SmartCliff
CAREER MOBILITY SOLUTIONS

SDE Readiness Training

Empowering Tomorrow's Innovators

Module I

Java Software Development: Effective Problem Solving

Understanding Test Cases

15.02.2025

What is a Test Case?

- A test case is a planned **set of inputs** and **conditions** used to check if a **program works correctly** and gives the **expected results**. It helps **identify bugs, ensure quality, and validate system behavior.**
- Test cases **play a crucial role** in problem-solving, whether in software development, coding interviews, or competitive programming. They help **ensure correctness, efficiency, and robustness of solutions.**

Types of Test Cases

- When solving a problem, test cases help **ensure correctness, efficiency, and robustness**. Below are **different types of test cases** commonly used in problem-solving.
- To understand the types of test cases, we can consider the problem of **writing a Java program to find the largest element in an array**

1. **Basic Test Cases (Sanity Checks)**
    - Verify the simplest expected inputs.
    - Ensure the core logic works.
    - **Example**
    - **Input:** [3, 7, 2, 9, 5].
    - **Expected Output**: 9

Types of Test Cases

2. Edge Test Cases

- Test the boundaries of input constraints.
- Includes minimum and maximum limits.
- **Example:**
- Finding the largest element in an array with only one element → **[42]**
- Checking behavior with an empty array → **[]**

3. Negative Test Cases

- Provide invalid inputs to test error handling.
- **Example:**
- Input: ["hello", 5, 10] (Invalid data type) (Passing String instead of integer)
- Expected Output: "Error: Invalid input"

Types of Test Cases

4. **Large Input Test Cases (Performance Tests)**
    - Ensure the solution runs efficiently for large inputs.
    - **Example:** Finding the largest number in an array of 1 million elements.
    - Checking time complexity with increasing input size.
    - **Input:** An array of 1,000,000 random numbers
    - **Expected Output**: Largest number from the array

5. **Random Test Cases**
    - Generate randomized inputs to check for unexpected failures.
    - **Example:** Generate an array with random numbers between -1000 and 1000.
    - **Input**: [17, -8, 100, 0, 56, -45]
    - **Expected Output**: 100

Types of Test Cases

6. **Special Case Test Cases**
    - Cover unique situations that might break the logic.
    - **Example:** All elements in the array are the same → [5, 5, 5, 5].
    - All negative numbers → [-10, -5, -3, -8].
    - **Input:** [-10, -5, -3, -8] (All negative numbers)
    - **Expected Output:** -3
7. **Stress Test Cases**
    - Generate randomized inputs to check for unexpected failures. Check system stability under extreme conditions.
    - **Example:** Find the largest element in an array with 100 million elements.
    - **Input:** An array of 100 million numbers
    - **Expected Output:** Largest number in the array (Ensuring efficient execution)

Importance of Considering Test Cases

- **Ensures Correctness** – Confirms the solution produces expected results.
- **Covers Edge Cases** – Handles extreme values and boundary conditions.
- **Improves Robustness** – Catches invalid inputs and prevents crashes.
- **Optimizes Performance** – Verifies efficiency on large inputs.
- **Prevents Regression Issues** – Ensures new changes don’t break old functionality.
- **Aids Debugging** – Helps identify issues faster and fix them easily.
- **Supports Automation** – Enables scalable and automated testing.

Note: Test cases make solutions reliable, efficient, and bug-free. Always test before deploying!

Hidden Test Case in Problem Solving

- A **hidden test case** is a test case that is **not visible to the programmer** when submitting a solution on coding platforms (e.g., LeetCode, CodeChef, HackerRank).
- **Why Are Hidden Test Cases Used?**
  - **Prevent Hardcoding** – Ensures solutions are dynamic, not written just for visible test cases.
  - **Check Edge Cases** – Tests boundary conditions, large inputs, and tricky scenarios.
  - **Evaluate Efficiency** – Ensures the solution runs within the required time complexity.
  - **Verify Robustness** – Detects errors not caught by public test cases.

Note: Hard coding means **directly embedding fixed values or outputs** in the code instead of writing a **dynamic, flexible solution** that works for all inputs. This approach often **fails for hidden test cases** in coding challenges.

Hard Coding: Example

```
public class HardCodedExample {
    public static void main(String[] args) {
        System.out.println("The largest number is: 9"); // Always prints 9
    }
}
```

This program **does not take input** and **always prints 9**, which means it will fail for other test cases.

To Avoid Hard Coding write program considering

- ✓ Takes input dynamically
- ✓ Works for any test case, including hidden ones
- ✓ Handles edge cases

How to Handle Hidden Test Cases?

- **Use Proper Input Handling** – Read inputs dynamically instead of assuming fixed values.
- **Think About Edge Cases** – Consider empty arrays, negative numbers, large inputs, etc.
- **Optimize for Performance** – Use efficient algorithms to handle large test cases within time limits.
- **Implement Proper Error Handling** – Handle invalid inputs gracefully (e.g., checking for null or empty arrays).
- **Test Before Submitting** – Run your own test cases beyond the sample ones given in the problem.

Summary

“To excel in problem-solving interviews, it's crucial to proactively think about edge cases while designing your solution to pass not just the visible but also the hidden test cases”.

Proper Preparation Prevents Poor Performance

- Charlie Batch


THANK YOU
SmartCliff
Career Mobility Solutions