class Books

{

public List <String> getBooks()

{

return Collections.emptyList();

}

}

create test class

public class BooksSpec{

@Test

public void BooksEmptyorNOt(){

//Setup the books

Books books=new Books();

//acting calling the method

List<String> list= books.getBooks();

//assertions of result

//assertTrue(list.isEmpty(),()-> "Books should be empty");

//assertTrue(list.isEmpty());

}

}

run the test cases

again make changes

assertFalse(list.isEmpty());

then change the main class return the null data.

public List<String> getBooks(){

return null;

}

and in test case

assertTrue(list.isEmpty());

Scenario

Suppose we have a class called StringListProcessor that processes a list of strings by adding, removing, and retrieving elements.

StringListProcessor Class

import java.util.ArrayList;

import java.util.List;

public class StringListProcessor {

private List<String> stringList;

public StringListProcessor() {

this.stringList = new ArrayList<>();

}

// Adds a string to the list

public void addString(String str) {

stringList.add(str);

}

// Removes a string from the list

public boolean removeString(String str) {

return stringList.remove(str);

}

// Retrieves the string at a specific index

public String getString(int index) {

if (index < 0 || index >= stringList.size()) {

throw new IndexOutOfBoundsException("Invalid index");

}

return stringList.get(index);

}

// Returns the current size of the list

public int getSize() {

return stringList.size();

}

}

JUnit Test Class for StringListProcessor

import static org.junit.jupiter.api.Assertions.\*;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

public class StringListProcessorTest {

private StringListProcessor processor;

@BeforeEach

void setUp() {

processor = new StringListProcessor();

}

@Test

void testAddString() {

processor.addString("Hello");

processor.addString("World");

assertEquals(2, processor.getSize(), "Size should be 2 after adding two strings");

}

@Test

void testRemoveString() {

processor.addString("Hello");

processor.addString("World");

boolean removed = processor.removeString("Hello");

assertTrue(removed, "The string 'Hello' should be removed successfully");

assertEquals(1, processor.getSize(), "Size should be 1 after removing one string");

removed = processor.removeString("NonExistent");

assertFalse(removed, "Removing a non-existent string should return false");

assertEquals(1, processor.getSize(), "Size should remain 1 after trying to remove a non-existent string");

}

@Test

void testGetString() {

processor.addString("Hello");

processor.addString("World");

String str = processor.getString(0);

assertEquals("Hello", str, "The first element should be 'Hello'");

str = processor.getString(1);

assertEquals("World", str, "The second element should be 'World'");

}

@Test

void testGetStringWithInvalidIndex() {

processor.addString("Hello");

assertThrows(IndexOutOfBoundsException.class, () -> {

processor.getString(2);

}, "Getting an element with an invalid index should throw an IndexOutOfBoundsException");

}

@Test

void testGetSize() {

assertEquals(0, processor.getSize(), "Size should be 0 for an empty list");

processor.addString("Hello");

assertEquals(1, processor.getSize(), "Size should be 1 after adding one string");

processor.addString("World");

assertEquals(2, processor.getSize(), "Size should be 2 after adding two strings");

processor.removeString("Hello");

assertEquals(1, processor.getSize(), "Size should be 1 after removing one string");

}

}

Explanation of the Tests

testAddString():

This test checks if strings are correctly added to the list. After adding two strings, the size of the list is asserted to be 2.

testRemoveString():

This test verifies that a string can be removed from the list. It checks both successful and unsuccessful removals and asserts the size of the list after each operation.

testGetString():

This test retrieves strings from the list by their index and checks if the correct string is returned.

testGetStringWithInvalidIndex():

This test ensures that attempting to retrieve a string using an invalid index throws an IndexOutOfBoundsException.

testGetSize():

This test checks the size of the list at various stages: when the list is empty, after adding elements, and after removing elements.

Running the Tests

You can run these tests using your IDE, Maven, or Gradle, as described in the previous examples. Each test method should pass, confirming that the StringListProcessor class behaves as expected.