Chapter 4: Software Testing and Test-driven development (TDD)

1. Write tests for converting temperatures from Celsius to Fahrenheit and vice versa.
   Hint: Use **assertEquals(expected, actual)** to compare the expected result with the actual result returned by the method.

2. Write a simple method in a **Calculator** class that adds two integers. Then, create a JUnit test case to verify that the method works correctly by adding two numbers together.

3. Write a class **BankAccount** with methods **deposit(double amount)** and withdraw(double amount). The account balance should start at 0.0, and the methods should update the balance accordingly.
   Write a JUnit test that:
   - Ensures a deposit of 100.0 increases the balance to 100.0.
   - Ensures a withdrawal of 50.0 decreases the balance to 50.0.
   - Verifies that a withdrawal of 60.0 fails (balance should remain 50.0)

4. Create a method **getEvenNumbers(int[] numbers)** in a **NumberUtils** class that filters out and returns only the even numbers from a given array of integers. Write a JUnit test case to verify that the method correctly returns a list of even numbers.
   For example:
   Input: [1, 2, 3, 4, 5, 6]
   Expected Output: [2, 4, 6]

5. **Complex Assertion with assertAll**
   Write a class **Product** with fields **name** (String), **price** (double), and **quantity** (int). Write a method **isAffordable**(double budget) that returns true if the total price (price * quantity) is less than or equal to the given budget. Write a JUnit test that:
   - Verifies that the name is not null.
   - Verifies that the price is a positive value.
   - Verifies that the isAffordable() method works correctly with different budgets using assertAll.

6. In an inventory management system, you need a method **isProductAvailable(String productName, int quantity)** to check if the given product is in stock. The method should return true if the requested quantity is available in stock and false if the requested quantity exceeds the available stock.

7. In a notification service, you need to implement a **sendEmail(String email, String message)** method to send an email. The method should return true if the email is sent successfully and false if the email address is invalid.

8. In an Learning management system, students can enroll in courses. The **EnrollmentService** class needs a method **enrollStudent(String studentUsername, String courseName)** to allow students to enroll in courses. The method should return true if the student is successfully enrolled, and false if the student is already enrolled in the course.

9. Create a class StringManipulator with the following methods:
   a. **reverse(String input):**
   This method should take a string and return the reversed version of the string.
   b. **toUpperCase(String input):**
   This method should convert all characters of the given string to uppercase.
   c. **isPalindrome(String input):**
   This method should return true if the input string is a palindrome (i.e., it reads the same forwards and backwards), and false otherwise.
   d. **countVowels(String input):**
   This method should count and return the number of vowels (a, e, i, o, u) in the input string.

   Write a single JUnit test case using **assertAll** to verify all the methods of the **StringManipulator** class.

10. You are developing a basic calculator application with operations like addition, subtraction, multiplication, and division. Each test case checks a specific operation.
    Tasks:
    Write a JUnit test using annotations that:
    - **Before** each test, initializes a Calculator object.
    - **After** each test, resets any necessary states or prints a message.
    - **BeforeClass**: Set up any global configuration (if needed).
    - **AfterClass**: Perform any clean-up after all tests are completed (e.g., release resources if any).

11. You are given a **LibraryService** class that manages books in a library. The **LibraryService** allows adding books to the library and searching for books by title. The class uses an internal **ArrayList** to store the books.
    Your task is to write unit tests for the **LibraryService** class. You will need to test the methods for adding and searching for books using JUnit. Additionally, you must use the JUnit annotations (**@Before, @BeforeClass**, **@After**, **@AfterClass**) to manage setup and cleanup of resources during the tests.

Follow the TDD Approach

12. Write a function that takes an integer as input and returns True if it is a prime number, otherwise returns False.

13. Write a function to calculate the factorial of a given non-negative integer.

14. Create a class Rectangle with the following:

- Attributes: length and width.
- Methods: area() to calculate the area of the rectangle.

  perimeter() to calculate the perimeter of the rectangle.

- Create a test cases

15. Create a base class Shape with a method area() that returns 0.

Create two derived classes:

- Circle with attribute radius and area() method to calculate the area
- Rectangle with attributes length and width and area() method to calculate the area.