Test Case-based Verification of Programs: Equivalence Partitions

Dr. Natarajan Meghanathan
Associate Professor of Computer Science
Jackson State University
E-mail: natarajan.meghanathan@jsums.edu

Test Cases Example

The following exercise involves

- writing a Java program,
- developing test cases, and
- altering the code to account for the test cases.

The basis of the Java program will be provided for you.

Example – Program Specification

Problem: Given two positive integers, determine whether the first one is larger than the second one.

Input: The input should be two positive integers.

Output: Print “Yes” if the first number is greater. Otherwise, print “No.”

Example - Code

```
import java.util.Scanner;
public class Compare {
    public static void main(String args[]) {
            Scanner in = new Scanner(System.in);
            System.out.print(“Enter the first positive integer: “);
            int first = in.nextInt();
            System.out.print(“Enter the second positive integer: “);
            int second = in.nextInt();
            if (first > second)
                        System.out.println(“Yes”);
            else
                        System.out.println(“No”);
    }
}
```

Input Types

We will now develop test cases for this program using the method of equivalent partitions.

The input could be of several types:

- Integer
- Non-integer
- Positive
- Negative

We need to develop our equivalent partitions for these scenarios.

Test Cases

Equivalent partitions allow us to simplify the amount of test cases we will need to use.

Developing equivalent partitions allows us to group many different specific input values into larger classes of input.

By determining how the program reacts to one set of input from a particular class of input (from a particular partition), we can assume that the program will act in a similar manner to all sets of input from the same class or partition.

In this way, we can use a handful of test cases to test almost any input our program could receive.

Equivalent Partitions

Of our four types of input (positive, negative, integer, non-integer), the largest type is non-integer.

We will begin creating our partitions by separating our input into integer and non-integer.

Non-Integer	Integer

Equivalent Partitions

Since any input that is not an integer is not valid, according to our input specification, we do not need to make any further partitions in the Non-Integer category.

These means that all non-integers will be treated the same.

Equivalent Partitions

In our Integer category, there can be either positive or negative integers (0 will be considered positive).

Therefore, we must create 2 partitions inside of the Integer partition: Positive and Negative.

Non-Integer	Integer	
	Positive	Negative

Equivalent Partitions

We now have 3 equivalent partitions:

- Non-Integer
- Integer, Positive
- Integer, Negative

Now that we have considered the input types which might be provided to the program, we must consider what the program does.

Equivalent Partitions

For our program, if the input is of the correct type (2 positive integers), there are three different scenarios that can take place:

- The first number is larger
- The second number is larger
- The numbers are equal

We must now create partitions for these three scenarios within our “Integer, Positive” partition.

Equivalent Partitions

Non-Integer	Integer			
	Positive			Negative
	First Number Larger	Second Number Larger	First and Second Numbers Equal	
(1)	(2)	(3)	(4)	(5)

Test Cases

Each test case will consist of the following fields:

- Name
- Input
- Oracle (expected output)
- Log (actual output)

The test case names will correspond to the numbers at the bottom of each equivalent partition.

For example, Test Case (1) will be the test case for Non-Integers, and Test Case (5) will be the test case for Negative Integers.

Test Cases

Name:	Test Case (1)
Input:	5.17, 3.12
Oracle:	Enter only positive integers
Log:	

Name:	Test Case (2)
Input:	8, 4
Oracle:	Yes
Log:	

Test Cases

Name:	Test Case (3)
Input:	3, 5
Oracle:	No
Log:	

Name:	Test Case (4)
Input:	4, 4
Oracle:	No
Log:	

Test Cases

Name:	Test Case (5)
Input:	-3, -8
Oracle:	Enter only positive integers
Log:	

Testing

Run the program with each set of input data and complete the “Log:” portions of the test cases.

Test Cases

Upon completing the Test Case Logs, you will find that for Test Case (1), instead of providing the expected “Yes,” the program threw an InputMismatchException.

We must adjust the code so that this Test Case is properly handled by issuing an appropriate message to the user.

Program Alteration

We will adjust the code be encapsulating the integer-specific parts of our program within a *try* block.

Immediately following the *try* block, we will insert a *catch* block, which will catch all exceptions (including the InputMismatchException) and issue a warning message to the user.

Updated Code

```
import java.util.Scanner;
public class Compare {
	public static void main(String args[]) {
		Scanner in = new Scanner(System.in);
		try{
			System.out.print("Enter the first positive integer: ");
			int first = in.nextInt();
			System.out.print("Enter the second positive integer: ");
			int second = in.nextInt();
			if (first > second)
				System.out.println("Yes");
			else
				System.out.println("No");
		}//end try block
		catch (Exception e) {
			System.out.println("Please enter only positive integers.");
		}//end catch block
	}//end main() method
}//end Compare class
```

Test Cases

We must now adjust our code to properly handle Test Case (5) which returned a result, but should not have accepted the input.

The input should not have been accepted because the input specification states that only positive integers should be accepted.

Program Alteration

We will do this by inserting an *if* statement after the integers are received from the user.

If either of the numbers is less than zero, we will issue a warning to the user.

Updated Code

```
import java.util.Scanner;
public class Compare {
    public static void main(String args[]) {
        Scanner in = new Scanner(System.in);
        try{
            System.out.print(“Enter the first positive integer: “);
            int first = in.nextInt();
            System.out.print(“Enter the second positive integer: “);
            int second = in.nextInt();
            if ( (first < 0) || (second < 0) )
            {
                System.out.println(“Please enter only positive integers.”);
                return;
            }//end if
            if (first > second)
                System.out.println(“Yes”);
            else
                System.out.println(“No”);
        }//end try block
        catch (Exception e) {
            System.out.println(“Please enter only positive integers.”);
        }//end catch block
    }//end main() method
}//end Compare class
```

Testing

You should have found that now the program will only accept values from the user that are positive integers, just as the input specification stated.