

ASSIGNMENT 5
Java Loop Constructs
(HW 5)

ConcurrentModificationException in Java

The ConcurrentModificationException occurs when an object is tried to be modified concurrently when it is not permissible. This exception usually comes when one is working with **Java Collection classes**.

For Example - It is not permissible for a thread to modify a Collection when some other thread is iterating over it. This is because the result of the iteration becomes undefined with it. Some implementation of the Iterator class throws this exception, including all those general-purpose implementations of Iterator which are provided by the JRE. Iterators which do this are called **fail-fast** as they throw the exception quickly as soon as they encounter such situation rather than facing undetermined behavior of the collection any time in the future.

How to avoid ConcurrentModificationException in a multi-threaded environment?

To avoid the ConcurrentModificationException in a multi-threaded environment, we can follow the following ways-

1. Instead of iterating over the collection class, we can iterate over the array. In this way, we can work very well with small-sized lists, but this will deplete the performance if the array size is very large.
2. Another way can be locking the list by putting it in the synchronized block. This is not an effective approach as the sole purpose of using multi-threading is relinquished by this.
3. JDK 1.5 or higher provides with ConcurrentHashMap and CopyOnWriteArrayList classes. These classes help us in avoiding concurrent modification exception.

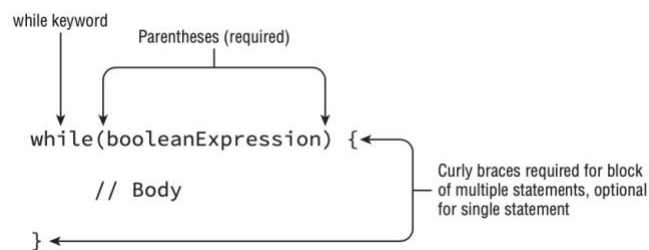
How to avoid ConcurrentModificationException in a single-threaded environment?

By using iterator's remove() function, you can remove an object from an underlying collection object.

****<https://www.javatpoint.com/concurrentmodificationexception-in-java>**

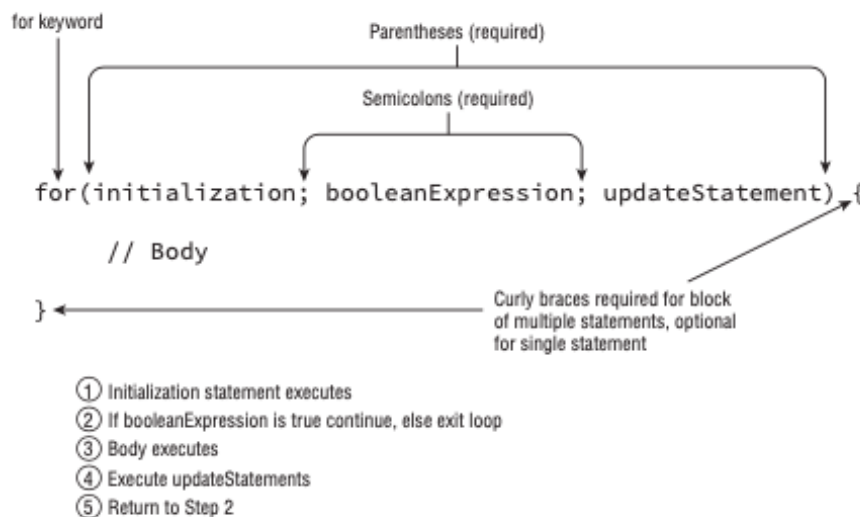
****<https://docs.oracle.com/javase/8/docs/api/index.html?java/util/ConcurrentModificationException.html>**

Q1-) Option D



I think the **Option D** is the answer. This is the structure of a while statement. During execution, the boolean expression is evaluated before each iteration of the loop and exits if the evaluation returns false. It is important to note that a while loop may terminate after its first evaluation of the boolean expression. In this manner, the statement block may never be executed. Unlike a while loop, though, a do-while loop guarantees that the statement or block will be executed at least once, because boolean condition on a do-while loop is at the end of the loop. It does not control entry to the loop. So, Option A is incorrect. A traditional for loop also has a boolean condition that controls to entry to the loop with a counter. This Option may be the answer but I think it not the best choice. Option C cannot be the answer because there is no condition as part of the for-each loop.

Q2-) Option B



This is the structure of a for loop.

initialization:

is executed (one time) before the execution of the code block.

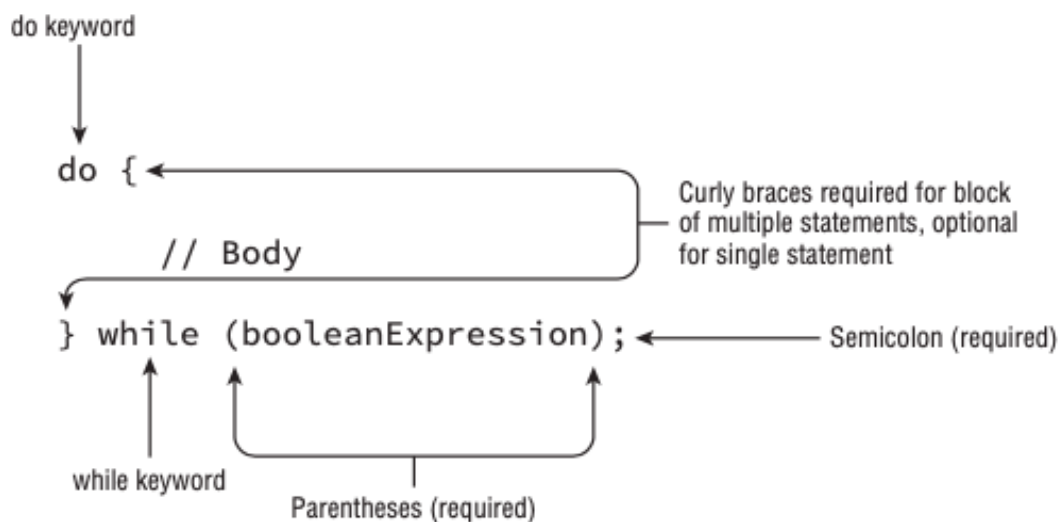
booleanExpression:

defines the condition for executing the code block.

updateStatement:

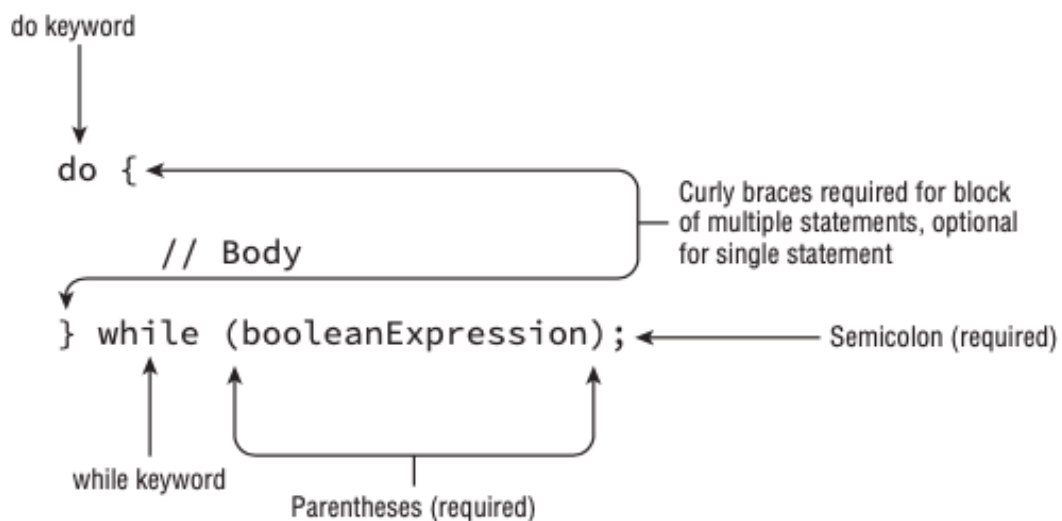
is executed (every time) after the code block has been executed.

A for loop has a loop variable counting up or down as the loop progresses. **Option B** is the answer.

Q3-) Option A

This is the structure of a do-while loop. As explained in Q1; unlike a while loop, a do-while loop guarantees that the statement or block will be executed at least once. Because, it checks the boolean condition after execution of the body. **Option A** is the answer.

Q4-) Option C



This is the structure of a for-each loop. The for-each loop declaration is composed of an initialization section and an object to be iterated over. The right-hand side of the for-each loop statement must be a built-in Java array or an object whose class implements `java.lang.Iterable`, which includes most of the Java Collections framework. The left-hand side of the for-each loop must include a declaration for an instance of a variable, whose type matches the type of a member of the array or collection in the right-hand side of the statement. On each iteration of the loop, the named variable on the left-hand side of the statement is assigned a new value from the array or collection on the right-hand side of the statement. **It can loop through an array without referring to the elements by index**, unlike a for loop. **Option C** is the answer.

Q5-) Option B

Option A: `break`

The `break` keyword is used to break out a `for` loop, a `while` loop or a `switch` block.

Option B: `continue`

`continue` keyword is used to end the current iteration in a `for` loop (or a `while` loop), and continues to the next iteration. This is the answer.

Option C: `end` & Option D: `skip`

These are not keywords.

Q6-) Option A

As explained in Q5, a "break" keyword is used to proceed with execution immediately after a loop. **Option A** is the answer.

Q7-) Option B

As explained in Q2, a traditional "for loop" has three segments within parentheses that are initialization, booleanExpression, updateStatement. **Option B** is the answer.

Q8-) Option C

- A traditional for loop can iterate through an array starting from index 0.
- A traditional for loop can iterate through an array starting from the end.

Both statements are true. A "traditional for loop" can iterate through an array starting from the first or the last index. **Option C** is the answer.

Q9-) Option A

- A for-each loop can iterate through an array starting from index 0.
- A for-each loop can iterate through an array starting from the end.

The first statement is true. A "for-each loop" can iterate through an array starting from index 0. This is the determined order. Yet, it cannot iterate through an array starting from the end. This is the ability of "traditional for loop". **Option A** is the answer.

Q10-) Option A

As explained in Q1&Q4, a "do-while" loop has a boolean condition that is first checked after a single iteration through the loop. **Option A** is the answer.

Q11-) Option B

```
int singer = 0;
while (singer)
System.out.println(singer++);
```

This code does not compile because a boolean condition is needed such as "**singer<5**". **Option B** is the answer.

Q12-) Option B

```
List<String> drinks = Arrays.asList("can", "cup");

for (int container = drinks.size() - 1; container >= 0;
container--)

    System.out.print(drinks.get(container) + ",");
```

The code compiles and outputs "cup, can,". "drinks.size() - 1" is "1" and it is the last index of the list. This loop prints elements of a list from end to start. **Option B** is the answer.

Q13-) Option A

The code compiles and outputs "glass, end" When the first time through the loop, it outputs "glass" because index is 0. After that, the "break" keyword skips all the other executions and breaks out this (infinite)loop. Finally, the line "System.out.print("end");" prints "end". **Option A** is the answer.

Q14-) Option A

At the beginning, "letters.length()" is 0, and each execution of the loop increments it by 1. So, this loop is entered 2 times during the program. Every execution of this loop adds "a" to the letters and after the loop ends it will be "aa". **Option A** is the answer.

Q15-) Option D

```
java peregrine.TimeLoop September 3 1940?
```

Since the code produces an **infinite loop**, **Option D** is the answer. Program starts with "3" arguments so, "i=3" and it increments by one after each execution. So, it is always greater than "0". This makes the loop infinite.

Q16-) Option B

"count" is defaults to "0" because it is a class variable. When the first execution of the loop if statement's body is not run because "Washington" consists of 11 characters. On the next iteration, if statement checks index 1 and makes count "2". The second element of the array "Monroe" is 6 characters so, if statement's body is run and it breaks out the loop. After the loop ends, the code prints "count" which is "2". **Option B** is the answer.

Q17-) Option C

This code does not compile because "count" is declared inside a loop and it is not accessible from outside of the loop. **Option C** is the answer.

Q18-) Option D

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

All three statements can be left blank. If we do this, this will be an infinite loop. **Option D** is the answer.

Q19-) Option C

As explained in Q18, "for(;;) {}" is an infinite loop. In order to this do{} while(true) and while(true) {} are also infinite loops. Unlike this loops, a "for-each" loop does not allow us to write code that creates an infinite loop. Because, it loops through an array or ArrayList and there is no infinite array or ArrayList. **Option C** is the answer.

Q20-) Option A

This loop starts from beginning of the "drinks" and goes to the last index. In every execution, it prints the element of that index. So, the "can, cup" is the output and **Option A** is the answer.

Q21-) Option D

```
do (  
  
System.out.println("helium");  
  
) while (false);
```

This code does not compile. Because the body of a loop cannot be surrounded by parentheses. This is not legal. Braces should be used to do this. **Option D** is the answer.

Q22-) Option B

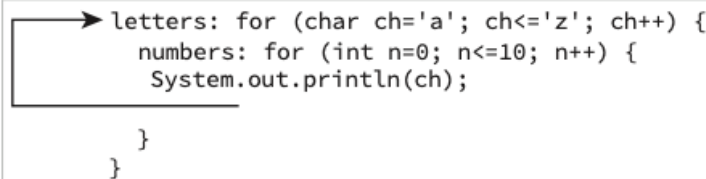
```
for (int i=0; i<fun.length; i++)  
    System.out.println(fun[i]);
```

This code snippet prints the elements of the “fun” array from beginning to end. We can do this with a “for-each” loop and its valid syntax is as follows:

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

Option B is the answer.

Q23-) Option C



```
letters: for (char ch='a'; ch<='z'; ch++) {  
    numbers: for (int n=0; n<=10; n++) {  
        System.out.println(ch);  
    }  
}
```

In order to follow the arrow, we must break out of the inner loop. To do this, we can write “break;” and “break numbers”. If we use “break letters” both loops are ends. **Option C** is the answer.

Q24-) Option B

As explained above, in order to follow the arrow we must break out of the inner loop. To do this, we can write “continue letters”. **Option B** is the answer. “continue numbers” & “continue” statements resume execution at the inner loop.

Q25-) Option C

```
int singer = 0;
while (singer > 0)
    System.out.println(singer++);
```

This code completes with no output because the loop is never executed. The condition is always false and **Option C** is the answer.

Q26-) Option C

```
for (Object obj : taxis) { }
```

This is a for-each loop and it is allowed and arrays and ArrayLists are can be used in a for-each. So, we can compile this snippet with `ArrayList<Integer> & int[]` but the code does not compile with a `"StringBuilder"`. **Option C** is the answer.

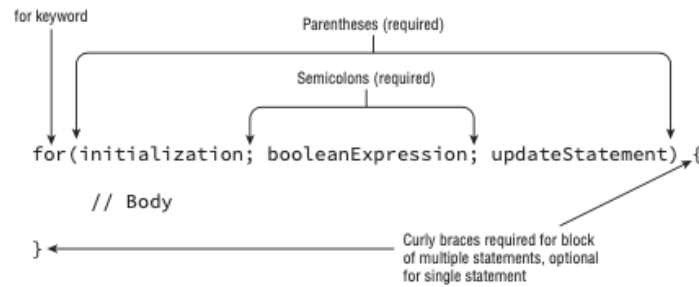
Q27-) Option B

This do-while is legal. The code compiles and outputs "inflate-done". While "balloonInflated" is false, "!balloonInflated" is true so, the "if" statement executes and prints "inflate-" and makes "balloonInflated" false. This means that the loop will never run again and programs ends with "inflate-done" output. **Option B** is the answer.

Q28-) Option D

At the beginning, `letters.length()` is equals to "0" so, the loop is entered. Each execution of this loops increases `letters.length()` by "2", it never will be "3" and the loop never ends. This is an infinite loop, **Option D** is the answer.

Q29-) Option B



initialization expression, boolean conditional, update statement is the correct order in a for loop. **Option B** is the answer.

Q30-) Option B

On the first iteration inner loop is executed 1 time and makes count 9. On the second iteration, the inner loop executes two times and makes count 7. The next iteration, it executes 3 times and makes count 4. On the next, since “chars” becomes 4 elements, the inner loop runs four times so count becomes 0 and both loops ends. **Option B (4)** is the answer.

Q31-) Option A

At the beginning “`i=10`”, the inner loop is entered and makes “7” the value of “`i`”. Since “`i`” still greater than “3”, the inner loop is executed again and sets “`i`” to “4”. After that it executed again because “`4 > 3`”. This execution makes the value of “`i`” 1 and the inner loop is never executed again. Finally, the value of “`k`” is incremented by “1” and it becomes “1”. This is the output, **Option A** is the answer.

Q32-) Option D

```
for (int i=fun.length-1; i>=0; i--)  
System.out.println(fun[i]);
```

This code snippet prints elements of “`fun`” array in descending order. We cannot do this by using a “for-each” loop because a “for-each” only allows us to print in ascending order. **Option D** is the answer.

Q33-) Option C

If the loop body is not surrounded by curly brackets, the loop body is only one statement. So, break statement is outside of the loop. The code does not compile because break statement is not allowed there. **Option C** is the answer.

Q34-) Option C

The code does not compile because, multiple update statements should be separated with a “;” not a “,”. **Option C** is the answer. If we fix this error it compiles and outputs “Downtown Day-Uptown Night-”.

Q35-) Option D

```
java peregrine.TimeLoop September 3 1940
```

The class called with “3” arguments, so $i=3$ on the first iteration. When attempting to print `args[3]`, Java throws an **ArrayIndexOutOfBoundsException** because `args[2]` is the last element of the array.

Q36-) Option B

Since `String tie="null"`, condition is true. While loop is executed one time, and “shoelace” is assigned to `tie`. This makes the condition false and the loop never executes again. “shoelace” is the output, **Option B** is the answer.

Q37-) Option C

If we remove the lines 25 and 28, the `break` statement remains outside of the loop. This is not a legal. The code does not compile because `break` statement is not allowed there. **Option C** is the answer.

Q38-) Option C

The code compiles and outputs “4”. **Option C** is the answer. The “`continue`” statement is useless here, it is redundant. At the beginning, “`count`” defaults to “0” because it is a class variable. Until it is equal to “`stops.length`”, the loop is executed 4 times. Each iteration increments “`count`” by “1”. So, it becomes 4 finally.

Q39-) Option C

This code does not compile because a boolean condition is needed. “`builder`” is a `StringBuilder` object, not a boolean. **Option C** is the answer.

Q40-) Option A

Since there is a break statement, this is not an infinite loop. Until count is equal to "2", the inner loop is executed 2 times. Each iteration increments "count" by "1". So, it becomes 2 finally. **Option A** is the answer.

Q41-) Option C

```
t: while (true) {  
    f: while(true) {  
        _____  
    }  
}
```

To compile this code without causing an infinite loop, we should end both loops. "break;" "break f;" statements only ends the inner loop, these are not the correct choices. "break t;" is ends both of these loops, **Option C** is the answer.

Q42-) Option B

The code compiles and outputs "Downtown Day-Uptown Night-". **Option B** is the answer. The loop is executed 2 times and it prints first 2 elements of arrays.

Q43-) Option B

These are nested loops. On each execution of the outer loop, the inner loop is executed 2 times. Since the array list has 2 items, the inner loop is executed 4 times. So, **Option B** is the answer.

Q44-) Option B

```
for (alpha; beta; gamma) {  
    delta;  
}
```

"beta" is the condition statement. If it is false, the loop is never entered. **Option B** is the answer.

Q45-) Option B

```
for (alpha; beta; gamma) {  
    delta;  
}
```

If the loop body is run exactly once, flow of execution in this for loop is as follows:

"alpha, beta, delta, gamma, beta"

The initializer runs first. After that condition statement is checked and body is run. Then, update statement is run and condition is checked again. . **Option B** is the answer.

Q46-) Option C

- `for (int k=0; k < 5; k++) {}`
- `for (int k=1; k <= 5; k++) {}`
- `int k=0; while (k++ < 5) {}`

These three loops is executed 5 times. The first and third loops go through from index 0 to 4, the second one is goes through from index 1 to 5.

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

This a "do-while" loop. The body is runs first, then the condition is checked and it iterates from index 0 to 4, 5 times more. **Option C** is the answer.

Q47-) Option D

Since there is a ";" colon after the while, the body of while is empty. The condition is always true so, this is an infinite loop. A new value is never assigned to "tie" at remains as "null" forever. **Option D** is the answer.

Q48-) Option C

The code does not compile, because for is a keyword and it is not allowed to use for a label of a loop. If we change it with a valid label such as "f", the code compiles and outputs "5". **Option C** is the answer.

Q49-) Option D

When the first run of the body do-while, "balloonInflated" becomes "true" and this makes this loop an infinite loop. **Option D** is the answer.

Q50-) Option B

Java allows us to initialize multiple variables in a for loop by separating them with a comma. "int i=0, j=0;" is the valid syntax, **Option B** is the answer.