



Dhirubhai Ambani Institute of Information and Communication Technology

Gandhinagar, Gujarat

Name : Ghorl Zeel Jlvrajbhail

Student ID : 202201287

Course : IT 314 Software Engineering

Professor : Saurabh Tiwari

Semester : Autumn 2024

Lab : Program Inspection, Debugging and Static Analysis

Tower of Hanoi

Given code :

```
//Tower of Hanoi

public class MainClass {

    public static void main(String[] args) {

        int nDisks = 3;

        doTowers(nDisks, 'A', 'B', 'C');

    }

    public static void doTowers(int topN, char from,
char inter, char to) {

        if (topN == 1){

            System.out.println("Disk 1 from "
+ from + " to " + to);

        }else {

            doTowers(topN - 1, from, to, inter);

            System.out.println("Disk "
+ topN + " from " + from + " to " + to);

            doTowers(topN ++, inter--, from+1, to+1)

        }

    }

}
```

Output: Disk 1 from A to C

Disk 2 from A to B

Disk 1 from C to B

Disk 3 from A to C

Disk 1 from B to A

Disk 2 from B to C

Disk 1 from A to C

Program Inspection of Tower of Hanoi Code Fragment:

1. **How many errors are there in the program? Mention the errors you have identified.**

Errors identified: 3

- **Error 1: Incorrect increment operation (topN++)**
 - **Category:** Computation Error
 - The topN++ increment inside the recursive call is incorrect because ++ performs a post-increment, meaning it will increment topN after using it in the recursive call, which is not intended here. Instead, it should just be topN - 1 like the first recursive call.
- **Error 2: Improper decrement (inter--) and increment (from+1, to+1) on char variables**
 - **Category:** Data Reference Error
 - Decrementing and incrementing characters in this context (i.e., inter--, from+1, to+1) is logically incorrect. These characters represent the tower labels ('A', 'B', 'C'), and arithmetic operations on them would yield unintended results.
- **Error 3: Missing semicolon after the second recursive call**
 - **Category:** Syntax Error
 - In the line doTowers(topN ++, inter--, from+1, to+1), the statement is missing a semicolon at the end, causing a compilation error.

2. **Which category of program inspection would you find more effective?**

Most effective category: Computation and Data Reference Errors

- **Reason:** These categories helped in identifying the incorrect increment/decrement operations, which are key logical errors in this recursive problem. Without correcting these, the algorithm wouldn't perform the Tower of Hanoi logic correctly.

3. **Which type of error you are not able to identify using the program inspection?**

Missed error type: Runtime performance issues and deep recursion limits

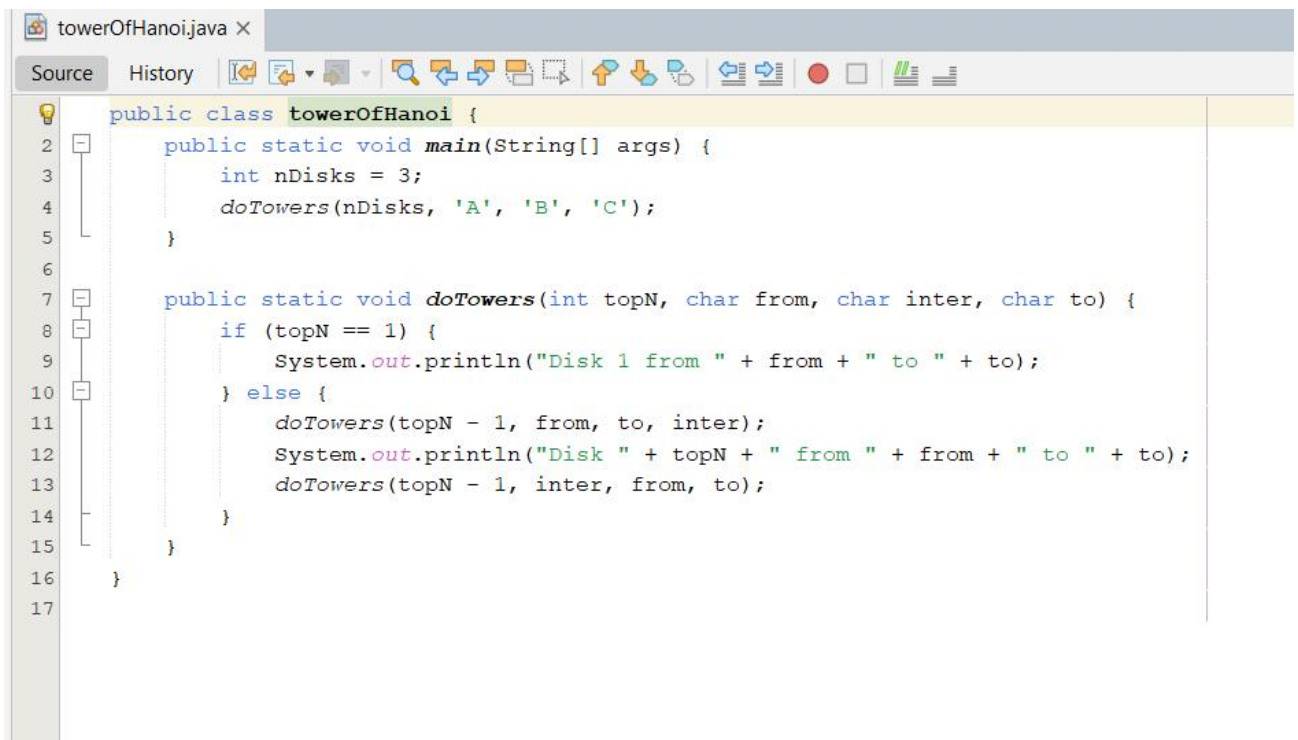
- **Reason:** While this program may run correctly after fixing the identified errors, performance-related issues (such as running out of stack space for deep recursions if the number of disks is large) would not be identifiable through static inspection alone. These would need to be caught through runtime testing.

4. **Is the program inspection technique worth applying?**

Applicability: Yes, the program inspection technique is worth applying.

- **Reason:** Through static inspection, logical and syntactical errors were identified early in the recursive logic, saving time on debugging. This technique ensures that common errors are caught before runtime, though it should be paired with dynamic testing for performance and runtime behavior.

Code Debugging :



```
1 public class towerOfHanoi {
2     public static void main(String[] args) {
3         int nDisks = 3;
4         doTowers(nDisks, 'A', 'B', 'C');
5     }
6
7     public static void doTowers(int topN, char from, char inter, char to) {
8         if (topN == 1) {
9             System.out.println("Disk 1 from " + from + " to " + to);
10        } else {
11            doTowers(topN - 1, from, to, inter);
12            System.out.println("Disk " + topN + " from " + from + " to " + to);
13            doTowers(topN - 1, inter, from, to);
14        }
15    }
16 }
17
```

Identified Errors

1. Incorrect Arithmetic Operations:

- In the line `doTowers(topN ++, inter--, from+1, to+1)`, the use of post-increment (`topN++`) and post-decrement (`inter--`) is not needed and incorrect.
- Additionally, modifying the character variables (`from + 1` and `to + 1`) will convert them to integers, which is not appropriate for this context.

Corrections Made

1. Recursive Calls:

- Changed the recursion call from `doTowers(topN ++, inter--, from+1, to+1)` to simply `doTowers(topN - 1, inter, from, to)`. This keeps the correct values for the characters and uses the right logic for the recursive calls.

Debugging Submission

1. Errors Identified:

- Total of 1 major issue related to incorrect arithmetic operations in the recursive calls.

2. Breakpoints Needed:

- Set breakpoints at the beginning of the doTowers method to examine how the topN, from, inter, and to variables change during recursion.

3. Steps Taken to Fix Errors:

- Removed unnecessary post-increment and post-decrement in the recursive calls.
- Passed the character variables correctly without modification.

4. Complete Executable Code:

- The fixed code provided above can be submitted as your executable code.

Fixed Code

```
// Tower of Hanoi
```

```
public class MainClass {  
    public static void main(String[] args) {  
        int nDisks = 3;  
        doTowers(nDisks, 'A', 'B', 'C');  
    }  
  
    public static void doTowers(int topN, char from, char inter, char to) {  
        if (topN == 1) {  
            System.out.println("Disk 1 from " + from + " to " + to);  
        } else {  
            // Recursive call to move (n-1) disks from 'from' to 'inter' via 'to'  
            doTowers(topN - 1, from, to, inter);  
            // Move the nth disk  
            System.out.println("Disk " + topN + " from " + from + " to " + to);  
            // Recursive call to move (n-1) disks from 'inter' to 'to' via 'from'  
            doTowers(topN - 1, inter, from, to);  
        }  
    }  
}
```

```
        doTowers(topN - 1, inter, from, to);  
    }  
}  
}
```

Output

Disk 1 from A to C
Disk 2 from A to B
Disk 1 from C to B
Disk 3 from A to C
Disk 1 from B to A
Disk 2 from B to C
Disk 1 from A to C

Static Analysis Tools

Choose a static analysis tool (in Java, Python, C, C++) in any programming language of your interest and identify the defects. You can also choose your own code fragment from GitHub (more than 2000 LOC) in any programming language to perform static analysis. Submit your results in the .xls or .jpg format only.

Link of the excel file(.xls):

https://docs.google.com/spreadsheets/d/1ETFxQdw1_RNXufBPwhlt7OumxA7IJCjh/edit?usp=sharing&ouid=117105520491645507929&rtpof=true&sd=true