**CDAC MUMBAI**

**Lab Assignment**

**SECTION 1: Error-Driven Learning Assignment: Loop Errors**

**Instructions:**

**Analyze each code snippet for errors or unexpected behavior. For each snippet, determine:**

**1. Why does the error or unexpected behavior occur?**

**2. How can the code be corrected to achieve the intended behavior?**

**Snippet 1:**

**public class InfiniteForLoop {**

**public static void main(String[] args) {**

**for (int i = 0; i < 10; i--) {**

**System.out.println(i);**

**}**

**}**

**}**

**// Error to investigate: Why does this loop run infinitely? How should the loop control variable be adjusted?**

### Snippet 1: `InfiniteForLoop`

```java

public class InfiniteForLoop {

public static void main(String[] args) {

for (int i = 0; i < 10; i--) {

System.out.println(i);

}

}

}

```

\*\*Error:\*\*

- The loop runs infinitely because the loop control variable `i` is being decremented (`i--`) instead of incremented. As a result, `i` is moving further away from the loop termination condition (`i < 10`), rather than approaching it. This causes the loop to never satisfy the exit condition, leading to an infinite loop.

\*\*Correction:\*\*

- To make the loop terminate correctly, the loop control variable should be incremented (`i++`) so that `i` eventually reaches `10` and the loop condition `i < 10` becomes `false`, stopping the loop:

```java

for (int i = 0; i < 10; i++) {

System.out.println(i);

}

```

- This correction will allow the loop to execute 10 times, printing values from `0` to `9`.

================================================================================

**Snippet 2:**

**public class IncorrectWhileCondition {**

**public static void main(String[] args) {**

**int count = 5;**

**while (count = 0) {**

**System.out.println(count);**

**count--;**

**}**

**}**

**}**

**// Error to investigate: Why does the loop not execute as expected? What is the issue with the condition in the**

**`while` loop?**

### Snippet 2: `IncorrectWhileCondition`

```java

public class IncorrectWhileCondition {

public static void main(String[] args) {

int count = 5;

while (count = 0) {

System.out.println(count);

count--;

}

}

}

```

\*\*Error:\*\*

- The condition `while (count = 0)` is incorrect because it uses the assignment operator `=` instead of the equality operator `==`. This code assigns `0` to the variable `count` and then evaluates the condition. In Java, the result of an assignment (`=`) is the value that was assigned, which in this case is `0`. Since `0` is considered `false` in a boolean context, the loop never executes.

\*\*Correction:\*\*

- To check if `count` is equal to `0`, you need to use the equality operator `==`:

```java

while (count == 0) {

System.out.println(count);

count--;

}

```

- If you intend for the loop to run while `count` is greater than `0`, the condition should be:

```java

while (count > 0) {

System.out.println(count);

count--;

}

```

This corrected version will print the value of `count` until it decreases to `0`, after which the loop will terminate.

================================================================================

**Snippet 3:**

**public class DoWhileIncorrectCondition {**

**public static void main(String[] args) {**

**int num = 0;**

**do {**

**System.out.println(num);**

**num++;**

**} while (num > 0);**

**}**

**}**

**// Error to investigate: Why does the loop only execute once? What is wrong with the loop condition in the `do-**

**while` loop?**

### Snippet 3: `DoWhileIncorrectCondition`

```java

public class DoWhileIncorrectCondition {

public static void main(String[] args) {

int num = 0;

do {

System.out.println(num);

num++;

} while (num > 0);

}

}

**```**

\*\*Error:\*\*

- The `do-while` loop executes at least once because the condition is checked after the loop body is executed. In this case, the loop only runs once because the condition `num > 0` becomes `true` after the first iteration when `num` is incremented to `1`. The loop then exits because the condition is `false` on the second check.

\*\*Why Does This Happen?\*\*

- Initially, `num` is `0`, so the loop prints `0` and increments `num` to `1`. After this, the condition `num > 0` is `true`, and since it's a `do-while` loop, it checks the condition after each iteration. As `num` is now `1`, the loop stops executing because the condition `num > 0` is `true`, leading to an exit from the loop after the first iteration.

\*\*Correction:\*\*

- If the intention is to keep the loop running for a specific number of iterations, you should adjust the loop condition. For example, if you want the loop to run while `num` is less than a certain value, change the condition accordingly:

```java

do {

System.out.println(num);

num++;

} while (num < 10); // Example condition to loop while num is less than 10

```

- This corrected loop will print values from `0` to `9` before exiting, as the condition `num < 10` will become `false` once `num` reaches `10`.

================================================================================

**Snippet 4:**

**public class OffByOneErrorForLoop {**

**public static void main(String[] args) {**

**for (int i = 1; i <= 10; i++) {**

**System.out.println(i);**

**}**

**// Expected: 10 iterations with numbers 1 to 10**

**// Actual: Prints numbers 1 to 10, but the task expected only 1 to 9**

**}**

**}**

**// Error to investigate: What is the issue with the loop boundaries? How should the loop be adjusted to meet the**

**expected output?**

### Snippet 4: `OffByOneErrorForLoop`

```java

public class OffByOneErrorForLoop {

public static void main(String[] args) {

for (int i = 1; i <= 10; i++) {

System.out.println(i);

}

// Expected: 10 iterations with numbers 1 to 10

// Actual: Prints numbers 1 to 10, but the task expected only 1 to 9

}

}

```

\*\*Error:\*\*

- The loop currently prints numbers from `1` to `10`, which is consistent with the loop condition `i <= 10`. However, if the expected output is to print numbers from `1` to `9`, this loop runs one iteration too many.

\*\*Correction:\*\*

- To print numbers from `1` to `9`, the loop condition should be adjusted to stop before `10`. The correct loop condition would be `i < 10`:

```java

for (int i = 1; i < 10; i++) {

System.out.println(i);

}

```

- This will print the numbers from `1` to `9` as expected.

\*\*Summary:\*\*

- The issue is with the loop boundary condition. Changing the condition from `i <= 10` to `i < 10` ensures that the loop stops before `10`, producing the desired output of numbers from `1` to `9`.

================================================================================

**Snippet 5:**

**public class WrongInitializationForLoop {**

**public static void main(String[] args) {**

**for (int i = 10; i >= 0; i++) {**

**System.out.println(i);**

**}**

**}**

**}**

**// Error to investigate: Why does this loop not print numbers in the expected order? What is the problem with the**

**initialization and update statements in the `for` loop**

### Snippet 5: `WrongInitializationForLoop`

```java

public class WrongInitializationForLoop {

public static void main(String[] args) {

for (int i = 10; i >= 0; i++) {

System.out.println(i);

}

}

}

```

\*\*Error:\*\*

- The loop is intended to print numbers in descending order from `10` to `0`, but it doesn't work as expected. The problem lies in the loop's update statement `i++`, which increments `i` instead of decrementing it. This causes the loop condition `i >= 0` to always be true after the first iteration, leading to an infinite loop where `i` continues to increase.

\*\*Correction:\*\*

- To print the numbers in descending order, you need to decrement `i` in each iteration. The correct update statement is `i--`:

```java

for (int i = 10; i >= 0; i--) {

System.out.println(i);

}

```

- This will print the numbers from `10` down to `0`, as the loop control variable `i` decreases with each iteration.

\*\*Summary:\*\*

- The issue is with the loop's update statement. It should decrement the loop control variable (`i--`) rather than increment it (`i++`). The corrected loop will properly print the numbers in descending order from `10` to `0`.

================================================================================

**Snippet 6:**

**public class MisplacedForLoopBody {**

**public static void main(String[] args) {**

**for (int i = 0; i < 5; i++)**

**System.out.println(i);**

**System.out.println("Done");**

**}**

**}**

**// Error to investigate: Why does "Done" print only once, outside the loop? How should the loop body be enclosed to**

**include all statements within the loop?**

### Snippet 6: `MisplacedForLoopBody`

```java

public class MisplacedForLoopBody {

public static void main(String[] args) {

for (int i = 0; i < 5; i++)

System.out.println(i);

System.out.println("Done");

}

}

```

\*\*Error:\*\*

- In this code, the statement `System.out.println("Done");` is not part of the loop body because it is not enclosed in braces `{}`. Therefore, it is executed only once after the loop has completed, rather than being part of the loop and executing with each iteration.

\*\*Why Does This Happen?\*\*

- In Java, if a `for` loop (or any loop or conditional) does not have braces `{}`, only the first statement following the loop is considered to be inside the loop. The second `System.out.println("Done");` is outside the loop and only runs after the loop has finished.

\*\*Correction:\*\*

- To ensure that `"Done"` is printed on each iteration of the loop, you need to enclose both statements within curly braces `{}` to form a proper block of code within the loop:

```java

for (int i = 0; i < 5; i++) {

System.out.println(i);

System.out.println("Done");

}

```

- This will cause the loop to print the numbers from `0` to `4`, followed by `"Done"` after each number.

\*\*Summary:\*\*

- The issue arises because the loop body is not properly enclosed in braces. Enclosing the statements in `{}` ensures that both `System.out.println(i);` and `System.out.println("Done");` are executed within each iteration of the loop.

================================================================================

**Snippet 7:**

**public class UninitializedWhileLoop {**

**public static void main(String[] args) {**

**int count;**

**while (count < 10) {**

**System.out.println(count);**

**count++;**

**}**

**}**

**}**

**// Error to investigate: Why does this code produce a compilation error? What needs to be done to initialize the loop**

**variable properly?**

### Snippet 7: `UninitializedWhileLoop`

```java

public class UninitializedWhileLoop {

public static void main(String[] args) {

int count;

while (count < 10) {

System.out.println(count);

count++;

}

}

}

```

\*\*Error:\*\*

- The code produces a compilation error because the variable `count` is used in the `while` loop condition and within the loop body without being initialized. In Java, local variables must be initialized before they are used.

\*\*Why Does This Happen?\*\*

- The variable `count` is declared but not assigned a value before being used in the `while` loop. Accessing an uninitialized variable leads to a compilation error because the compiler cannot determine what value `count` should hold.

\*\*Correction:\*\*

- Initialize the variable `count` before using it in the `while` loop. For example, if you want to start counting from `0`, you should initialize `count` to `0`:

```java

public class UninitializedWhileLoop {

public static void main(String[] args) {

int count = 0;

while (count < 10) {

System.out.println(count);

count++;

}

}

}

```

- This correction will allow the loop to execute properly, printing numbers from `0` to `9`.

\*\*Summary:\*\*

- The compilation error occurs because the `count` variable is uninitialized. Initializing `count` (e.g., `int count = 0;`) before the loop resolves this issue and allows the loop to run as intended.

================================================================================

**Snippet 8:**

**public class OffByOneDoWhileLoop {**

**public static void main(String[] args) {**

**int num = 1;**

**do {**

**System.out.println(num);**

**num--;**

**} while (num > 0);**

**}**

**}**

**// Error to investigate: Why does this loop print unexpected numbers? What adjustments are needed to print the**

**numbers from 1 to 5?**

### Snippet 8: `OffByOneDoWhileLoop`

```java

public class OffByOneDoWhileLoop {

public static void main(String[] args) {

int num = 1;

do {

System.out.println(num);

num--;

} while (num > 0);

}

}

```

\*\*Error:\*\*

- The loop currently prints only the number `1` and then exits. This happens because the loop condition `num > 0` becomes `false` after the first iteration when `num` is decremented to `0`.

\*\*Why Does This Happen?\*\*

- The loop starts with `num` initialized to `1`. During the first iteration, `1` is printed, and then `num` is decremented to `0`. On the next check of the condition `num > 0`, the condition is `false`, so the loop exits. The loop therefore runs only once, printing `1`.

\*\*Correction:\*\*

- To print the numbers from `1` to `5`, the logic needs to be adjusted to increment `num` in each iteration instead of decrementing it, and the loop should continue as long as `num` is less than or equal to `5`.

Here's the corrected code:

```java

public class OffByOneDoWhileLoop {

public static void main(String[] args) {

int num = 1;

do {

System.out.println(num);

num++;

} while (num <= 5);

}

}

```

\*\*Explanation:\*\*

- \*\*Initialization:\*\* `num` starts at `1`.

- \*\*Loop Condition:\*\* The loop runs as long as `num` is less than or equal to `5`.

- \*\*Update Statement:\*\* Increments `num` after each iteration, ensuring the loop progresses toward termination.

\*\*Summary:\*\*

- The original loop only prints `1` because the condition causes an early exit. The correct approach to print numbers from `1` to `5` is to increment `num` within the loop and adjust the condition to `num <= 5`. This adjustment will print the numbers `1` through `5`.

================================================================================

**Snippet 9:**

**public class InfiniteForLoopUpdate {**

**public static void main(String[] args) {**

**for (int i = 0; i < 5; i += 2) {**

**System.out.println(i);**

**}**

**}**

**}**

**// Error to investigate: Why does the loop print unexpected results or run infinitely? How should the loop update**

**expression be corrected?**

### Snippet 9: `InfiniteForLoopUpdate`

```java

public class InfiniteForLoopUpdate {

public static void main(String[] args) {

for (int i = 0; i < 5; i += 2) {

System.out.println(i);

}

}

}

```

\*\*Error:\*\*

- The loop does not run infinitely, but it prints unexpected results. The reason for this behavior is that the update expression `i += 2` increments `i` by `2` each time, which causes `i` to take values of `0`, `2`, and then `4`. When `i` becomes `6` (after the last increment), it does not satisfy the loop condition `i < 5`, so the loop stops.

\*\*Why Does This Happen?\*\*

- With `i` starting at `0` and incrementing by `2`, the values of `i` will be `0`, `2`, and `4`. These values are printed, but the loop does not include `5`, which is the boundary condition.

\*\*Correction:\*\*

- If the intention is to include numbers up to and including `4` (printing numbers `0`, `2`, and `4`), the loop is functioning as expected. However, if the intention is to include numbers up to `5` or change the increments, you should adjust the update expression or loop condition accordingly.

For example, to include values from `0` to `5` (or print up to `5`), you can modify the condition or update expression:

```java

// Option 1: Include `5` if you want to print `0`, `2`, `4`

for (int i = 0; i <= 5; i += 2) {

System.out.println(i);

}

// Option 2: Modify increment if you want to handle different steps

for (int i = 0; i < 6; i++) { // Increment by 1 each step

System.out.println(i);

}

```

\*\*Summary:\*\*

- The loop does not run infinitely but stops when the condition `i < 5` is no longer satisfied. If the goal is to print numbers with a specific step or range, adjust the loop's increment or condition based on the desired output.

================================================================================

**Snippet 10:**

**public class IncorrectWhileLoopControl {**

**public static void main(String[] args) {**

**int num = 10;**

**while (num = 10) {**

**System.out.println(num);**

**num--;**

**}**

**}**

**}**

**// Error to investigate: Why does the loop execute indefinitely? What is wrong with the loop condit**

### Snippet 10: `IncorrectWhileLoopControl`

```java

public class IncorrectWhileLoopControl {

public static void main(String[] args) {

int num = 10;

while (num = 10) {

System.out.println(num);

num--;

}

}

}

```

\*\*Error:\*\*

- The loop executes indefinitely because the condition `num = 10` is an assignment, not a comparison. This code assigns `10` to `num`, and since the assignment expression evaluates to `10` (which is treated as `true` in a boolean context), the loop condition is always `true`. This results in an infinite loop.

\*\*Why Does This Happen?\*\*

- In Java, the assignment operator `=` is used to assign a value to a variable, while the equality operator `==` is used to compare values. The assignment operator returns the assigned value, which in this case is `10`, making the condition always `true`.

\*\*Correction:\*\*

- Use the equality operator `==` to compare `num` with `10` if you intend to check if `num` equals `10`:

```java

while (num == 10) {

System.out.println(num);

num--;

}

```

However, this loop will still execute indefinitely because `num` is initially `10` and decreases by `1` on each iteration, which means it will eventually be less than `10`.

- To properly handle this and avoid an infinite loop, you should modify the condition to make it exit based on the decrementing value:

```java

while (num > 0) {

System.out.println(num);

num--;

}

```

This will print numbers from `10` down to `1` and then stop when `num` becomes `0`.

\*\*Summary:\*\*

- The loop executes indefinitely because the condition `num = 10` is an assignment, not a comparison. Use `num == 10` for comparison or adjust the condition to exit the loop properly.

================================================================================

**Snippet 11:**

**public class IncorrectLoopUpdate {**

**public static void main(String[] args) {**

**int i = 0;**

**while (i < 5) {**

**System.out.println(i);**

**i += 2; // Error: This may cause unexpected results in output**

**}**

**}**

**}**

**// Error to investigate: What will be the output of this loop? How should the loop variable be updated to achieve the**

**desired result?**

### Snippet 11: `IncorrectLoopUpdate`

```java

public class IncorrectLoopUpdate {

public static void main(String[] args) {

int i = 0;

while (i < 5) {

System.out.println(i);

i += 2; // Error: This may cause unexpected results in output

}

}

}

```

\*\*Error:\*\*

- The loop prints values starting from `0` and increments `i` by `2` each time. The loop condition `i < 5` causes the loop to stop when `i` reaches or exceeds `5`.

\*\*Output:\*\*

- The loop will produce the following output:

```

0

2

4

```

Here's why:

- \*\*Iteration 1:\*\* `i = 0`, `System.out.println(i)` prints `0`, then `i` is incremented to `2`.

- \*\*Iteration 2:\*\* `i = 2`, `System.out.println(i)` prints `2`, then `i` is incremented to `4`.

- \*\*Iteration 3:\*\* `i = 4`, `System.out.println(i)` prints `4`, then `i` is incremented to `6`.

- The loop condition `i < 5` fails when `i` is `6`, so the loop exits.

\*\*Correction:\*\*

- If the desired output is to print numbers from `0` to `4` (inclusive) in increments of `1`, you should update the loop increment and condition to handle a different stepping:

```java

int i = 0;

while (i < 5) {

System.out.println(i);

i++; // Increment by 1 to cover each number from 0 to 4

}

```

This correction will print:

```

0

1

2

3

4

```

\*\*Summary:\*\*

- The loop increments `i` by `2`, causing it to print every second number starting from `0` and stopping before `5`. If you want to print all numbers from `0` to `4`, increment `i` by `1` instead. Adjusting the update statement to `i++` will achieve this.

================================================================================

**Snippet 12:**

**public class LoopVariableScope {**

**public static void main(String[] args) {**

**for (int i = 0; i < 5; i++) {**

**int x = i \* 2;**

**}**

**System.out.println(x); // Error: 'x' is not accessible here**

**}**

**}**

**// Error to investigate: Why does the variable 'x' cause a compilation error? How does scope**

### Snippet 12: `LoopVariableScope`

```java

public class LoopVariableScope {

public static void main(String[] args) {

for (int i = 0; i < 5; i++) {

int x = i \* 2;

}

System.out.println(x); // Error: 'x' is not accessible here

}

}

```

\*\*Error:\*\*

- The variable `x` causes a compilation error because it is declared inside the `for` loop's block, which limits its scope to within that loop. Consequently, it is not accessible outside of the loop, leading to the compilation error on `System.out.println(x);`.

\*\*Why Does This Happen?\*\*

- In Java, variables declared within a block (such as the block of a loop) are only accessible within that block. This is known as variable scope. The variable `x` is declared inside the `for` loop, so its scope is limited to the loop body. Once the loop completes, `x` is out of scope and cannot be accessed outside of the loop.

\*\*Correction:\*\*

- To make `x` accessible outside of the loop, declare `x` outside the loop and then assign a value to it within the loop:

```java

public class LoopVariableScope {

public static void main(String[] args) {

int x = 0; // Declare x outside the loop

for (int i = 0; i < 5; i++) {

x = i \* 2; // Update x inside the loop

}

System.out.println(x); // Now x is accessible here

}

}

```

This code will output `8` because `x` is updated during each iteration of the loop, and by the end of the loop, it holds the value `8` (when `i` is `4`).

\*\*Summary:\*\*

- The compilation error occurs because `x` is declared inside the `for` loop and is thus only visible within that loop's block. To access `x` outside the loop, declare it before the loop and then update it within the loop.

================================================================================