Homework 5

Q1)(D)

A while loop is used to repeatedly execute a set of statements as long as its condition evaluates to true. This loop checks the condition before it starts the execution of the statement. Its syntax can be expressed as:

```
while (expression) {
     statement(s)
}
```

The while statement evaluates *expression*, which must return a boolean value. If the expression evaluates to true, the while statement executes the *statement*(s) in the while block. The while statement continues testing the expression and executing its block until the expression evaluates to false.

Q2)(B)

Traditional for loop is best known for using an index or counter.The for statement provides a compact way to iterate over a range of values. Programmers often refer to it as the "for loop" because of the way in which it repeatedly loops until a particular condition is satisfied. The general form of the for statement can be expressed as follows:

```
for (initialization; termination;
     increment) {
    statement(s)
}
```

Q3)(A)

The Java programming language provides a do-while statement, which can be expressed as follows:

```
do {
     statement(s)
} while (expression);
```

The difference between do-while and while is that do-while evaluates its expression at the bottom of the loop instead of the top. Therefore, the statements within the do block are always executed at least once.

Q4)(C)

The Java for-each loop or enhanced for loop is introduced since J2SE 5.0. It provides an alternative approach to traverse the array or collection in Java. It is mainly used to traverse the array or collection elements. The advantage of the for-each loop is that it eliminates the possibility of bugs and makes the code more readable. It is known as the for-each loop because it traverses each element one by one.

Q5)(B)

The continue statement skips the current iteration of a for, while , or do-while loop. The unlabeled form skips to the end of the innermost loop's body and evaluates the boolean expression that controls the loop.

Q6)(A)

When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

The Java *break* statement is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.

Q7)(B)

A for loop has 3 parts: initialization, condition, and change. The parts are separated by semicolons (;).

```
for (initialization; condition; change)
```

Each of the three parts is optional, but the semicolons are not optional.The code in the initialization area is executed before the first execution of the loop, the condition is checked each time through the loop and the loop continues as long as the condition is true, at the end of each execution of the body of the loop the changes are done.

Q8)(C)

We can use traditional for loop to iterate through an array starting from the end by initializing index to the value which is the last index in the array and then decrement it until it reaches 0, which is the start of the array. Since we can control the initialization, condition and change we can start looping from end, or at index 0 or even at some other index.

Q9)(A)

For-each loop, unlike traditional for loop only iterates forward over the array in single steps.

Q10)(A)

Unlike while statement, do-while evaluates its expression at the bottom of the loop instead of the top. Therefore, the statements within the do block are always executed at least once.

Q11)(B)

The code does not compile. The while statement evaluates *expression*, which must return a boolean value.

Q12)(B)

The *Java List* interface, java.util.List, represents an ordered sequence of objects. The elements contained in a Java List can be inserted, accessed, iterated and removed according to the order in which they appear internally in the Java List. The ordering of the elements is why this data structure is called a *List*.

The Java List interface, java.util.List, represents an ordered sequence of objects. Each element in a Java List has an index. The first element in the List has index 0.

Q13)(A)

In the first iteration in which the type value is 0, first element from the list is acquired and printed, then with the break statement loop is exited. Note that neither increment nor decrement statement is given to the loop statement so if there wasn't a break statement, the loop would go on forever.

Q14)(A)

The code outputs "aa". The variable letters is initialized with an empty string. While statement boolean expression check if the string length isn't equal to 2. It keeps executing the while statement code block while the string length is not equal to 2. Inside the code block letter variable is concatenated with string "a", after two iteration variable becomes "aa" and length is 2, boolean expression returns false and the while statement code blocks does not get executed.

Q15)(D)

when run with `java peregrine.TimeLoop September 3 1940` the code would caught in an infinite loop because `i` is initialized with the length of the argument array given and then the condition `i >= 0` is always going to be true considering `i` is get incremented not decremented after every iteration.

Q16)(B)

Class and instance variables in java get their default values of they are not initialized. count variable get the default value 0 in this case. While loop basically traverses through the String array using the count variable, checking the every element’s length if it’s smaller than 8 or not. If it is smaller than 8 then it breaks the while loop and print the count variable.

First element in the array which is “Washington” is a String of length 10, so the iteration continues.On second iteration count is 1,second element’s length is smaller than 8 so if evaluates to true and the if block get executed which exits the loop and count gets incremented. count is incremented two times because the iteration is branched two times. So at the time of the print the count variable is 2.

Q17)(C)

The code does not compile because count is declared inside the first do-while loop and it is not reachable outside of the scope of the do-while loop. It is just in the scope of the first do-while loop. By the way if we would declared the count variable outside of the do-while, it seem like the code would run in an infinite loop but that’s not:W correct. Notice the break statement inside the first do-while loop. This would break the do-while loop.

Q18)(D)

All of the segments given below are optional in traditional for loops.

```
for (segmentA; segmentB; segmentC)
```

For example for(;;) would run in an infinite loop, Since there is not any condition that prevents it from running in an infinite loop.

Q19)(C)

Only with the enhanced for loop does not allow us to write code that creates an infinite loop. Enhanced for loop iterates over an iterable objects and it only iterates one step forward each time until there are no elements left to iterate.

Q20)(A)

This traditional for loop basically iterates through the list and printing each element using the container variable as index.

Q21)(D)

The code does not compile because of syntax error, parentheses are not allow to cover loop bodies, braces should have been used instead. Note that if the block is a one-liner braces is optional.

Q22)(B)

Option A and option C are wrong since there is no such syntax. Equivalent code for the given loop which iterates over the array and prints its elements is given in option C as follows:

```
for (String arrayElement: fun)
            System.out.println(arrayElement);
```

Q23)(C)

An unlabeled break statement terminates the innermost switch, for, while, or do-while statement, but a labeled break terminates an outer statement. Here both "`break number;`" and "`break;`" statements ends the inner loop and hands control to the outer loop labeled letters.

Q24)(B)

The `continue` statement skips the current iteration of a `for`, `while` , or `do-while` loop. The unlabeled form skips to the end of the innermost loop's body and evaluates the boolean expression that controls the loop. A labeled `continue` statement skips the current iteration of an outer loop marked with the given label. We want to skip to the outer loop in this case so only `continue letters;` is going to do what is being asked.

Q25)(C)

The code completes with no output. Variable singer is assigned to primitive type int 0. Boolean expression checks if the singer variable is bigger than 0, since it is not bigger than 0 while block never gets executed hence no output is printed.

Q26)(C)

Enhanced for loop can only iterate over an array or an instance of java.lang.Iterable. ArrayList is an instance of Iterable type, int[] is certainly an array but StringBuilder is not an Iterable so can not be used as a subject to an enhanced for loop.

Q27)(B)

The code outputs inflate-done. Boolean value `balloonInflated` initialized with false at first. in the do-while block `balloonInflated` is checked if it is true or false and then assigned to true if it is false which it is and “inflate-” is printed. When the while statement check the boolean expression in itself balloonInflated is already assigned to true even if it wasn’t at first. Lastly “done” is printed to screen.

Q28)(D)

This code is an infinite loop because; while statement gets executed since the variable length at first is not equal to three it is zero, since it concatenates 2 length string every time length of the variable letters is never going to be 3. It will alway be an even number. SO the boolean expression is never going to evaluate to false.

Q29)(B)

The for loop has three parts: initialization expression is the first part, boolean condition is the second and the update statement is the last part of a for loop.

```
for (initialization; condition; change)
```

Q30)(B)

The code outputs the size of the List which is 4. do-while executes until the count variable is not bigger than 0. First the code adds character ‘a’ to the list and than it decrements the count variable for every element in the list, which is 1 in the first iteration. After the first iteration count gets decremented by 2 because there is two elements in the list now. This goes on with the sequence 1-9, 2-7, 3-4, 4-0 for list size and count variable respectively.

Q31)(A)

The code in the question outputs 1. While code block gets executed in the first iteration of the for loop and it decrements the `i` variable of the **`for`** loop down to 1. `k` variable get incremented by 1 and at the end of the first iteration `i` is decremented by 1 as the update statement states in the **`for`** loop statement. So essentially for loop only run once, hence the `k` is 1 when it's printed to the screen.

Q32)(D)

Option A and option C are syntactically wrong representations of enhanced for loop. option B is fine but enhanced for loop only iterates over the array forwards, not backwards, so none of them are equivalent the for loop given in the example.

Q33)(C)

The code does not compile because no braces are used for the for loop so only one statement can follow but there is a break statement too after a print statement. Essentially **`break`** is out of for statement and break can only be used inside of a loop or switch statement.

Q34)(C)

The code does not compile. Multiple change statements must be separated with comma not with semicolon. Having the syntax error corrected code would print "Downtown Day-Uptown Night-". `i variable` would never be 2 because of the condition statement, right hand of the && operator that checks `j` variable is smaller than the size of the `times` array prevents that, so third element in the `nycTourLoops` will not be iterated over.

Q35)(D)

The code compiles fine but when we try to run it with arguments or no arguments at all, code will throw `indexOutOfBoundException` because `i` is assigned to the `length` of the array but the last element's index is `length-1` because array index starts at 0. We are trying to access an index which is out of the bound of the array.

Q36)(B)

Since tie is assigned to **`null`**, while condition evaluates to **`true`** and the while block will get executed. First statement inside the while block re-assigns `tie` variable to a non-null variable, assigning a non-null value to `tie` guarantees while block will not get executed again. After while statement `tie` variable gets printed to the screen.

Q37)(C)

The code no longer compiles because if we remove the surrounding do-while loop, it leaves the break statement out in the open which should used in a loop or switch statement, also a label should point to a loop bu there is no loop statement anymore.

Q38)(C)

The code would print `4`. It iterates through the array, incrementing count with every if-then statement. Note **`continue`** statement does not have any effect here since the `count` is incremented in the boolean expression not after if statement and there is no any other statement to execute inside while block after continue statement that would affect the result. **`continue`** does not skipping any code since there is none that it could.

Q39)(C)

The code does not compile. While condition need to be type boolean or some expression that evaluates to a boolean value.

Q40)(A)

The code prints 2. Inner do-while loop runs two times resulting with `count` variable equals to value of 2. While the outer do-while loop have boolean literal **`true`** as a condition, outer loop only runs once because of the **`break`** statement. This makes outer do-while loop redundant.

Q41)(C)

Both while loop have an boolean literal **`true`** which causes code to go into an infinite loop. To prevent the code from going into an infinite loop, we need to terminate the outer while loop, to do that we need labeled break statement which enables us to terminate the outer loop instead of the inner loop.

Q42)(B)

This code is the corrected version of the one in the question 34. Code will print "Downtown Day-Uptown Night-". `i variable` would never be 2 because of the condition statement, right hand of the `&&` operator that checks `j` variable is smaller than the size of the `times` array prevents that, so third element in the `nycTourLoops` will not be iterated over.

Q43)(B)

The code prints 4 line. It iterates over the same list. Every loop runs twice so the print statement will run four times hence it will output four lines. Note that more than one line comes after the first for loop which does not use brackets but there is actually only one statement which is the inner for loop so it is syntactically correct.

Q44)(B)

First, the initializer gets executed in a for statement. Second, boolean condition is checked and after the block is executed (if the condition evaluates to true), update statement get executed lastly. But since the condition is given as `false` in the question delta never gets executed, so the flow can be described as alpha, beta respectively.

Q45)(D)

We know that the loop runs exactly once. Alpha runs first and only once, then the beta is checked which we know that it returned true in the first time. This results in execution of the delta block. After the execution of delta block, update statement runs, then again the beta condition is checked which obviously returns false at this point because we know that the loop only run once. The following best describes the flow of the execution of the for loop given in the question:

```
alpha, beta, delta, gamma, beta
```

Q46)(C)

Option C given below iterates 6 times instead of five because it first executes the block and then check the condition unlike the other for loops.

```
int k=0; do { } while(k++ < 5)
```

Q47)(D)

The code runs in an infinite loop since the while condition is `true` and statement is terminated early with the semicolon, meaning that the loop body is empty and the assignment statement which is going to set `tie` variable to a non-null variable is never going to be reached.

Q48)(C)

The code does not compile. We cannot use java reserved keywords as labels for loop statements. Having the syntax error corrected, code would print 5. Notice the break statement is unnecessary since after the inner loop, outer while condition is going to evaluate to false anyway.

Q49)(D)

This code goes into an infinite loop. In the first iteration `ballonInflated` is assigned to true. **`while`** condition checks if it’s true and continues executing the loop. Because `balloonInflated` variable is now **`true`**, if-then block will not get executed. There is no statement to change the boolean value of the variable and since it is **`true`** the loop will go on forever.

Q50)(B)

All the other options other than the option B have incorrect syntax. Semicolon separates statements not variables. Type of the variable should only be declared once. WIth option B, code compiles and executes, loop iteration run only once.