UNIT TESTING - Basics

JUnit

JUnit is an open-source framework for writing and running automated tests in Java. It is used to test individual units of code, such as classes or methods, to ensure they work as expected.

NOTE : The org.junit package contains many interfaces and classes for junit testing such as Assert, Test, Before, After etc.

Why use JUnit testing?

Find bugs early: JUnit helps developers find and fix bugs before they become more complicated.
Improve code quality: JUnit helps developers create more reliable and bug-free code.
Support test-driven development: JUnit follows the principles of test-driven development (TDD).

Types of unit testing

There are two ways to perform unit testing:

1. manual testing
2. automated testing.

Manual Testing

If you execute the test cases manually without any tool support, it is known as manual testing. It is time consuming and less reliable.

Automated Testing

If you execute the test cases by tool support, it is known as automated testing. It is fast and more reliable.

What can JUnit test?

Unit tests: Test individual units of code, such as classes or methods
Integration tests: Test how all the components of an application work together
System tests: Test entire systems, such as web servers.

Regression testing

Regression testing is a type of software testing that verifies that new changes to an application don't break existing functionality. It's a key part of software quality assurance (QA) and is performed after every change to the code.

Why is it important?

Regression testing helps prevent old bugs from reappearing
It helps ensure that new features work as expected
It helps improve the user experience
It helps reduce defects during release

How is it performed?

Regression testing involves re-running test cases that were originally created to fix known problems
The test results are compared against a baseline of expected behavior
Any issues are reported to the development team for resolution
Once the issues are fixed, the test cases are re-run to confirm the fixes.

When is it performed?

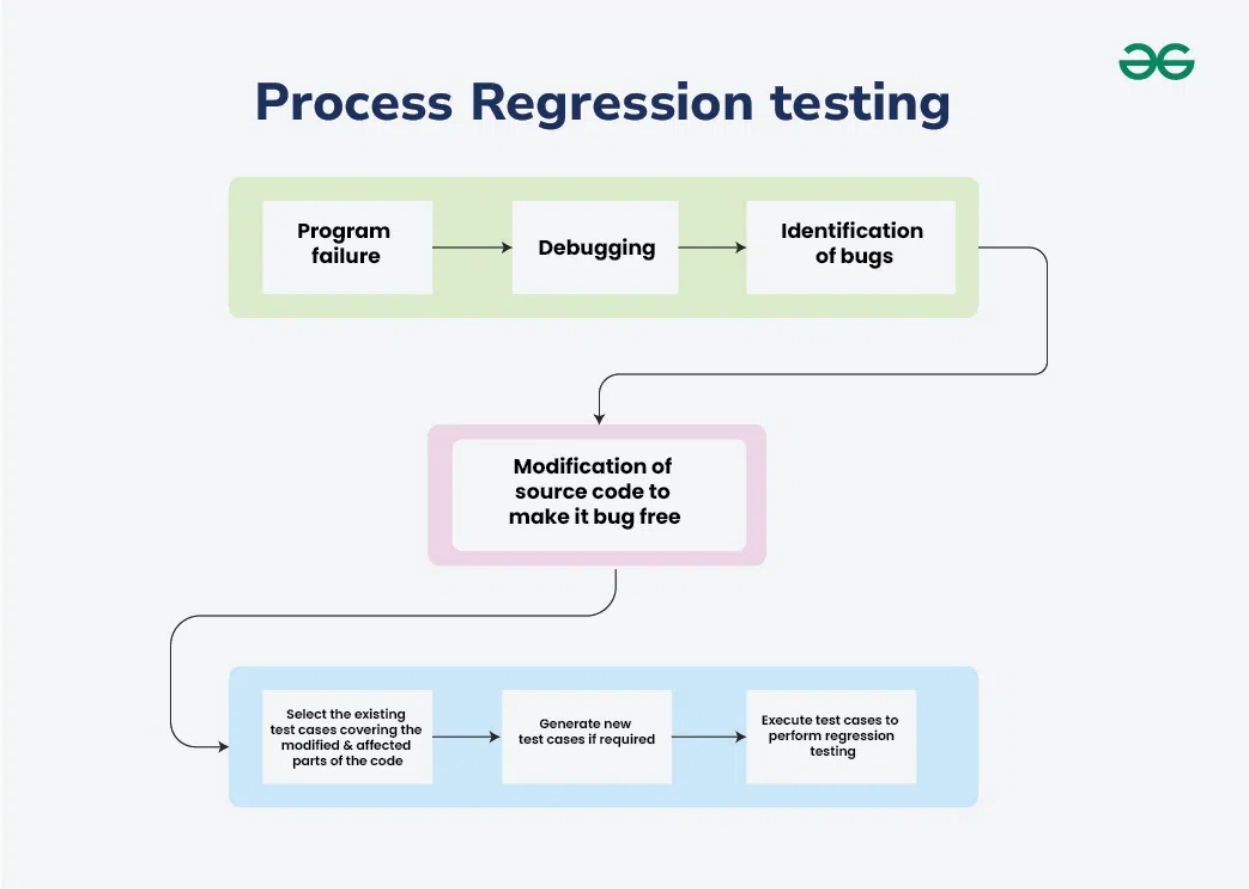
Regression testing is performed after any changes to the code, such as new features, bug fixes, or software enhancements.

When to do regression testing?

- When new functionality is added to the system and the code has been modified to absorb and integrate that functionality with the existing code.
- When some defect has been identified in the software and the code is debugged to fix it.
- When the code is modified to optimize its working.

Process of Regression testing

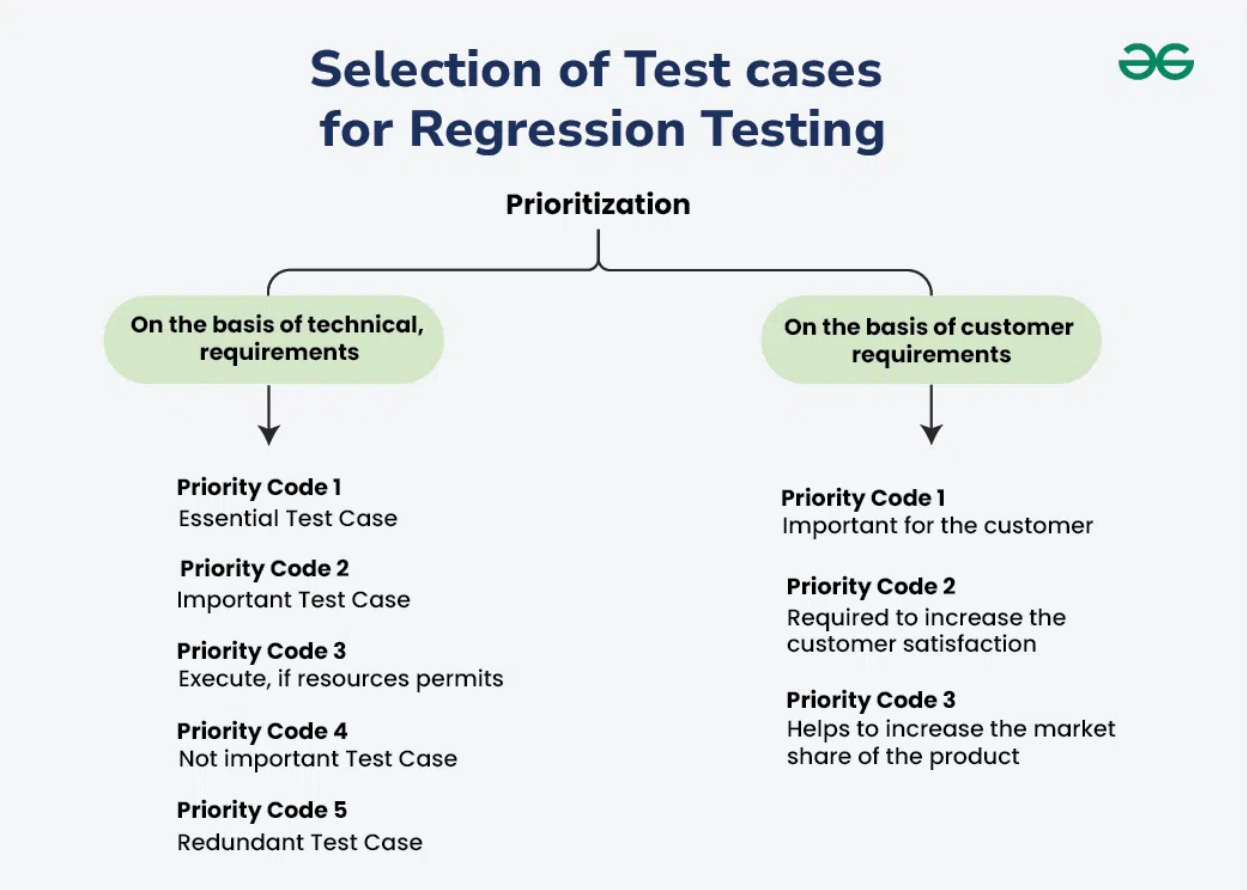
Firstly, whenever we make some changes to the source code for any reason like adding new functionality, optimization, etc. then our program when executed fails in the previously designed test suite for obvious reasons. After the failure, the source code is debugged to identify the bugs in the program. After identification of the bugs in the source code, appropriate modifications are made. Then appropriate test cases are selected from the already existing test suite which covers all the modified and affected parts of the source code. We can add new test cases if required. In the end, regression testing is performed using the selected test cases.



Techniques for the selection of Test cases for Regression Testing

- Select all test cases: In this technique, all the test cases are selected from the already existing test suite. It is the simplest and safest technique but not very efficient.
- Select test cases randomly: In this technique, test cases are selected randomly from the existing test suite, but it is only useful if all the test cases are equally good in their fault detection capability which is very rare. Hence, it is not used in most of the cases.
- Select modification traversing test cases: In this technique, only those test cases are selected that cover and test the modified portions of the source code and the parts that are affected by these modifications.
- Select higher priority test cases: In this technique, priority codes are assigned to each test case of the test suite based upon their bug detection capability,

customer requirements, etc. After assigning the priority codes, test cases with the highest priorities are selected for the process of regression testing. The test case with the highest priority has the highest rank. For example, a test case with priority code 2 is less important than a test case with priority code 1.



Advantages of Regression Testing

1. Automated unit testing
2. Comprehensive test coverage
3. System integration
4. Faster test execution completion
5. Improved developer productivity
6. Parallel testing

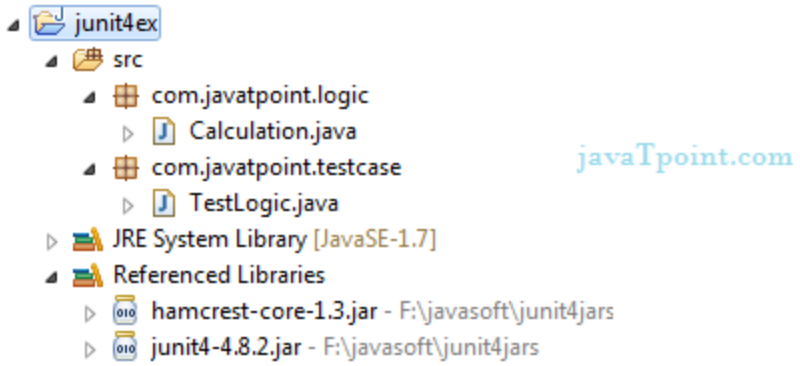
7. Reduced costs
8. Regression testing improves product quality
9. Reusability
10. Scalability
11. Time efficiency

Disadvantages of Regression Testing

1. It can be time and resource-consuming if automated tools are not used.
2. It is required even after very small changes in the code.
3. Time and Resource Constraints.
4. Among the significant risks associated with regression testing are the time and resources 5. required to perform it.
5. Incomplete or Insufficient Test Coverage.
6. False Positives and False Negatives.
7. Test Data Management Challenges.

MORE INFORMATION - https://www.geeksforgeeks.org/software-engineering-regression-testing/

Simple JUnit example in eclipse IDE



Write the program logic

Let's write the logic to find the maximum number for an array.

```
package com.javatpoint.logic;
public class Calculation {

    public static int findMax(int arr[]){
        int max=0;
        for(int i=1;i<arr.length;i++){
            if(max<arr[i])
                max=arr[i];
        }
        return max;
    }
}
```

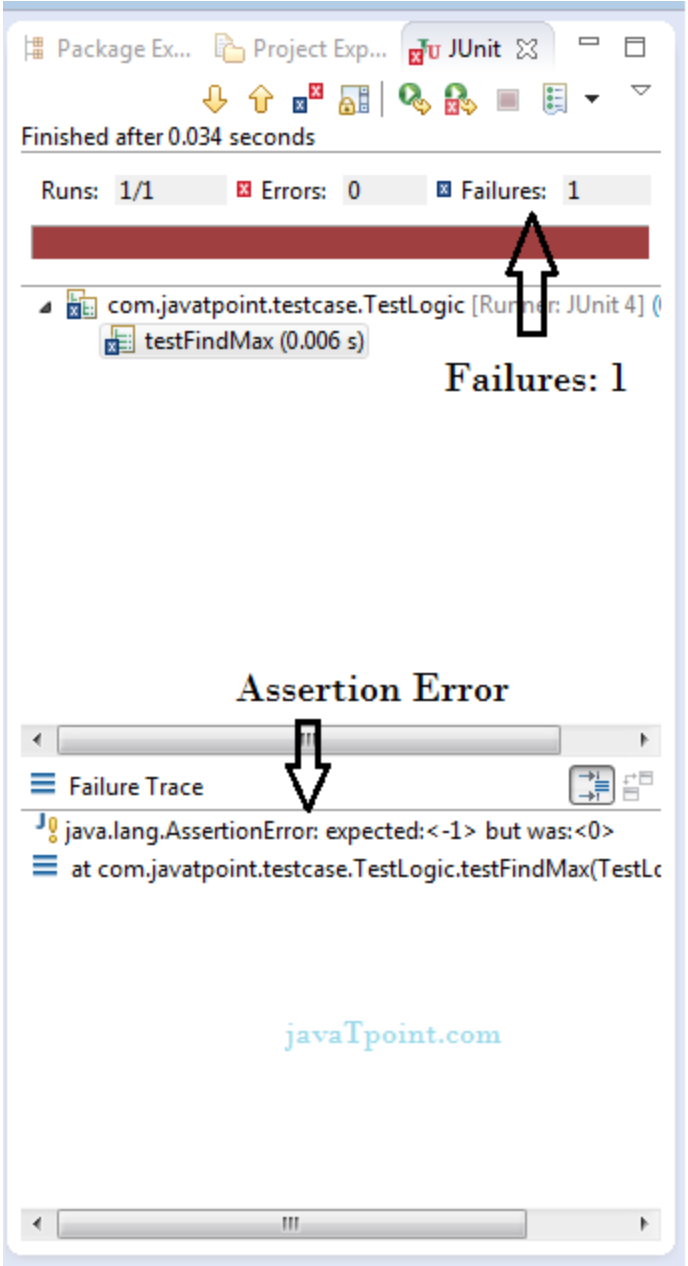
Write the test case

Here, we are using JUnit 4, so there is no need to inherit TestCase class. The main testing code is written in the testFindMax() method. But we can also perform some task before and after each test, as you can see in the given program.

```
package com.javatpoint.testcase;

import static org.junit.Assert.*;
import com.javatpoint.logic.*;
import org.junit.Test;

public class TestLogic {

    @Test
    public void testFindMax(){

        assertEquals(4,Calculation.findMax(new int[]{1,3,4,2}));
        assertEquals(-1,Calculation.findMax(new int[]{-12,-1,-3,
    }
}
```

To run this example, **right click on TestLogic class → Run As → 1Junit Test**.

`Output:` `Assertion Error`



As you can see, when we pass the negative values, it throws AssertionError because second time findMax() method returns 0 instead of -1. It means our program logic is incorrect.

Correct program logic

As you can see, program logic to find the maximum number for the given array is not correct because it doesn't return -1 in case of negative values. The correct program logic is given below:

```
package com.javatpoint.logic;
public class Calculation {

    public static int findMax(int arr[]){
        int max=arr[0];  //arr[0] instead of 0
        for(int i=1;i<arr.length;i++){
            if(max<arr[i])
                max=arr[i];
        }
        return max;
    }
}
```

If you run the junit program again, you will see the following output.

Another example of Junit framework

Write the program code

```
package com.javatpoint.logic;
public class Calculation {
    //method that returns maximum number
    public static int findMax(int arr[]){
        int max=0;
```

```
        for(int i=1;i<arr.length;i++){
            if(max<arr[i])
                max=arr[i];
        }
        return max;
    }
    //method that returns cube of the given number
    public static int cube(int n){
        return n*n*n;
    }
    //method that returns reverse words
    public static String reverseWord(String str){

        StringBuilder result=new StringBuilder();
        StringTokenizer tokenizer=new StringTokenizer(str," ");

        while(tokenizer.hasMoreTokens()){
        StringBuilder sb=new StringBuilder();
        sb.append(tokenizer.nextToken());
        sb.reverse();

        result.append(sb);
        result.append(" ");
        }
        return result.toString();
    }
}
```

Write the test case

```
package com.javatpoint.testcase;

import static org.junit.Assert.assertEquals;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
```

```
import org.junit.BeforeClass;
import org.junit.Test;
import com.javatpoint.logic.Calculation;

public class TestCase2 {

    @BeforeClass
    public static void setUpBeforeClass() throws Exception {
        System.out.println("before class");
    }
    @Before
    public void setUp() throws Exception {
        System.out.println("before");
    }

    @Test
    public void testFindMax(){
        System.out.println("test case find max");
        assertEquals(4,Calculation.findMax(new int[]{1,3,4,2}));
        assertEquals(-2,Calculation.findMax(new int[]{-12,-3,-4,
    }
    @Test
    public void testCube(){
        System.out.println("test case cube");
        assertEquals(27,Calculation.cube(3));
    }
    @Test
    public void testReverseWord(){
        System.out.println("test case reverse word");
        assertEquals("ym eman si nahk",Calculation.reverseWord("
    }
    @After
    public void tearDown() throws Exception {
        System.out.println("after");
    }
```

```
    @AfterClass
    public static void tearDownAfterClass() throws Exception {
        System.out.println("after class");
    }

}
```

```
Output:before class
before
test case find max
after
before
test case cube
after
before
test case reverse word
after
after class
```