Submitted By - **Rithika**

Graph Theory in programs?

Program logic -> Control Flow Graph = graph where node are block of code or decision points , edges are possible flow paths

Cyclomatic Complexity (McCabe's Formula)?

How complex your code logic is, by counting the number of independent paths through the program

Cyclomatic Complexity (CC) = E – N + 2

E-> Edges, N->Nodes

Example: Passing level of a student

If marks >= 90 → Grade A

If marks >= 75 → Grade B

Else Grade C

Each **if** creates an independent path

marks >= 90 → A

marks >= 75 → B

else → C

CC = number of decisions + 1 = 2 + 1 = 3

Cognitive Complexity?

Measures how difficult code is for a human to understand

Real-World ex:

If door is open	If door not open → exit
If card is valid	If card invalid → exit
If PIN is correct	If PIN wrong → exit
print "Access Granted"	print "Access Granted"

High Cognitive Complexity:

- Nested conditions

Example:

Independent Paths

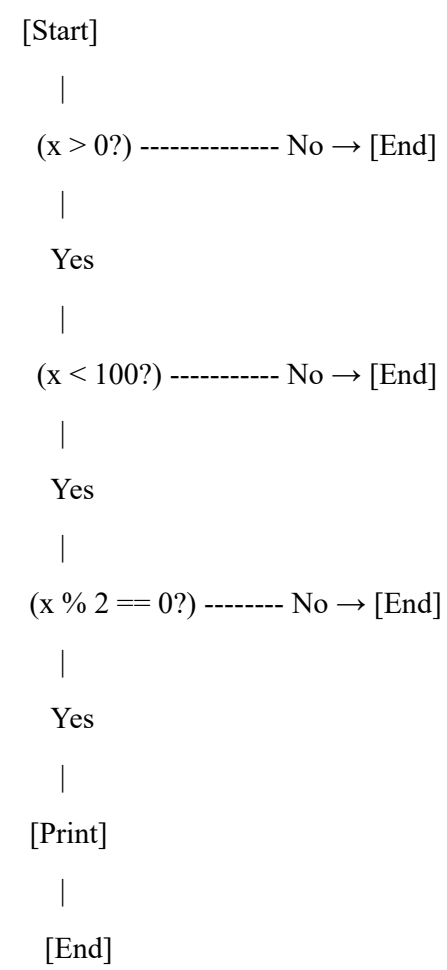
1. condition false at x>0

2. true, then fail at x<100
3. true, true, fail mod2 check
4. all true → print

Start → if(x>0) → if(x<100) → if(x%2==0) → print → End

CC = Decision points + 1 = 3 + 1 = 4

Using graph:



Cyclomatic Complexity (CC) = E – N + 2

= 8 – 6 + 2

= 4

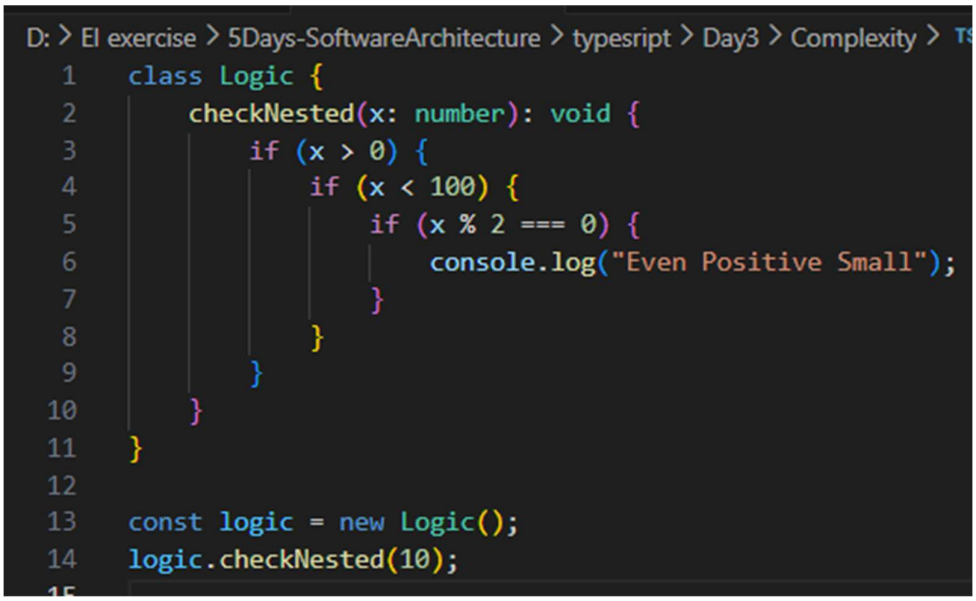
In java:

```
> EI exercise > 5Days-SoftwareArchitecture > Java > Day3 > Complexity > J Bad_nested.jav
1   class Logic {
2       // POOR readability due to deep nesting
3       // Cyclomatic Complexity = 4
4       public void checkNested(int x) {
5           if (x > 0) {
6               if (x < 100) {
7                   if (x % 2 == 0) {
8                       System.out.println("Even Positive Small");
9                   }
10              }
11          }
12      }
13  }
14  class Bad_nested {
15      public static void main(String[] args)
16      {
17          var logic = new Logic();
18          logic.checkNested(10);
19
20      }
21  }
```

```
D:\EI exercise\5Days-SoftwareArchitecture\Java\Day3\Complexity>javac Bad_nes
ted.java

D:\EI exercise\5Days-SoftwareArchitecture\Java\Day3\Complexity>java Bad_nest
ed
Even Positive Small
```

In typescript:



```
D: > EI exercise > 5Days-SoftwareArchitecture > typesript > Day3 > Complexity > TS
1   class Logic {
2       checkNested(x: number): void {
3           if (x > 0) {
4               if (x < 100) {
5                   if (x % 2 === 0) {
6                       console.log("Even Positive Small");
7                   }
8               }
9           }
10      }
11  }
12
13  const logic = new Logic();
14  logic.checkNested(10);
```

```
D:\EI exercise\5Days-SoftwareArchitecture\typesript\Day3\Complexity>tsc bad_
nested.ts

D:\EI exercise\5Days-SoftwareArchitecture\typesript\Day3\Complexity>node bad
_nested
Even Positive Small
```

Low Cognitive complexity:

- Flat Structure

Using Graph:

Start

|

[x <= 0?] ---Yes---> End

|

No

|

[x >= 100?] ---Yes---> End

|

No

|

[x % 2 != 0?] ---Yes---> End

|

No

|

Print

|

End

CC = 3 decisions + 1 = 4

In java:

D: > EI exercise > 5Days-SoftwareArchitecture > Java > Day3 > Complexity >

```java
class Logic {
    // Flat structure improves readability
    // Cyclomatic Complexity = still 4 (same logic, sa
    public void checkFlat(int x) {
        if (x <= 0) return;
        if (x >= 100) return;
        if (x % 2 != 0) return;

        System.out.println("Even Positive Small");
    }
}

public class low_comp {
    public static void main(String[] args) {
        Logic logic = new Logic();
        logic.checkFlat(10);
    }
}
```

```
D:\EI exercise\5Days-SoftwareArchitecture\Java\Day3\Complexity>javac low_com
p.java

D:\EI exercise\5Days-SoftwareArchitecture\Java\Day3\Complexity>java low_comp

Even Positive Small
```

In Typescript:

D: > EI exercise > 5Days-SoftwareArchitecture > typesript > Day3 > Complex

```typescript
class Logic {
    checkFlat(x: number): void {
        if (x <= 0) return;
        if (x >= 100) return;
        if (x % 2 !== 0) return;

        console.log("Even Positive Small");
    }
}

const logic = new Logic();
logic.checkFlat(10);
```

```
D:\EI exercise\5Days-SoftwareArchitecture\typesript\Day3\Complexity>tsc high
_comp.ts

D:\EI exercise\5Days-SoftwareArchitecture\typesript\Day3\Complexity>node hig
h_comp
Even Positive Small
```

My Scenario **: Bank Management**

CC = E – N + 2

E = number of edges (transitions)

N = number of nodes (blocks)

(or)

CC = Number of decision points + 1

(decision - if, else if, while, for, switch-case, catch, &&, ||)

High cognitive complexity

Using graph representation:

Start

|

v

[if balance > 0] ----No----> END

|

Yes

|

[if amount <= balance] ----No----> END

|

Yes

|

[if otpVerified] ----No----> END

|

Yes

|

[if !isHoliday] ----No----> END

|

Yes

|

[switch]

/ \

SAV CUR DEF

| | |

Msg Msg Default

| | |

END

CC = E – N + 2 = 15 – 10 + 2 = 7

Element	Count
if	4
cases	2 additional paths

CC = 6 + 1 = 7

IN JAVA:

D: > EI exercise > 5Days-SoftwareArchitecture > Java > Day3 > Complexity > J bad_nested_bank.java

```java
class BankService {

    public void withdraw(double balance, double amount, boolean otpVerified, boolean isHoliday, String accountType) {

        if (balance > 0) {                              // 1
            if (amount <= balance) {                    // 2
                if (otpVerified) {                      // 3
                    if (!isHoliday) {                   // 4
                        switch (accountType) {          // 5
                            case "SAVINGS":
                                System.out.println("Withdraw Successful from Savings");
                                break;
                            case "CURRENT":
                                System.out.println("Withdraw Successful from Current");
                                break;
                            default:
                                System.out.println("Invalid Account");
                                break;
                        }
                    } else {
                        System.out.println("Cannot withdraw: Bank is closed today (holiday).");
                    }
                } else {
                    System.out.println("Cannot withdraw: OTP not verified.");
                }
            } else {
                System.out.println("Cannot withdraw: Insufficient balance.");
            }
        } else {
            System.out.println("Cannot withdraw: Account balance must be positive.");
        }
    }
}
```

```java
public class bad_nested_bank {
    public static void main(String[] args) {
        BankService svc = new BankService();

        System.out.println("=== Test 1: Successful withdraw (SAVINGS) ===");
        svc.withdraw(1000.0, 100.0, true, false, "SAVINGS");

        System.out.println("\n=== Test 2: Successful withdraw (CURRENT) ===");
        svc.withdraw(500.0, 200.0, true, false, "CURRENT");

    }
}
```

```
D:\EI exercise\5Days-SoftwareArchitecture\Java\Day3\Complexity>javac bad_nes
ted_bank.java

D:\EI exercise\5Days-SoftwareArchitecture\Java\Day3\Complexity>java bad_nest
ed_bank
=== Test 1: Successful withdraw (SAVINGS) ===
Withdraw Successful from Savings

=== Test 2: Successful withdraw (CURRENT) ===
Withdraw Successful from Current
```

IN TYPESCRIPT:

EI exercise > 5Days-SoftwareArchitecture > typesript > Day3 > Complexity > TS bad_nested_bank.ts >

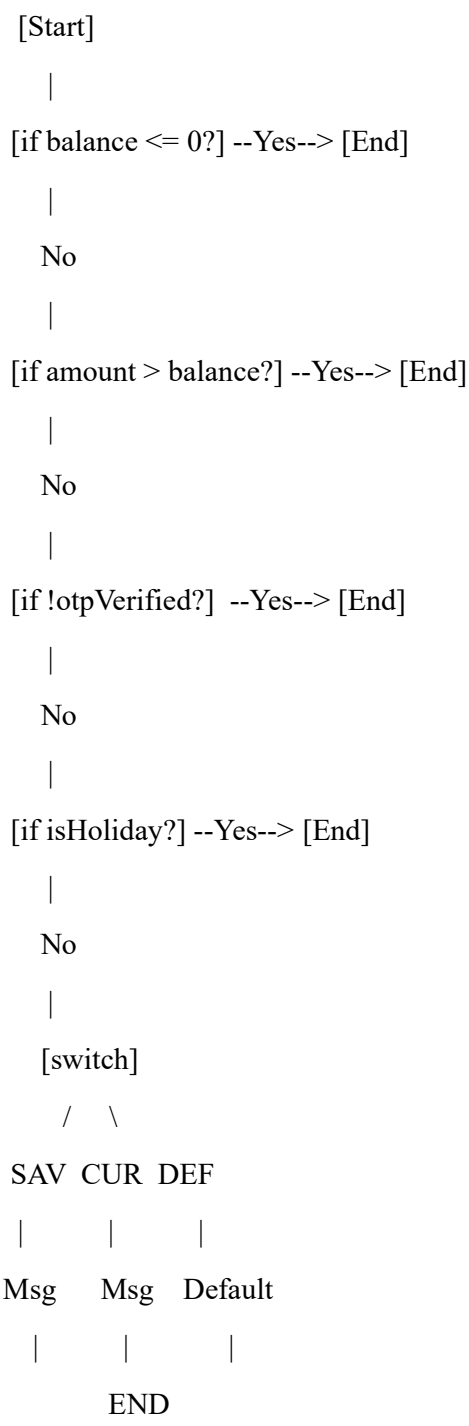
```typescript
class BankServiceNested {
  withdraw(
    balance: number,
    amount: number,
    otpVerified: boolean,
    isHoliday: boolean,
    accountType: string
  ): void {
    if (balance > 0) {                          // decision 1
      if (amount <= balance) {                  // decision 2
        if (otpVerified) {                      // decision 3
          if (!isHoliday) {                     // decision 4
            switch (accountType) {              // decision 5 (switch)
              case "SAVINGS":                   // case path 1
                console.log("Withdraw Successful from Savings");
                break;
              case "CURRENT":                   // case path 2
                console.log("Withdraw Successful from Current");
                break;
              default:
                console.log("Invalid Account");
                break;
            }
          }
        }
      }
    }
  }

const svc = new BankServiceNested();
svc.withdraw(1000, 100, true, false, "SAVINGS");
```

```
D:\EI exercise\5Days-SoftwareArchitecture\typesript\Day3\Complexity>tsc bad_
nested_bank.ts

D:\EI exercise\5Days-SoftwareArchitecture\typesript\Day3\Complexity>node bad
_nested_bank
Withdraw Successful from Savings
```

Low cognitive Complexit : Using graph Representation



Decision	Count
if	4
cases	2

CC=6 +1= 7

IN JAVA:

D: > EI exercise > 5Days-SoftwareArchitecture > Java > Day3 > Complexity > flat_st_bank.java

```java
class BankServiceOptimized {

    public void withdraw(double balance, double amount, boolean otpVerified, boolean isHoliday, String accountType) {

        if (balance <= 0) {
            System.out.println("Cannot withdraw: Balance must be positive.");
            return;
        }

        if (amount > balance) {
            System.out.println("Cannot withdraw: Insufficient balance.");
            return;
        }

        if (!otpVerified) {
            System.out.println("Cannot withdraw: OTP not verified.");
            return;
        }

        if (isHoliday) {
            System.out.println("Cannot withdraw: Bank is closed (holiday).");
            return;
        }

        switch (accountType) {
            case "SAVINGS" -> System.out.println("Withdraw Successful from Savings");
            case "CURRENT" -> System.out.println("Withdraw Successful from Current");
            default -> System.out.println("Invalid Account Type");
        }
    }
}
```

```java
public class flat_st_bank {

    public static void main(String[] args) {

        BankServiceOptimized svc = new BankServiceOptimized();

        System.out.println("=== Test 1: Successful withdraw (SAVINGS) ===");
        svc.withdraw(1000.0, 200.0, true, false, "SAVINGS");

        System.out.println("\n=== Test 2: Successful withdraw (CURRENT) ===");
        svc.withdraw(800.0, 300.0, true, false, "CURRENT");

        System.out.println("\n=== Test 3: Insufficient balance ===");
        svc.withdraw(200.0, 500.0, true, false, "SAVINGS");

    }

}
```

```
D:\EI exercise\5Days-SoftwareArchitecture\Java\Day3\Complexity>javac flat_st
_bank.java

D:\EI exercise\5Days-SoftwareArchitecture\Java\Day3\Complexity>java flat_st_
bank
=== Test 1: Successful withdraw (SAVINGS) ===
Withdraw Successful from Savings

=== Test 2: Successful withdraw (CURRENT) ===
Withdraw Successful from Current

=== Test 3: Insufficient balance ===
Cannot withdraw: Insufficient balance.
```

IN TYPESCRIPT:

D: > EI exercise > 5Days-SoftwareArchitecture > typesript > Day3 > Complexity > TS flat_st_b

```typescript

class BankServiceFlat {
  withdraw(
    balance: number,
    amount: number,
    otpVerified: boolean,
    isHoliday: boolean,
    accountType: string
  ): void {
    //  flat and easy to read
    if (balance <= 0) return;
    if (amount > balance) return;
    if (!otpVerified) return;
    if (isHoliday) return;

    switch (accountType) {
      case "SAVINGS":
        console.log("Withdraw Successful from Savings");
        break;
      case "CURRENT":
        console.log("Withdraw Successful from Current");
        break;
      default:
        console.log("Invalid Account");
    }
  }
}


const svcFlat = new BankServiceFlat();
svcFlat.withdraw(1000, 100, true, false, "CURRENT");

```

```
D:\EI exercise\5Days-SoftwareArchitecture\typesript\Day3\Complexity>tsc flat
_st_bank.ts

D:\EI exercise\5Days-SoftwareArchitecture\typesript\Day3\Complexity>node fla
t_st_bank
Withdraw Successful from Current
```

CC same but CoC reduced?

- ✓ Because CC counts logical branches, not structure.
- ✓ But Cognitive Complexity counts human effort, so structure matters

CC vs CogC:

Feature	Cyclomatic Complexity (CC)	Cognitive Complexity (CogC)
Measures	Number of independent paths	Readability & nesting
Ignores	Nesting & control flow	Does **not** ignore these
Goal	Testing coverage	Maintainability & understandability
Example	if-else, switch, loops	Penalizes nested if, for, recursion
Interpretation	Higher CC → more tests	Higher CogC → harder to read/maintain

White-Box Testing?:

a software testing technique where the tester looks inside the code and tests the internal logic, structure, and paths of the program.

Common White-Box Testing Techniques:

1.) Statement coverage – Test every line of code atleast one
   - Like if , code have 10 lines -> writing testcases that execute all 10 lines
2.) Branch/Decision Coverage – Test every decision (T/F) of each conditions
   - If (amount >1000) then test with amount= 13000(t) , amount=200(False)
3.) Loop Testing – Test the loops in the code witll its iteration (0,1,…n)
   - for(int r=0,r>n;r++) then test n=0,n>1

Basis Path Testing/Path coverage:

Design test cases to execute every independent path in a program at least once.

Example: Bank Management

Decisions:

1. if (balance <= 0)

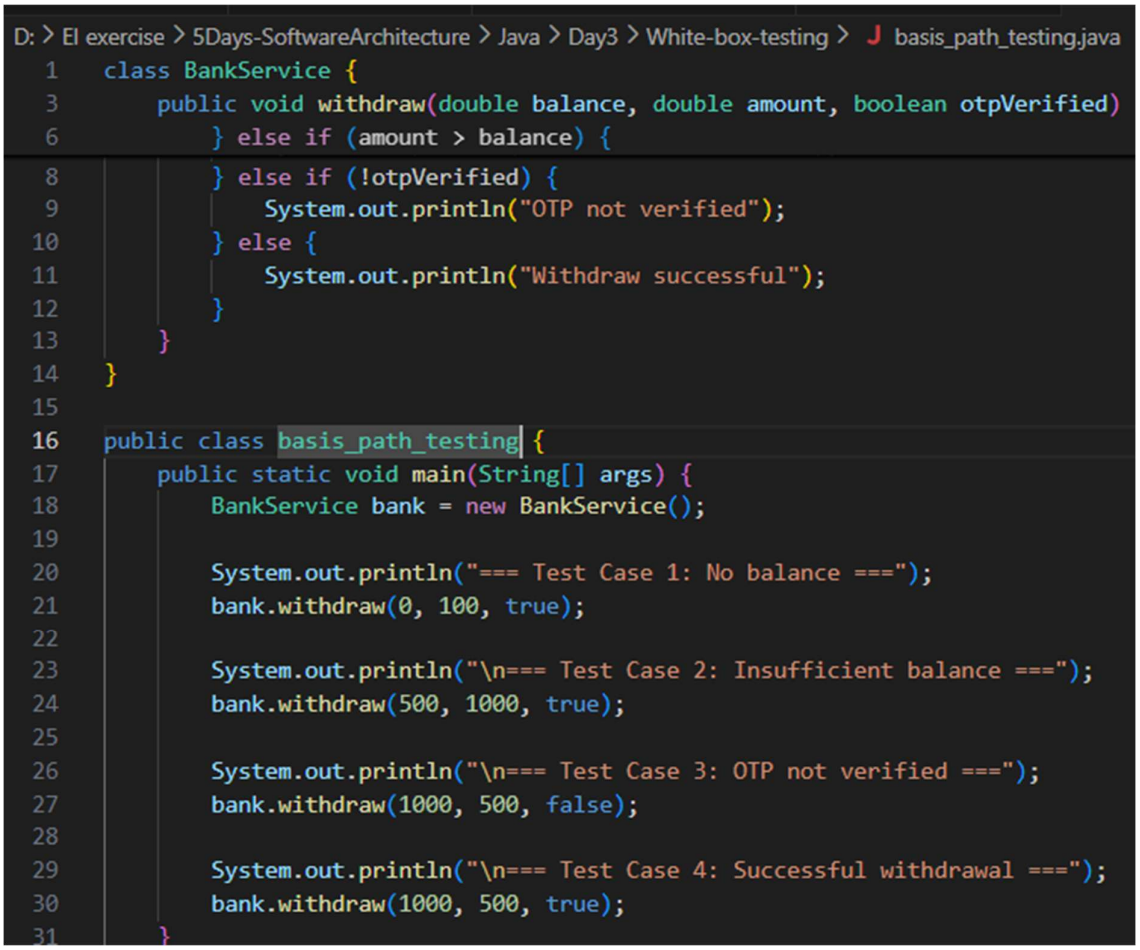
2. else if (amount > balance)
3. else if (!otpVerified)

CC= decisions +1= 3+1=4 , so 4 paths

Independent paths (Basis Paths):

1. Balance ≤ 0 → "No balance"
2. Balance > 0 & amount > balance → "Insufficient balance"
3. Balance > 0 & amount ≤ balance & OTP not verified → "OTP not verified"
4. Balance > 0 & amount ≤ balance & OTP verified → "Withdraw successful

In java:



```
D:\EI exercise\5Days-SoftwareArchitecture\Java\Day3\White-box-testing>javac
basis_path_testing.java

D:\EI exercise\5Days-SoftwareArchitecture\Java\Day3\White-box-testing>java b
asis_path_testing
=== Test Case 1: No balance ===
No balance

=== Test Case 2: Insufficient balance ===
Insufficient balance

=== Test Case 3: OTP not verified ===
OTP not verified

=== Test Case 4: Successful withdrawal ===
Withdraw successful
```

In TypeScript:

D: > EI exercise > 5Days-SoftwareArchitecture > typesript > Day3 > white-box-testing > TS basis_path_testing.

```typescript
class BankService {
    withdraw(balance: number, amount: number, otpVerified: boolean): void {
        if (balance <= 0) {
            console.log("No balance");
        } else if (amount > balance) {
            console.log("Insufficient balance");
        } else if (!otpVerified) {
            console.log("OTP not verified");
        } else {
            console.log("Withdraw successful");
        }
    }
}

// Create an instance
const bank = new BankService();

// Test cases
console.log("=== Test Case 1: No balance ===");
bank.withdraw(0, 100, true);

console.log("\n=== Test Case 2: Insufficient balance ===");
bank.withdraw(500, 1000, true);

console.log("\n=== Test Case 3: OTP not verified ===");
bank.withdraw(1000, 500, false);

console.log("\n=== Test Case 4: Successful withdrawal ===");
bank.withdraw(1000, 500, true);
```

```
D:\EI exercise\5Days-SoftwareArchitecture\typesript\Day3\white-box-testing>t
sc basis_path_testing.ts

D:\EI exercise\5Days-SoftwareArchitecture\typesript\Day3\white-box-testing>n
ode basis_path_testing
=== Test Case 1: No balance ===
No balance

=== Test Case 2: Insufficient balance ===
Insufficient balance

=== Test Case 3: OTP not verified ===
OTP not verified

=== Test Case 4: Successful withdrawal ===
Withdraw successful
```

Path Sensitization?

Ensuring each path user test is actually feasible

An impossible path (also called a "dead path") should not be tested because the conditions cannot happen logically.

Example: Impossible path but CC= 2+1=3 as , (x > 5 AND x < 3) can never happen, so it is **dead logic**

```java
public void checkValue(int x) {
    if (x > 5) {            // Condition 1
        if (x < 3) {        // Condition 2
            System.out.println("Impossible");
        }
    }
}
```

Thus, Path sensitization identify impossible paths like this and don't waste time testing them.

Steps to do Basis Path Testing with Path Sensitization?

Draw the Control Flow Graph (CFG) → Calculate Cyclomatic Complexity (CC) → List all independent paths according to CC → Analyze each path , if not feasible then mark as "Dead logic" → Design Test case for all feasible paths

Example: Bank Management

Start

└─> if (balance <= 0)?

```
    ├─Yes: "No balance" -> End (P1)
    └─No:
       └─> if (amount > balance)?
          ├─Yes: "Insufficient" -> End (P2)
          └─No:
             └─> if (!otpVerified)?
                ├─Yes: "OTP not verified" -> End (P3)
                └─No:
                   └─> if (isHoliday)?
                      ├─Yes: "Bank closed" -> End (P4)
                      └─No: "Withdraw successful" -> End (P5)
Impossible Path (Dead Logic - P6)
 └─> if (balance < 0 AND balance > 1000)?
    ├─Yes: "Impossible Path Reached!" -> End
    └─No: (never happens)
```

Decisions(CC= 5+ 1 =6 , so 6 independent paths)

1.) if (balance <= 0) → decision 1
2.) else if (amount > balance) → decision 2
3.) else if (!otpVerified) → decision 3
4.) else if (isHoliday) → decision 4
5.) Impossible path: if (balance < 0 && balance > 1000) → decision 5

Buy use path sensitization , only 5 Paths are feasible

In java:

```java
class BankService {
    public void withdraw(double balance, double amount, boolean otpVerified, boolean isHoliday, boolean
        // Path P1: No balance
        if (balance <= 0) {
            System.out.println("No balance");       // P1
        }
        // Path P2: Amount exceeds balance
        else if (amount > balance) {
            System.out.println("Insufficient");    // P2
        }
        // Path P3: OTP not verified
        else if (!otpVerified) {
            System.out.println("OTP not verified"); // P3
        }
        // Path P4: Bank closed
        else if (isHoliday) {
            System.out.println("Bank closed");     // P4
        }
        // Path P5: Successful withdrawal
        else {
            System.out.println("Withdraw successful"); // P5
        }
        // Condition: balance < 0 AND balance > 1000 at the same time → cannot happen
        if (balance < 0 && balance > 1000) {
            System.out.println("Impossible Path Reached!"); // P6: DEAD LOGIC
        }
    }
}
```

```java
public class path_sensitization {
    public static void main(String[] args) {
        BankService bank = new BankService();

        System.out.println("=== Test Case 1: No balance ===");
        bank.withdraw(0, 100, true, false, false);  // P1

        System.out.println("\n=== Test Case 2: Insufficient balance ===");
        bank.withdraw(500, 1000, true, false, false); // P2

        System.out.println("\n=== Test Case 3: OTP not verified ===");
        bank.withdraw(1000, 500, false, false, false); // P3

        System.out.println("\n=== Test Case 4: Bank holiday ===");
        bank.withdraw(1000, 500, true, true, false); // P4

        System.out.println("\n=== Test Case 5: Successful withdrawal ===");
        bank.withdraw(1000, 500, true, false, false); // P5

        System.out.println("\n=== Test Case 6: Impossible path (DEAD LOGIC) ===");
        bank.withdraw(-10, 2000, true, false, false); // P6 → Sensitized, impossible
    }
}
```

```
D:\EI exercise\5Days-SoftwareArchitecture\Java\Day3\White-box-testing>javac
path_sensitization.java

D:\EI exercise\5Days-SoftwareArchitecture\Java\Day3\White-box-testing>java p
ath_sensitization
=== Test Case 1: No balance ===
No balance

=== Test Case 2: Insufficient balance ===
Insufficient

=== Test Case 3: OTP not verified ===
OTP not verified

=== Test Case 4: Bank holiday ===
Bank closed

=== Test Case 5: Successful withdrawal ===
Withdraw successful

=== Test Case 6: Impossible path (DEAD LOGIC) ===
No balance
```

In typescript:

D: > EI exercise > 5Days-SoftwareArchitecture > typesript > Day3 > white-box-testing > TS path_sensitization.ts > BankService > withdraw

```typescript
class BankService {
    withdraw(balance: number, amount: number, otpVerified: boolean, isHoliday: boolean, isVIP: boolean):
        // Path P1: No balance
        if (balance <= 0) {
            console.log("No balance");       // P1
        }
        // Path P2: Amount exceeds balance
        else if (amount > balance) {
            console.log("Insufficient");    // P2
        }
        // Path P3: OTP not verified
        else if (!otpVerified) {
            console.log("OTP not verified"); // P3
        }
        // Path P4: Bank closed
        else if (isHoliday) {
            console.log("Bank closed");     // P4
        }
        // Path P5: Successful withdrawal
        else {
            console.log("Withdraw successful"); // P5
        }

        // Impossible path example (dead logic)
        if (balance < 0 && balance > 1000) {
            console.log("Impossible Path Reached!");
        }
    }
}
```

```
// Test the BankService
const bank = new BankService();

console.log("=== Test Case 1: No balance ===");
bank.withdraw(0, 100, true, false, false);  // P1

console.log("\n=== Test Case 2: Insufficient balance ===");
bank.withdraw(500, 1000, true, false, false); // P2

console.log("\n=== Test Case 3: OTP not verified ===");
bank.withdraw(1000, 500, false, false, false); // P3

console.log("\n=== Test Case 4: Bank holiday ===");
bank.withdraw(1000, 500, true, true, false); // P4

console.log("\n=== Test Case 5: Successful withdrawal ===");
bank.withdraw(1000, 500, true, false, false); // P5

console.log("\n=== Test Case 6: Impossible path (DEAD LOGIC) ===");
bank.withdraw(-10, 2000, true, false, false); // P6 ← Sensitized, impossible
```

```
D:\EI exercise\5Days-SoftwareArchitecture\typesript\Day3\white-box-testing>t
sc path_sensitization.ts

D:\EI exercise\5Days-SoftwareArchitecture\typesript\Day3\white-box-testing>n
ode path_sensitization
=== Test Case 1: No balance ===
No balance

=== Test Case 2: Insufficient balance ===
Insufficient

=== Test Case 3: OTP not verified ===
OTP not verified

=== Test Case 4: Bank holiday ===
Bank closed

=== Test Case 5: Successful withdrawal ===
Withdraw successful

=== Test Case 6: Impossible path (DEAD LOGIC) ===
No balance
```

Sonarsource – company that develops tools to analyze and improce code quality

calculated using three main rules:

1. +1 for each break in linear flow – e.g., if, for, while, switch, catch.
2. +1 for nesting – deeper nesting increases the score more rapidly.

3. Ignore shorthand constructs that improve readability, such as ternary operators or early returns.

Measure of CogC:

- Lower scores mean the code is easier to read and maintain.
- High scores indicate the need for refactoring to reduce nesting, split large methods, or simplify branching

Example:

Loop → +1

If → +1

Total Cognitive Complexity = 1 (loop) + 2 (if nested) = 3

In java:

D: > El exercise > 5Days-SoftwareArchitecture > Java > Day3 > Complexity > SimpleCogC.java

```java
import java.util.List;

public class SimpleCogC {

    // Calculate sum of positive numbers
    int sumPositive(List<Integer> numbers) {
        int sum = 0; // CCog = 0 (simple statement)

        for (int num : numbers) { // +1 (loop)
            if (num > 0) {          // +1 (if) +1 (nested inside loop) = 2
                sum += num;         // 0 (simple statement)
            }
        }
        return sum; // 0
    }

    public static void main(String[] args) {
        SimpleCogC example = new SimpleCogC();
        List<Integer> numbers = List.of(-5, 0, 2, 3, -1);
        int result = example.sumPositive(numbers);

        System.out.println("Sum of positive numbers: " + result);
        System.out.println("Cognitive Complexity: Loop(1) + if(2) = 3");
    }
}
```

```
D:\EI exercise\5Days-SoftwareArchitecture\Java\Day3\Complexity>javac SimpleC
ogC.java

D:\EI exercise\5Days-SoftwareArchitecture\Java\Day3\Complexity>java SimpleCo
gC
Sum of positive numbers: 5
Cognitive Complexity: Loop(1) + if(2) = 3
```

In Typescript:

: > EI exercise > 5Days-SoftwareArchitecture > typesript > Day3 > Complexity > TS SimpleC

```typescript
class SimpleCogC {

    sumPositive(numbers: number[]): number {
        let sum = 0; // CCog = 0

        for (let num of numbers) { // +1 (loop)
            if (num > 0) {          // +1 (if) +1 (nested inside
                sum += num;         // 0
            }
        }

        return sum; // 0
    }
}

// Usage
const example = new SimpleCogC();
const numbers = [-5, 0, 2, 3, -1];
const result = example.sumPositive(numbers);

console.log("Sum of positive numbers:", result);
console.log("Cognitive Complexity: Loop(1) + if(2) = 3");

```

```
D:\EI exercise\5Days-SoftwareArchitecture\typesript\Day3\Complexity>tsc Simp
leCogC.ts

D:\EI exercise\5Days-SoftwareArchitecture\typesript\Day3\Complexity>node Sim
pleCogC
Sum of positive numbers: 5
Cognitive Complexity: Loop(1) + if(2) = 3
```