EXPERIMRNT 12

Aim: To develop test cases for The Pet Finder Website using White Box testing

Theory:

Any engineered product (and most other things) can be tested in one of two ways:

- Knowing the specified function that a product has been designed to perform, tests can be conducted that demonstrate each function is fully operational while at the same time searching for errors in each function.
- Knowing the internal workings of a product, tests can be conducted to ensure that "all gears mesh," that is, internal operations are performed according to specifications and all internal components have been adequately exercised. The first test approach takes an external view and is called black-box testing. The second requires: an internal view and is termed white-box testing. Black-box testing alludes to tests that are conducted at the software interface.

Black-box testing delves into the fundamental aspects of a system, without much concern for how the software is internally structured. In contrast, white-box testing hinges on a meticulous examination of the procedural intricacies. It involves dissecting the logical paths within the software and scrutinizing how various components collaborate. On the surface, it might appear that rigorous white-box testing could yield perfectly error-free programs. The notion is that by defining all possible logical paths, devising comprehensive test cases, and meticulously assessing the results, we could ensure a flawless outcome. However, there's a practical challenge: even for relatively small programs, the number of potential logical paths can become overwhelmingly large. Still, it's essential to note that white-box testing remains a valuable practice. By focusing on a select number of critical logical paths and scrutinizing the integrity of vital data structures, it provides a pragmatic approach to enhance software reliability.

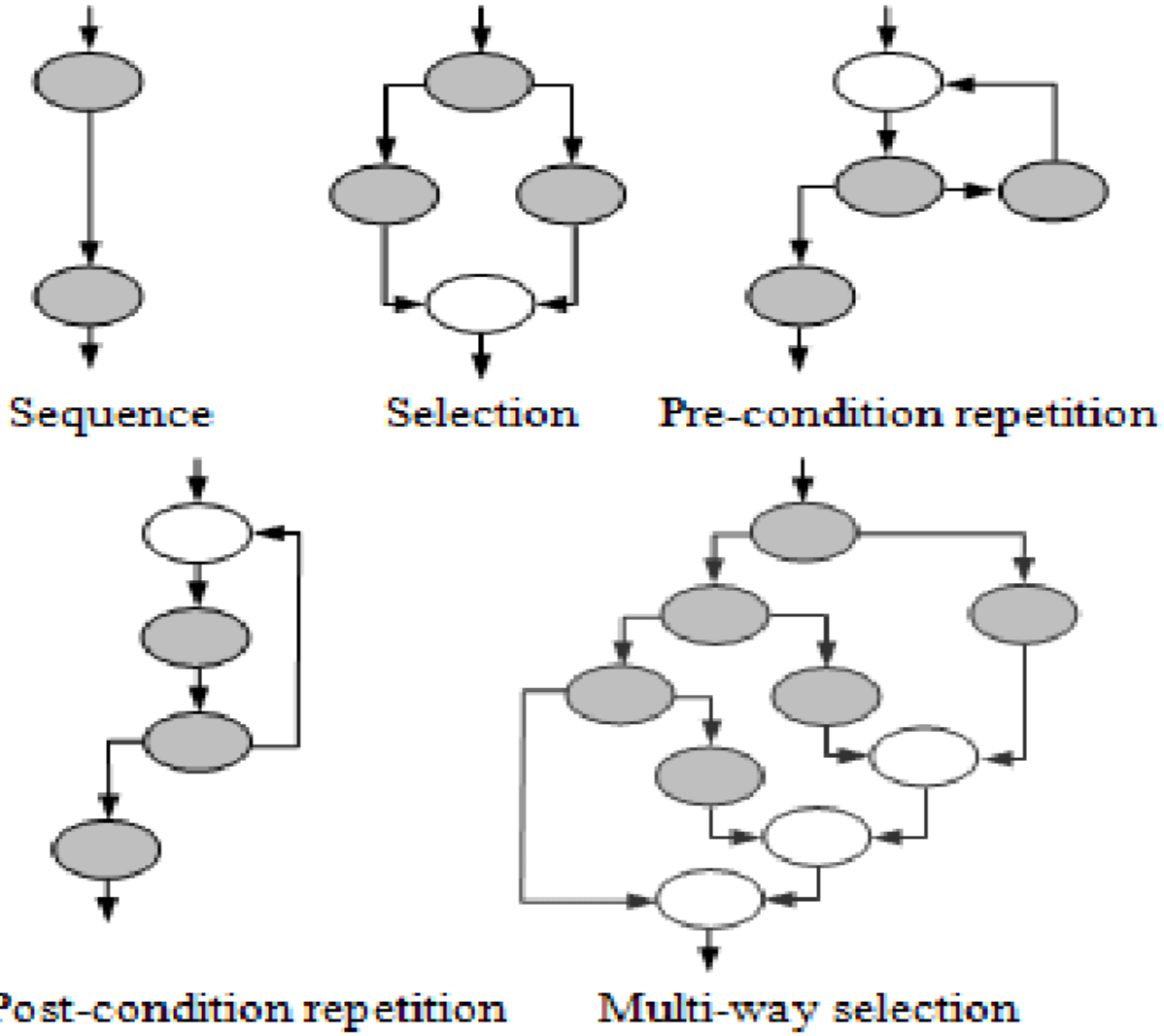
White-box testing, also known as glass-box testing, adopts a test-case design philosophy that leverages the control structure detailed in component-level

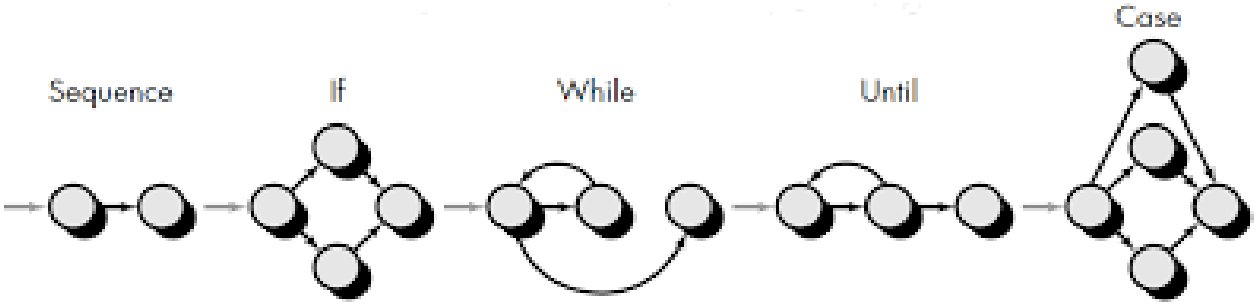
design. This approach facilitates the creation of test cases that serve several critical purposes:

1. **Coverage of All Independent Paths:** It ensures that every independent path within a module is traversed at least once during testing.
2. **Evaluation of Logical Decisions:** It comprehensively assesses all logical decisions from both their true and false perspectives.
3. **Loop Validation:** It verifies the behavior of loops by testing them at their boundaries and within their operational parameters.
4. **Data Structure Examination:** It involves scrutinizing the internal data structures to confirm their integrity and correctness.

In essence, white-box testing provides a methodical way to validate software by diving into its structural intricacies and ensuring that every aspect of its operation is rigorously examined.

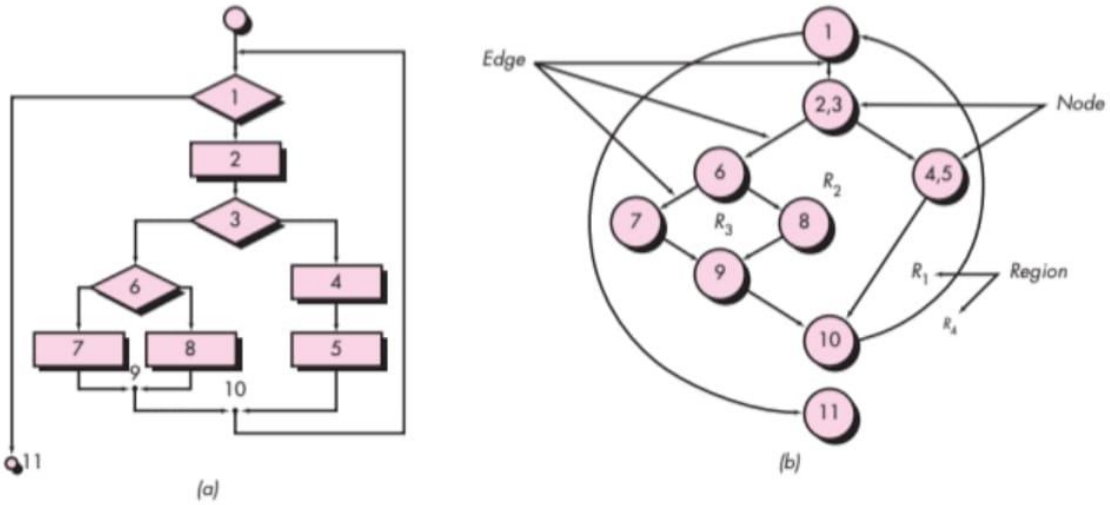
Basis Path Testing: Notion of a "basis" has attractive possibilities for structural testing. A basis in terms of a structure called a "vector space", which is a set of elements (called vectors) and which has operations that correspond to multiplication and addition defined for the vectors. Basis path testing is a white-box testing technique first proposed by Tom McCabe [MCC76]. The basis path method enables the test case designer to derive a logical complexity measure of a procedural design and use this measure as a guide for defining a basis set of execution paths. Test cases derived to exercise the basis set are guaranteed to execute every statement in the program at least one
time during testing.Cyclomatic complexity is a software metric that provides a quantitative measure of the logical complexity of a program. CC defines the number of independent paths in the basis set of a program and Provides us with an upper bound for the number of tests that must be conducted to ensure that all statements have been executed at least once. An independent path is any path through the program that introduces at least one new set of processing statements or a new condition. A simple notation for the representation of control flow, called a flow graph (or program graph) must be introduced. The flow graph depicts logical control flow using the following notation:





To illustrate the use of a flow graph, consider the procedural design representation in The figure below. Here, a flowchart is used to depict program control structure. Map the flowchart into a corresponding flow graph (assuming

that no compound conditions are contained in the decision diamonds of the flowchart). Each circle, called a flow graph node, represents one or more procedural statements. A sequence of process boxes and a decision diamond can map into a single node. The arrows on the flow graph, called edges or links, represent flow of control and are analogous to flowchart arrows. An edge must terminate at a node, even if the node does not represent any procedural statements (eg, see the flow graph symbol for the if-then-else construct). Areas bounded by edges and nodes are called regions. When counting regions, we include the area outside the graph as a region.



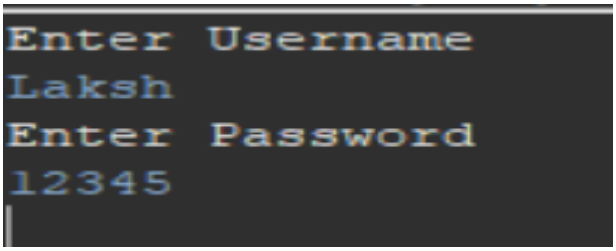
IMPLEMENTATION OF TEST CASE:

```
import static org.junit.Assert.*;
import org.junit.Test;
public class UserAuthenticationTest {
    @Test
    public void testAuthenticateUserValid() {
        // Arrange
        String username = "Laksh";
        String password = "12345";
        boolean isAuthenticated = UserAuthentication.authenticateUser(username,
password);
        assertTrue(isAuthenticated);
    }

    @Test
    public void testAuthenticateUserInvalid() {
        // Arrange
        String username = "Laksh";
        String password = " ib2XPqcFIRHmgVrJ ";

        // Act
        boolean isAuthenticated = UserAuthentication.authenticateUser(username,
password);

        // Assert
```

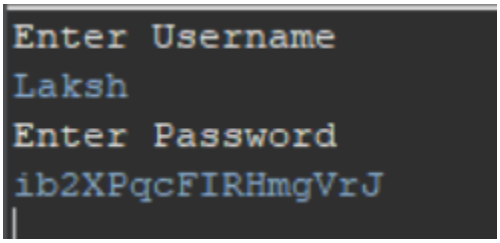
```
        assertFalse(isAuthenticated);
    }
}
```

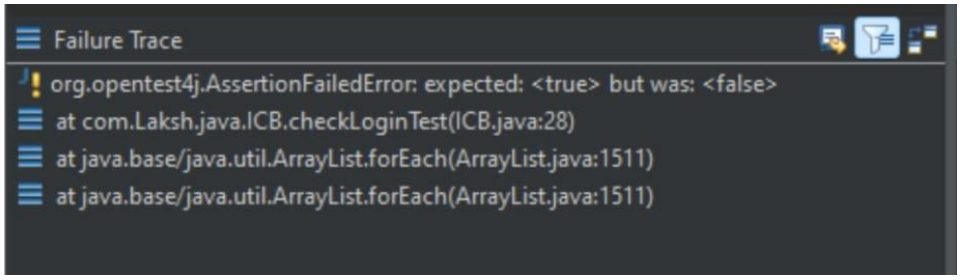
Output :

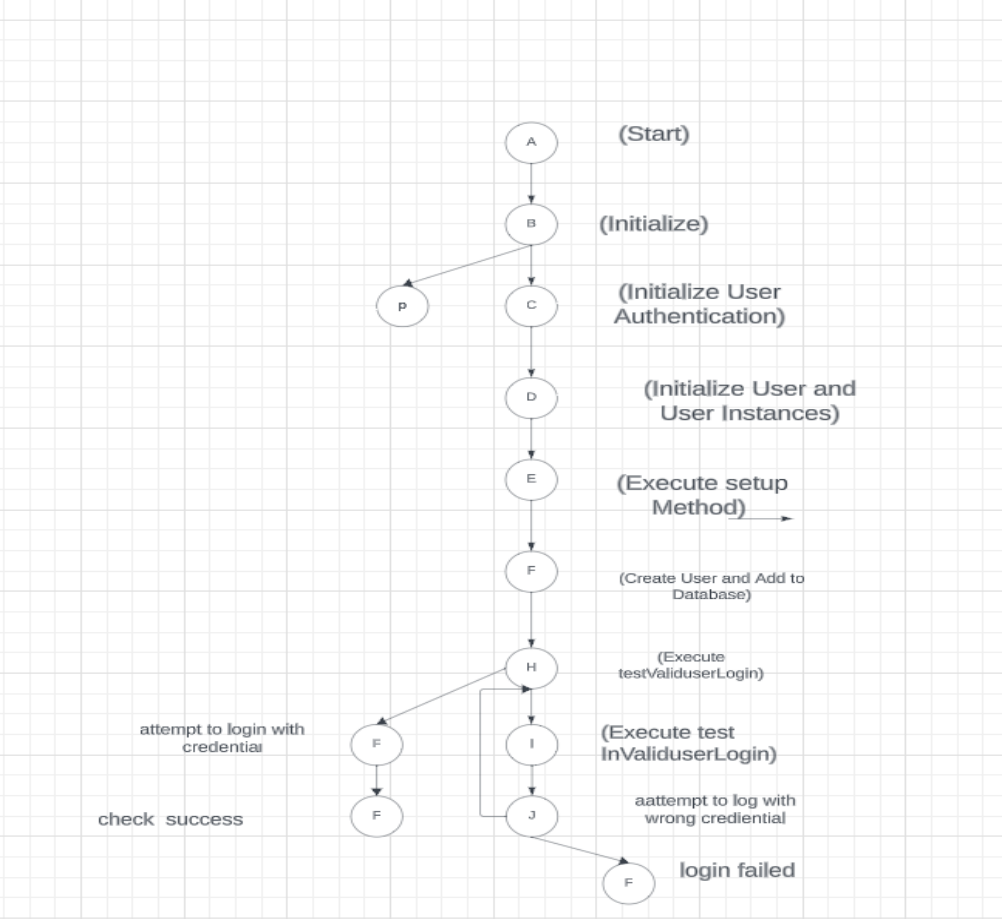
Valid:



Invalid:







Now, let's calculate the Cyclomatic Complexity:

- E (Edges) = 14

- N (Nodes) = 14

- P (Connected Components) = 1 (since the graph is a single connected component)

Using the formula:

V(G) = E - N + 2P

V(G) = 14 - 14 + 2*1

V(G) = 0 + 2

V(G) = 2

The cyclomatic complexity of the flow graph is 2.