Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Software Testing and Quality Assurance

Name: Varenya Uchil

SAPID: 60004210121

Branch: Computer Engineering C2-2

Experiment No. 3

Aim: White Box Testing on Units/Modules of Income Tax Calculator using JUnit/Selenium

Theory:

White box testing (also known as **clear box**, **glass box**, or **structural testing**) is a software testing method where the **internal structure**, **design**, and **code** of the software are known to the tester. The main goal is to **verify the logic and flow of the code**, rather than just its functional output.

Key Characteristics:

- **Code-based**: The tester requires knowledge of the programming language and source code.
- **Focus**: Internal logic, paths, conditions, loops, and data flow.
- **Tools Used**: JUnit (for Java), NUnit (for .NET), PyTest (for Python), coverage analyzers.

Common Techniques:

1. **Statement Coverage**:
   Ensure every line of code is executed at least once.
2. **Branch/Decision Coverage**:
   Ensure every possible outcome (true/false) of each decision (like if, switch) is tested.
3. **Path Coverage**:
   Test all possible paths through the program logic.
4. **Loop Testing**:
   Focuses on validating loops (zero, one, and multiple iterations).
5. **Condition Coverage**:
   Checks for every individual condition in decision statements (like A && B).

Advantages:

- Efficient at finding **hidden logical errors** or **boundary issues**
- Optimizes the code by identifying **dead code** or **redundant paths**
- Helps in **unit testing**, ensuring individual functions work as intended

Disadvantages:

- Not suitable for **large, complex systems** without high automation
- Requires detailed knowledge of the source code
- Changes in the code might require **rewriting tests**

Tools and IDE used -

1. **Java Development Kit (JDK):**

   This is the core requirement for developing and running Java applications. JUnit relies on the JDK to compile and execute your test cases.

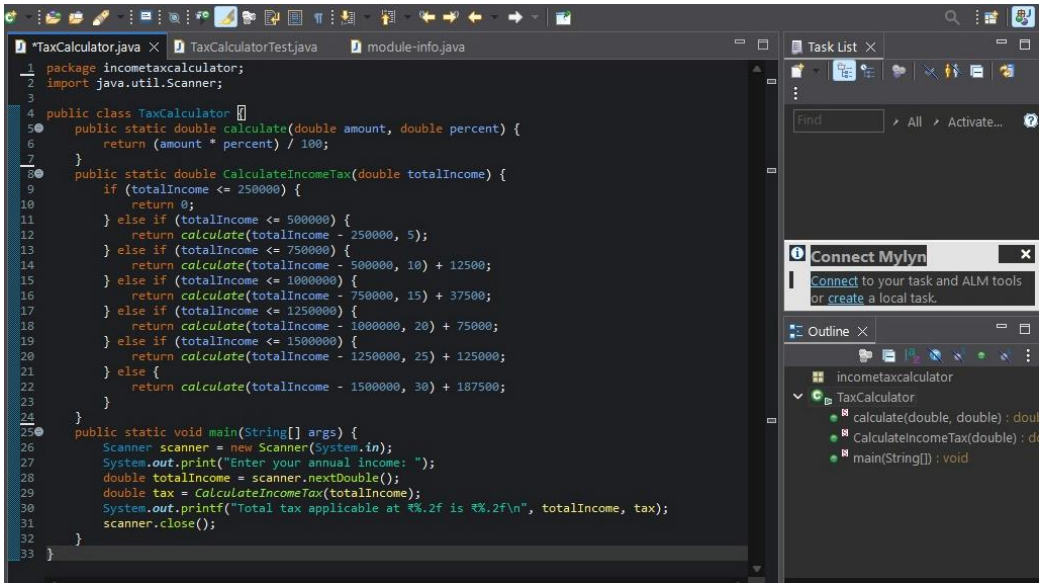
2. **JUnit Framework:**

   This is the testing framework that provides annotations, assertions, and other functionalities for writing and running unit tests.

3. **Eclipse IDE:**

   Eclipse is a popular open-source IDE well-suited for Java development, including JUnit testing.

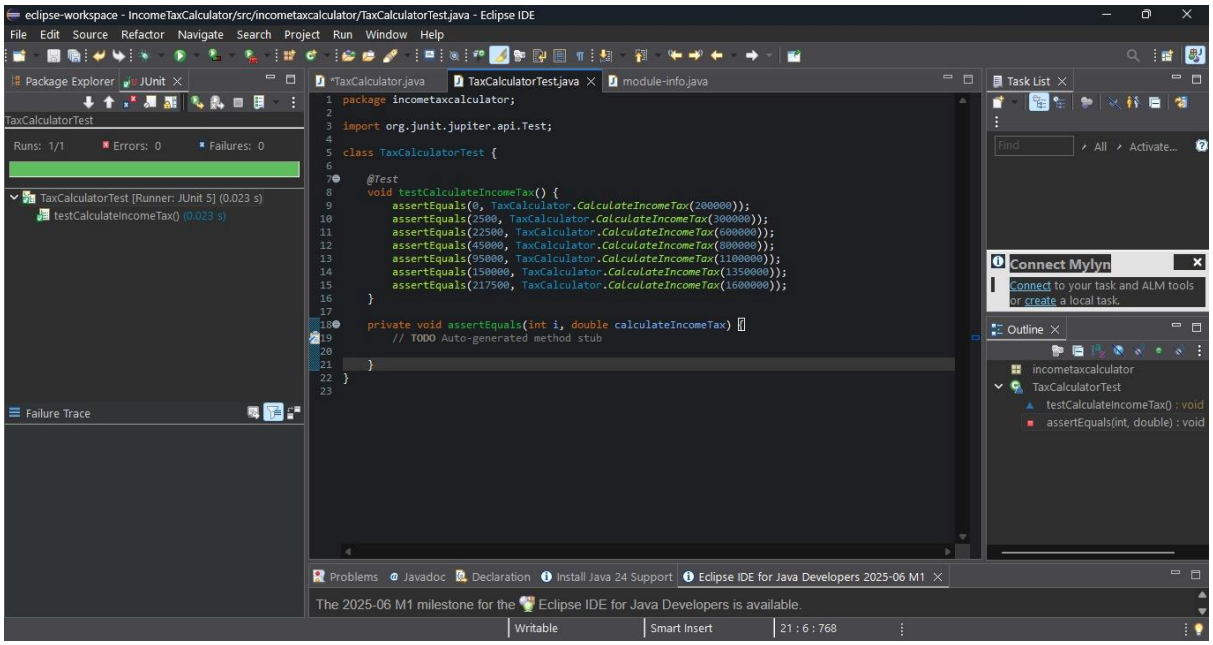
Code to be tested:




```java
package incometaxcalculator;
import java.util.Scanner;

public class TaxCalculator {
    public static double calculate(double amount, double percent) {
        return (amount * percent) / 100;
    }
    public static double CalculateIncomeTax(double totalIncome) {
        if (totalIncome <= 250000) {
            return 0;
        } else if (totalIncome <= 500000) {
            return calculate(totalIncome - 250000, 5);
        } else if (totalIncome <= 750000) {
            return calculate(totalIncome - 500000, 10) + 12500;
        } else if (totalIncome <= 1000000) {
            return calculate(totalIncome - 750000, 15) + 37500;
        } else if (totalIncome <= 1250000) {
            return calculate(totalIncome - 1000000, 20) + 75000;
        } else if (totalIncome <= 1500000) {
            return calculate(totalIncome - 1250000, 25) + 125000;
        } else {
            return calculate(totalIncome - 1500000, 30) + 187500;
        }
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter your annual income: ");
        double totalIncome = scanner.nextDouble();
        double tax = CalculateIncomeTax(totalIncome);
        System.out.printf("Total tax applicable at ₹%.2f is ₹%.2f\n", totalIncome, tax);
        scanner.close();
    }
}
```

Testing Code with Output:



Observations:

1. The provided JUnit test cases cover a range of income levels to verify the correctness of the CalculateIncomeTax function. These tests ensure that the function returns the expected tax amount for different income scenarios
2. The code is organized into separate functions (CalculateIncomeTax and calculate) for better modularity and readability. This separation allows for easier testing and maintenance.

Conclusion:

White-box testing, also known as glass-box testing, examines the internal structure and logic of the code. By employing JUnit, a popular Java unit testing framework, we could create targeted test cases to assess the functionality of individual units and modules within the Income Tax Calculator application.