Experiment No 12

Aim: Develop test cases for the projects using White Box Testing (JUnit)

Theory:

White Box Testing:

White box testing techniques analyze the internal structures the used data structures, internal design, code structure, and the working of the software rather than just the functionality as in black box testing. It is also called glass box testing or clear box testing or structural testing. White Box Testing is also known as transparent testing or open box testing.

White box testing is a software testing technique that involves testing the internal structure and workings of a software application. The tester has access to the source code and uses this knowledge to design test cases that can verify the correctness of the software at the code level.

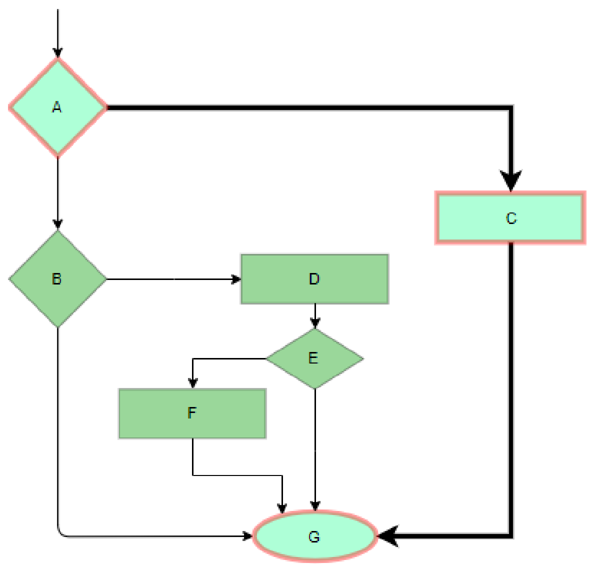
White box testing is also known as structural testing or code-based testing, and it is used to test the software's internal logic, flow, and structure. The tester creates test cases to examine the code paths and logic flows to ensure they meet the specified requirements.

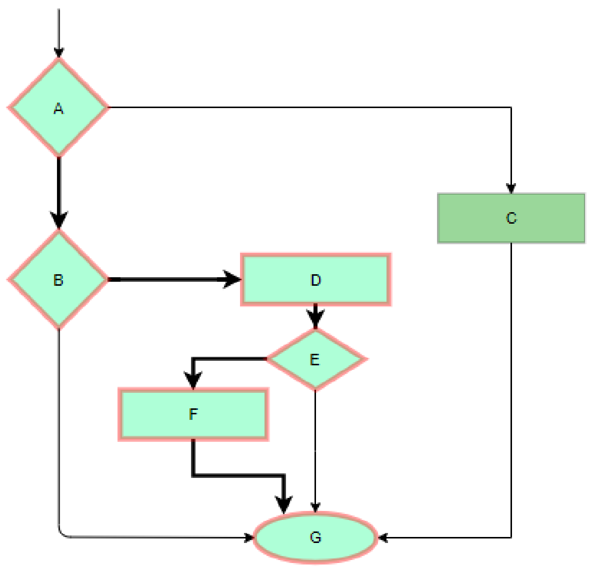
Working process of white box testing:

- Input: Requirements, Functional specifications, design documents, source code.
- Processing: Performing risk analysis to guide through the entire process.
- Proper test planning: Designing test cases so as to cover the entire code. Execute rinse-repeat until error-free software is reached. Also, the results are communicated.
- Output: Preparing final report of the entire testing process.

Testing techniques:

Statement coverage: In this technique, the aim is to traverse all statements at least once. Hence, each line of code is tested. In the case of a flowchart, every node must be traversed at least once. Since all lines of code are covered, helps in pointing out faulty code.





Branch Coverage: In this technique, test cases are designed so that each branch from all decision points is traversed at least once. In a flowchart, all edges must be traversed at least once.

Condition Coverage: In this technique, all individual conditions must be covered as shown in the following example:

1. READ X, Y
2. IF(X == 0 || Y == 0)
3. PRINT ‘0’
4. #TC1 – X = 0, Y = 55
5. #TC2 – X = 5, Y = 0

Multiple Condition Coverage: In this technique, all the possible combinations of the possible outcomes of conditions are tested at least once. Let’s consider the following example:

1. READ X, Y
2. IF(X == 0 || Y == 0)
3. PRINT ‘0’
4. #TC1: X = 0, Y = 0
5. #TC2: X = 0, Y = 5
6. #TC3: X = 55, Y = 0
7. #TC4: X = 55, Y = 5

Basis Path Testing: In this technique, control flow graphs are made from code or flowchart and then Cyclomatic complexity is calculated which defines the number of independent paths so that the minimal number of test cases can be designed for each independent path.

Steps:

1. Make the corresponding control flow graph
2. Calculate the cyclomatic complexity
3. Find the independent paths
4. Design test cases corresponding to each independent path
5. V(G) = P + 1, where P is the number of predicate nodes in the flow graph
6. V(G) = E – N + 2, where E is the number of edges and N is the total number of nodes
7. V(G) = Number of non-overlapping regions in the graph
8. #P1: 1 – 2 – 4 – 7 – 8
9. #P2: 1 – 2 – 3 – 5 – 7 – 8
10. #P3: 1 – 2 – 3 – 6 – 7 – 8
11. #P4: 1 – 2 – 4 – 7 – 1 – . . . – 7 – 8

Loop Testing: Loops are widely used and these are fundamental to many algorithms hence, their testing is very important. Errors often occur at the beginnings and ends of loops.

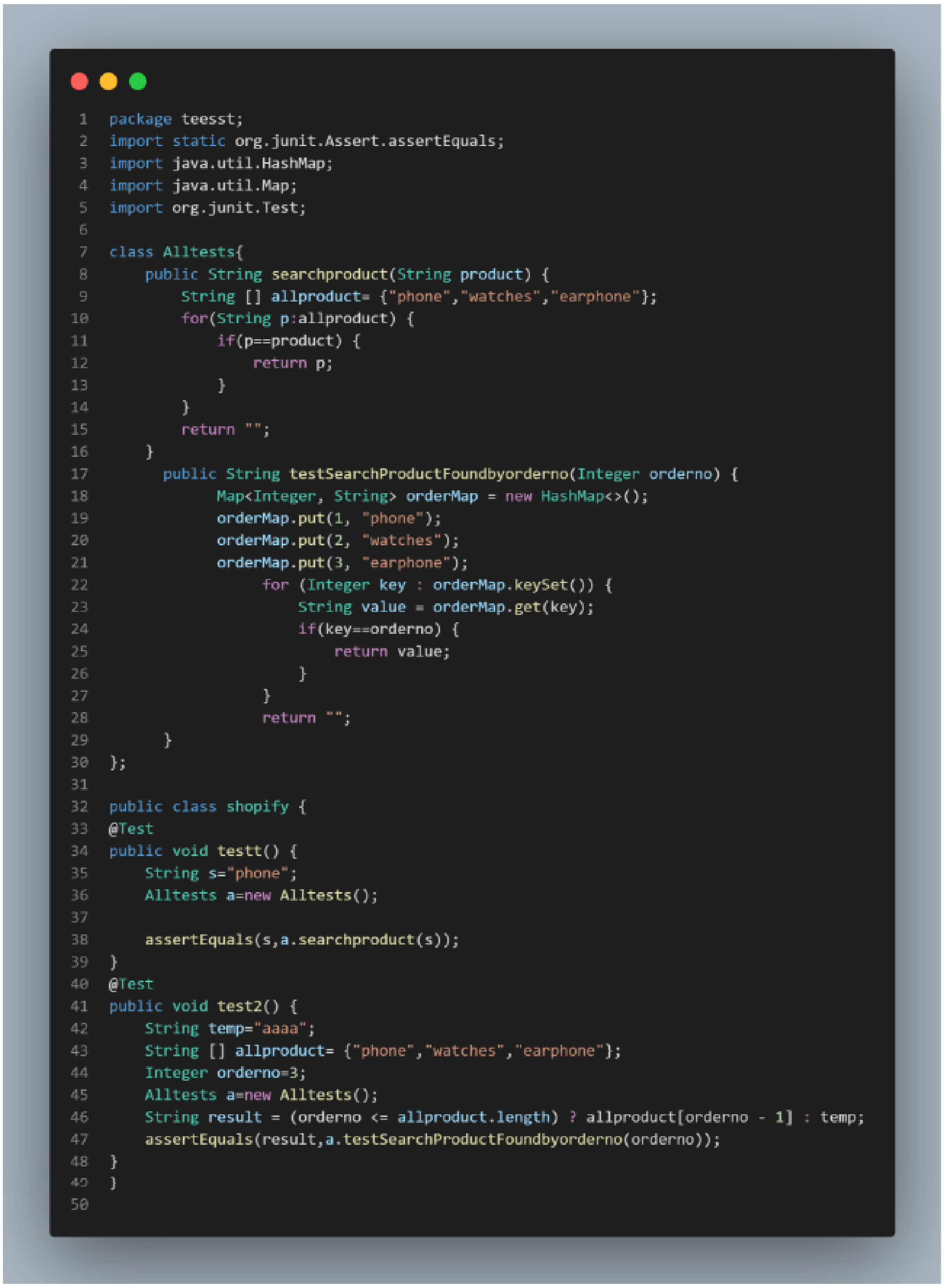
1. Simple loops: For simple loops of size n, test cases are designed that:

- Skip the loop entirely
- Only one pass through the loop
- 2 passes
- m passes, where m < n
- n-1 ans n+1 passes

2. Nested loops: For nested loops, all the loops are set to their minimum count and we start from the innermost loop. Simple loop tests are conducted for the innermost loop and this is worked outwards till all the loops have been tested.

3. Concatenated loops: Independent loops, one after another. Simple loop tests are applied for each. If they’re not independent, treat them like nesting.

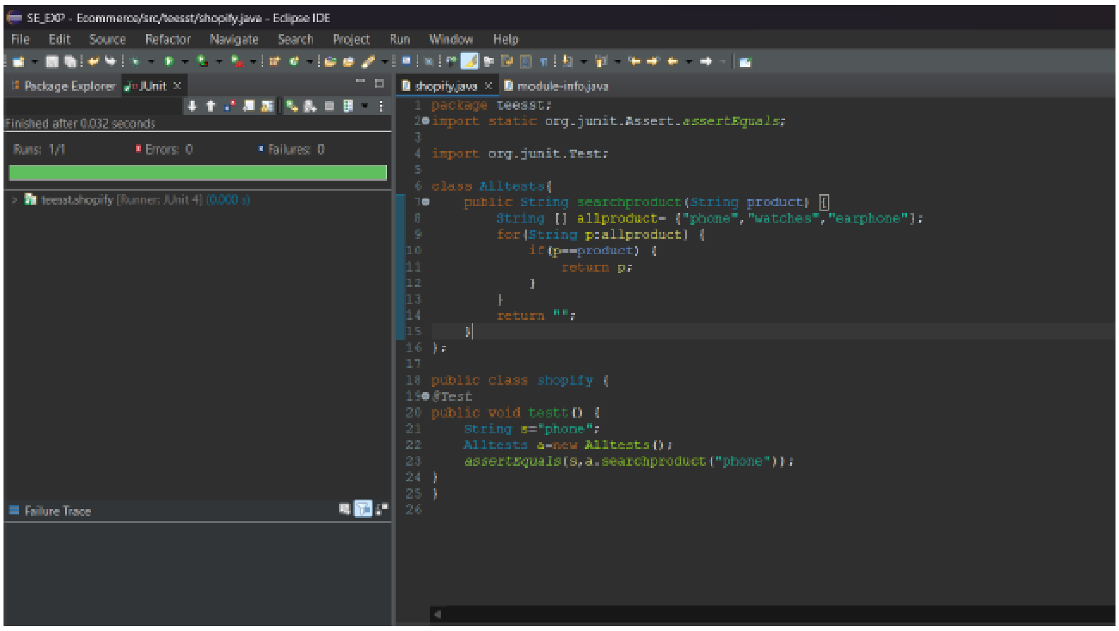
Code:



```java
package teesst;
import static org.junit.Assert.assertEquals;
import java.util.HashMap;
import java.util.Map;
import org.junit.Test;

class Alltests{
    public String searchproduct(String product) {
        String [] allproduct= {"phone","watches","earphone"};
        for(String p:allproduct) {
            if(p==product) {
                return p;
            }
        }
        return "";
    }
      public String testSearchProductFoundbyorderno(Integer orderno) {
            Map<Integer, String> orderMap = new HashMap<>();
            orderMap.put(1, "phone");
            orderMap.put(2, "watches");
            orderMap.put(3, "earphone");
                 for (Integer key : orderMap.keySet()) {
                     String value = orderMap.get(key);
                     if(key==orderno) {
                         return value;
                     }
                 }
                 return "";
      }
};

public class shopify {
@Test
public void testt() {
    String s="phone";
    Alltests a=new Alltests();

    assertEquals(s,a.searchproduct(s));
}
@Test
public void test2() {
    String temp="aaaa";
    String [] allproduct= {"phone","watches","earphone"};
    Integer orderno=3;
    Alltests a=new Alltests();
    String result = (orderno <= allproduct.length) ? allproduct[orderno - 1] : temp;
    assertEquals(result,a.testSearchProductFoundbyorderno(orderno));
}
}

```

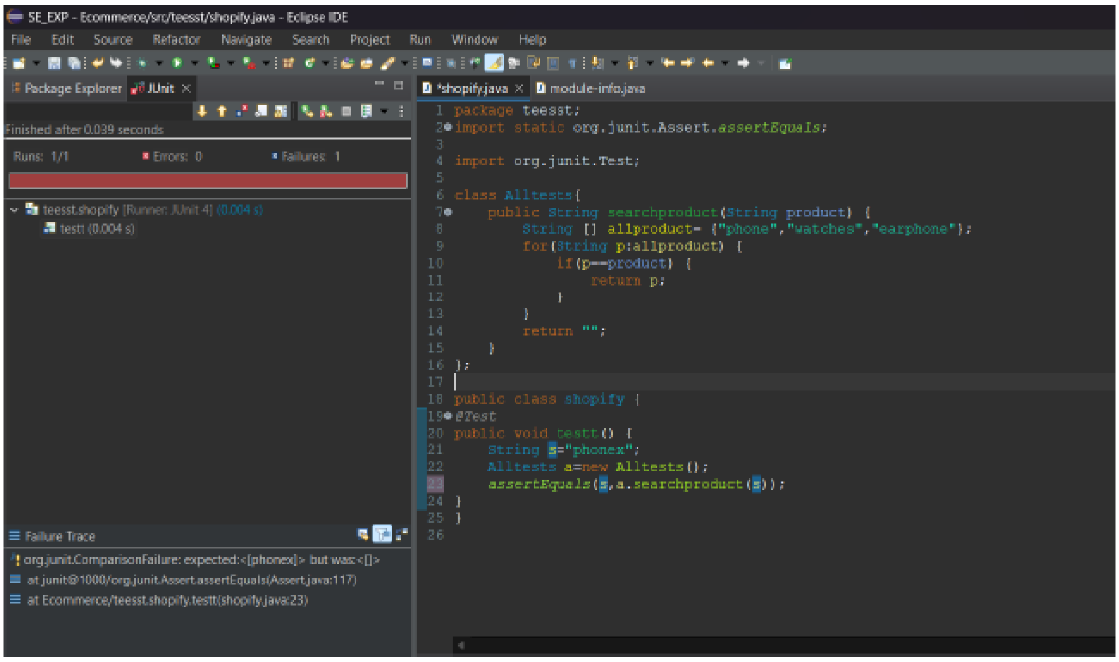
Output:

Test 1: Search a Product.


SE_EXP - Ecommerce/src/teesst/shopify.java - Eclipse IDE

```
Finished after 0.032 seconds
Runs: 1/1    Errors: 0    Failures: 0
> teesst.shopify [Runner: JUnit 4] (0.000 s)
```

```java
package teesst;
import static org.junit.Assert.assertEquals;

import org.junit.Test;

class Alltests{
    public String searchproduct(String product) {
        String [] allproduct= {"phone","watches","earphone"};
        for(String p:allproduct) {
            if(p==product) {
                return p;
            }
        }
        return "";
    }
};

public class shopify {
@Test
public void testt() {
    String s="phone";
    Alltests a=new Alltests();
    assertEquals(s,a.searchproduct("phone"));
}
}
```


SE_EXP - Ecommerce/src/teesst/shopify.java - Eclipse IDE

```
Finished after 0.039 seconds
Runs: 1/1    Errors: 0    Failures: 1
v teesst.shopify [Runner: JUnit 4] (0.004 s)
    testt (0.004 s)
```

```
org.junit.ComparisonFailure: expected:<[phonex]> but was:<[]>
at junit@1000/org.junit.Assert.assertEquals(Assert.java:117)
at Ecommerce/teesst.shopify.testt(shopify.java:23)
```

```java
package teesst;
import static org.junit.Assert.assertEquals;

import org.junit.Test;

class Alltests{
    public String searchproduct(String product) {
        String [] allproduct= {"phone","watches","earphone"};
        for(String p:allproduct) {
            if(p==product) {
                return p;
            }
        }
        return "";
    }
};

public class shopify {
@Test
public void testt() {
    String s="phonex";
    Alltests a=new Alltests();
    assertEquals(s,a.searchproduct(s));
}
}
```