increment operator and the pre-decrement operator, then the operator is applied first and the value return is the new value of the expression. Alternatively, if the operator is placed after the operand, referred to as the post-increment operator and the post-decrement operator, then the original value of the expression is returned, with operator applied after the value is returned.

Q17) (B)

Unless overridden with parentheses, Java operators follow order of operation and if two operators have the same level of precedence, then Java guarantees left-to-right evaluation

We that parentheses used in order to override order of precedence. Addition operator is evaluated before multiplication operator so the result is 13.

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.

Q19) (D)

- It is often helpful for readability to add parentheses around the expressions in ternary operations, although it is certainly not required.
- Left most operand of the ternary operation must be a boolean expression.

Code does not compile because second ternary operation used as an operand to the first ternary operation is not valid. Left most operand must be a boolean expression but it has the type int.

Q20) ()
Q21) (B)

Operators with the same precedence follows operator associativity defined for its operator group. In java, operators, operators can either follow follow left-associative, right-associative or have no associativity. Operators with left-associative are evaluated from left to right, operators with right-associative are evaluated from right to left and operators with no associativity, does not follow any predefined order.

- `System.out.print(5 + 6 + "7" + 8 + 9);`

"+" : Is the only overloaded operator in Java which will concatenate Number to String. It is also left-associative. So 5 + 6 is both numeric so they are added, 11 + “7” evaluates to a concatenation since + operator is overloaded and “7” is a string. Result is “117” which is a string and so on, this causes chain reaction as follows. (numeric values in **bold**)

- “117” + **8** + **9** (remember + is left-associative)
- “117” + **8** = “1178” (concatenation)
- “1178” + **9** = “11789” (concatenation)

Q22) ()
Q23) ()

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.

Q30) (C)

Increment and decrement operators require special care because the order they are applied to their associated operand can make a difference in how an expression is processed. If the operator is placed before the operand, referred to as the pre-increment operator and the pre-decrement operator, then the operator is applied first and the value return is the new value of the expression. Alternatively, if the operator is placed after the operand, referred to as the post-increment operator and the post-decrement operator, then the original value of the expression is returned, with operator applied after the value is returned.

Q31) (A)

String type in java is a reference type, that is to say they point to objects. One of the use case of equality operators is 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**. The authors of the String class implemented a standard method called equals to check the values inside the String rather than the String 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.

Q32) (B)

12 + 6 * 3 % (1 + 1) evaluates to in order:

- 12 + 6 * 3 % 2

Multiplication and modulus operators have same precedence so java guarantees in that situation **operator evaluation** is from left to right.

- 12 + 18 % 2
- 12 + 0
- 12

Q33) (D)

Exclusive OR is only true if the operands are different. So, when p=true and q=false, p^q must be **true** and when p=false and q=false, p^q must be **false**.

Q34) (C)

data.length >= 1 and **data.length < 2** means that it will only evaluate true if the length of the array is exactly 1. We can pass just one argument to the program and hence have an array of String with length 1 and that argument can be "sound" or "logic" and so expression inside if evaluates to true. If the element is not "sound" or "logic" nothing will be printed. Since this is a short-circuit operator below part never gets evaluated if the left-hand side is false. Although we're not null checking specifically, given that the '**data**' array is program arguments **nullPointerException** is not expected.

Q35) (C)

According to the java precedence rules the correct answer is "**+**, **/**, *****, **%**, **++**".
/, ***** and **%**, operators have the same level of precedence. In option A unary decrement operator is falsely positioned. Option B has subtraction operator in the wrong position since it has lower precedence over modulus or multiplication operator. Lastly in option D, again, the subtraction operator is wrongly positioned.

Q36) (No correct option)

- **Exclusive OR** is only true if the operands are different.
- **AND** is only true if both operands are true.
- **Inclusive OR** is only false if both operands are false

The logical operators, (**&**), (**|**), and (**^**), may be applied to both numeric and boolean data types. When they're applied to boolean data types, they're referred to as logical operators. Alternatively, when they're applied to numeric data types, they're referred to as bitwise operators, as they perform bitwise comparisons of the bits that compose the number. The conditional operators are nearly identical to the logical operators, **&** and **|**, respectively, except that the right-hand side of the expression may never be evaluated if the final result can be determined by the left-hand side of the expression.

A more common example of where conditional operators are used is checking for **null objects** before performing an operation, such as this:

```
if(x != null && x.getValue() < 5) { // Do something}
```

In this example, if x was **null**, then the short-circuit prevents a NullPointerExceptionfrom ever being thrown, since the evaluation of **x.getValue() < 5** is never reached. Alternatively, if we used a logical **&**, then both sides would always be evaluated and when **x** was null this would throw an exception.

Q37) (C)

x || y (x or y) expression most closely represents the given venn diagram. Note that z is unrelated here. “Or” operator represents overlap in venn diagram, meaning any one of the overlapping diagram is suffice the expression to evaluate to true, hence both diagram is filled-in. For example **and** operator represents the part on the diagram where all terms are present.

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 indexOutOfBound 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.