Homework Hustlers: https://discord.gg/aJ55rZBV

- Wizard.

Write tests for converting temperatures from Celsius to Fahrenheit and vice versa.

```
~

import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class TempConversionTest {

    @Test
    public void testCelsiusToFahrenheit() {
        assertEquals(32.0, tmpConversion.celciusToFer(0), 0.01);
        assertEquals(212.0, tmpConversion.celciusToFer(100), 0.01);
    }

    @Test
    public void testFahrenheitToCelsius() {
        assertEquals(0.0, tmpConversion.ferToCel(32), 0.01);
        assertEquals(100.0, tmpConversion.ferToCel(212), 0.01);
    }
}

class tmpConversion {
    public static double celciusToFer(double celsius) {
        return (celsius * 9/5) + 32;
    }

    public static double ferToCel(double fahrenheit) {
        return (fahrenheit - 32) * 5/9;
    }

}
```

```
~
```

```
.
├─ JUnit Jupiter ✔
└─ JUnit Vintage ✔
   └─ TempConversionTest ✔
      ├─ testFahrenheitToCelsius ✔
      └─ testCelsiusToFahrenheit ✔

Test run finished after 63 ms
[         3 containers found      ]
[         0 containers skipped    ]
[         3 containers started    ]
[         0 containers aborted    ]
[         3 containers successful ]
[         0 containers failed     ]
[         2 tests found           ]
[         0 tests skipped         ]
[         2 tests started         ]
[         0 tests aborted         ]
[         2 tests successful      ]
[         0 tests failed          ]
```

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.

```
~

import org.junit.Test;
import static org.junit.Assert.assertEquals;
class add{
    public static int add(int a, int b) {
        return a + b;
    }
    public static void main(String[] args) {

    }
}
public class addTest{
    @Test
    public void testAdd() {
        assertEquals(5, add.add(2, 3));
    }

}
```

```
~

Thanks for using JUnit! Support its development at https://junit.org/sponsoring

╷
├─ JUnit Jupiter ✔
└─ JUnit Vintage ✔
   └─ addTest ✔
      └─ testAdd ✔

Test run finished after 58 ms
[         3 containers found      ]
[         0 containers skipped    ]
[         3 containers started    ]
[         0 containers aborted    ]
[         3 containers successful ]
[         0 containers failed     ]
[         1 tests found           ]
[         0 tests skipped         ]
[         1 tests started         ]
[         0 tests aborted         ]
[         1 tests successful      ]
[         0 tests failed          ]
```

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)

```
~

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

public class BankAccountTest {

    @Test
    public void testDeposit() {
        BankAccount account = new BankAccount();
        account.deposit(100.0);
        assertEquals(100.0, account.getBalance());
    }

    @Test
    public void testWithdraw() {
        BankAccount account = new BankAccount();
        account.deposit(100.0);

        boolean worked = account.withdraw(50.0);
        assertTrue(worked, "success withdraw");
        assertEquals(50.0, account.getBalance());
    }

    @Test
    public void testFailedWithdrawal() {
        BankAccount account = new BankAccount();
        account.deposit(100.0);
        account.withdraw(50.0);

        boolean fail = account.withdraw(60.0);
        assertFalse(fail, "should fail");
        assertEquals(50.0, account.getBalance());
    }
}

class BankAccount {
    private double balance = 0.0;

    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
        }
    }

    public boolean withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            return true;
        }
        return false;
    }

    public double getBalance() {
        return balance;
    }
}
```

```
~

╷
├─ JUnit Jupiter ✔
│  └─ BankAccountTest ✔
│     ├─ testWithdraw() ✔
│     ├─ testFailedWithdrawal() ✔
│     └─ testDeposit() ✔
└─ JUnit Vintage ✔

Test run finished after 99 ms
[         3 containers found      ]
[         0 containers skipped    ]
[         3 containers started    ]
[         0 containers aborted    ]
[         3 containers successful ]
[         0 containers failed     ]
[         3 tests found           ]
[         0 tests skipped         ]
[         3 tests started         ]
[         0 tests aborted         ]
[         3 tests successful      ]
[         0 tests failed          ]
```

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]

```
~

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import java.util.ArrayList;
import java.util.List;

public class NumberUtils {
    public static int[] getEvenNumbers(int[] numbers) {
        List<Integer> evens = new ArrayList<>();
        for (int number : numbers) {
            if (number % 2 == 0) {
                evens.add(number);
            }
        }
        int[] result = new int[evens.size()];
        for (int i = 0; i < evens.size(); i++) {
            result[i] = evens.get(i);
        }
        return result;
    }

    @Test
    public void testGetEvenNumbers() {
        int[] input = {1, 2, 3, 4, 5, 6};
        int[]  want = {2, 4, 6};
        assertArrayEquals(want, getEvenNumbers(input));
    }
}
```

```
~

Thanks for using JUnit! Support its development at https://junit.org/sponsoring

╷
├─ JUnit Jupiter ✔
│  └─ NumberUtils ✔
│     └─ testGetEvenNumbers() ✔
└─ JUnit Vintage ✔

Test run finished after 93 ms
[         3 containers found      ]
[         0 containers skipped    ]
[         3 containers started    ]
[         0 containers aborted    ]
[         3 containers successful ]
[         0 containers failed     ]
[         1 tests found           ]
[         0 tests skipped         ]
[         1 tests started         ]
[         0 tests aborted         ]
[         1 tests successful      ]
[         0 tests failed          ]
```

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.

```
~

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

public class ProductTest {
    @Test
    public void testProduct() {
        Product product = new Product("Laptop", 1000, 3);
        assertAll(
            "Product Assertions",
            () -> assertNotNull(product.getName()),
            () -> assertTrue(product.getPrice() > 0),
            () -> assertTrue(product.isAffordable(4000)),
        );
    }
}

class Product {
    String name;
    double price;
    int quantity;

    public Product(String name, double price, int quantity) {
        this.name = name;
        this.price = price;
        this.quantity = quantity;
    }

    public String getName() { return name; }
    public double getPrice() { return price; }
    public boolean isAffordable(double budget) { return (price * quantity) <= budget; }
}
```

```
~
|
├─ JUnit Jupiter ✔
|  └─ ProductTest ✔
|     └─ testProduct() ✔
└─ JUnit Vintage ✔

Test run finished after 93 ms
[         3 containers found      ]
[         0 containers skipped    ]
[         3 containers started    ]
[         0 containers aborted    ]
[         3 containers successful ]
[         0 containers failed     ]
[         1 tests found           ]
[         0 tests skipped         ]
[         1 tests started         ]
[         0 tests aborted         ]
[         1 tests successful      ]
[         0 tests failed          ]
```

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.

```
~

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

public class InventoryTest {
    @Test
    public void testIsProductAvailable() {
        assertAll(
            () -> assertTrue(Inventory.isProductAvailable("laptop", 5)),
            () -> assertFalse(Inventory.isProductAvailable("phone", 6)),
            () -> assertFalse(Inventory.isProductAvailable("Moniter", 1)),
            () -> assertTrue(Inventory.isProductAvailable("tablet", 8))
        );
    }
}
class Inventory {
    public static String[] productNames = {"laptop", "phone", "tablet"};
    public static int[] productQuantities = {10, 5, 8};

    public static boolean isProductAvailable(String productName, int quantity) {
        for (int i = 0; i < productNames.length; i++) {
            if (productNames[i].equals(productName)) {
                return productQuantities[i] >= quantity;
            }
        }
        return false;
    }
}
```

```
~

[wizard@archlinux 4]$ java-test InventoryTest

Thanks for using JUnit! Support its development at https://junit.org/sponsoring

╷
├─ JUnit Jupiter ✔
│  └─ InventoryTest ✔
│     └─ testIsProductAvailable() ✔
└─ JUnit Vintage ✔

Test run finished after 87 ms
[         3 containers found      ]
[         0 containers skipped    ]
[         3 containers started    ]
[         0 containers aborted    ]
[         3 containers successful ]
[         0 containers failed     ]
[         1 tests found           ]
[         0 tests skipped         ]
[         1 tests started         ]
[         0 tests aborted         ]
[         1 tests successful      ]
[         0 tests failed          ]
```

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.

```
~

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

public class NotificationServiceTest {
    @Test
    public void testSendEmail() {
        assertAll(
            () -> assertTrue(NotificationService.sendEmail("something@gmail.com", "work")),
            () -> assertFalse(NotificationService.sendEmail("not valid", "no work")),
            () -> assertFalse(NotificationService.sendEmail("!@gmail.com", "work"))
        );
    }
}
```

```
class NotificationService {
    public static boolean sendEmail(String email, String message) {
        if(email.matches("^[a-zA-Z0-9\\.]+@gmail\\.com$")){
            System.out.println("Sent Messagej" + message);
            return true;
        };
        return false;
    }
}
```

```
~

Thanks for using JUnit! Support its development at https://junit.org/sponsoring

╷
├─ JUnit Jupiter ✔
└─ JUnit Vintage ✔

Test run finished after 45 ms
[         2 containers found      ]
[         0 containers skipped    ]
[         2 containers started    ]
[         0 containers aborted    ]
[         2 containers successful ]
[         0 containers failed     ]
[         0 tests found           ]
[         0 tests skipped         ]
[         0 tests started         ]
[         0 tests aborted         ]
[         0 tests successful      ]
[         0 tests failed          ]
```

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.

```
~

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

public class EnrollmentServiceTest {
    @Test
    public void testEnrollStudent() {
        assertAll(
            () -> assertTrue(EnrollmentService.enrollStudent("someone", "java")),
            () -> assertFalse(EnrollmentService.enrollStudent("student1", "java"))
        );
    }
}

class EnrollmentService {
    static String[] enrolledStudents = {"student1:java", "student2:c"};

    public static boolean enrollStudent(String student, String courseName) {
        String enrollmentRecord = student + ":" + courseName;
        for (String record : enrolledStudents) {
            if (record.equals(enrollmentRecord)) {
                return false;
            }
        }

        System.out.println("Enrolled " + student);
        return true;
    }
}
```

```
~


Thanks for using JUnit! Support its development at https://junit.org/sponsoring

╷
├─ JUnit Jupiter ✔
└─ JUnit Vintage ✔

Test run finished after 42 ms
[         2 containers found      ]
[         0 containers skipped    ]
[         2 containers started    ]
[         0 containers aborted    ]
[         2 containers successful ]
[         0 containers failed     ]
[         0 tests found           ]
[         0 tests skipped         ]
[         0 tests started         ]
[         0 tests aborted         ]
[         0 tests successful      ]
[         0 tests failed          ]
```

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.

```
~

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

public class  StringTest {
    @Test
    public void testString() {
        assertAll(
            () -> assertEquals("tac", Stringmethods.reverse("cat")),
            () -> assertEquals("HELLO", Stringmethods.toUpperCase("hello")),
            () -> assertTrue(Stringmethods.isPalindrome("racecar")),
            () -> assertEquals(4, Stringmethods.countVowels("someone"))
        );
    }
}

class Stringmethods {
    public static String reverse(String input) {
        StringBuilder reverser = new StringBuilder(input);
        return reverser.reverse().toString();
    }

    public static String toUpperCase(String input) { return input.toUpperCase(); }

    public static boolean isPalindrome(String input) {
        String reversed = reverse(input);
        return input.equals(reversed);
    }

    public static int countVowels(String input) {
        int count = 0;
        for (char c : input.toLowerCase().toCharArray()) {
            if ("aeiou".indexOf(c) != -1) count++;
        }
        return count;
    }
}
```

```
~

Thanks for using JUnit! Support its development at https://junit.org/sponsoring

╷
├─ JUnit Jupiter ✔
│  └─ StringTest ✔
│     └─ testString() ✔
└─ JUnit Vintage ✔

Test run finished after 94 ms
[         3 containers found      ]
[         0 containers skipped    ]
[         3 containers started    ]
[         0 containers aborted    ]
[         3 containers successful ]
[         0 containers failed     ]
[         1 tests found           ]
[         0 tests skipped         ]
[         1 tests started         ]
[         0 tests aborted         ]
[         1 tests successful      ]
[         0 tests failed          ]
```

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).

```
~

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class CalculatorTest {
    private static Calculator calculator;

    @BeforeAll
    public static void beforeclass() {
        System.out.println("Some thing to run only once");
    }

    @AfterAll
    public static void afterclass() {
        calculator = null;
        System.out.println("something to clean up after each class");
    }

    @BeforeEach
    public void bef() { calculator = new Calculator(); }

    @AfterEach
    public void after() { System.out.println("Runned once"); }

    @Test
    public void testadd() { assertEquals(5, calculator.add(2, 3)); }

    @Test
    public void testsub() { assertEquals(1, calculator.subtract(3, 2)); }

    @Test
    public void testmul() { assertEquals(6, calculator.multiply(2, 3)); }
    @Test
    public void testdiv() { assertEquals(2, calculator.divide(6, 3)); }
}

class Calculator {
    public int add(int a, int b) { return a + b; }

    public int subtract(int a, int b) { return a - b; }

    public int multiply(int a, int b) { return a * b; }

    public int divide(int a, int b) { return a / b; }
}
```

```
~

[wizard@archlinux 4]$ java-test CalculatorTest
Some thing to run only once
Runned once
Runned once
Runned once
Runned once
something to clean up after each class

Thanks for using JUnit! Support its development at https://junit.org/sponsoring

.
├─ JUnit Jupiter ✔
│  └─ CalculatorTest ✔
│     ├─ testadd() ✔
│     ├─ testdiv() ✔
│     ├─ testmul() ✔
│     └─ testsub() ✔
└─ JUnit Vintage ✔

Test run finished after 110 ms
[         3 containers found      ]
[         0 containers skipped    ]
[         3 containers started    ]
[         0 containers aborted    ]
[         3 containers successful ]
[         0 containers failed     ]
[         4 tests found           ]
[         0 tests skipped         ]
[         4 tests started         ]
[         0 tests aborted         ]
[         4 tests successful      ]
[         0 tests failed          ]
```

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

```
~

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

public class TestPrime {

    @Test
    public void test() {
        assertTrue(Primecheck.isPrime(5));

        assertFalse(Primecheck.isPrime(10));
    }
}

class Primecheck {
    public static boolean isPrime(int n) {
        for (int i = 2; i <= Math.sqrt(n); i++) {
            if (n % i == 0) {
                return false;
            }
        }
        return true;
    }
}
```

```
~

╷
├─ JUnit Jupiter ✔
│  └─ TestPrime ✔
│     └─ test() ✔
└─ JUnit Vintage ✔

Test run finished after 85 ms
[         3 containers found      ]
[         0 containers skipped    ]
[         3 containers started    ]
[         0 containers aborted    ]
[         3 containers successful ]
[         0 containers failed     ]
[         1 tests found           ]
[         0 tests skipped         ]
[         1 tests started         ]
[         0 tests aborted         ]
[         1 tests successful      ]
[         0 tests failed          ]
```

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

```
~

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

public class TestFactorial {

    @Test
    public void testFactorial() {
        assertEquals(120, Factorial.calculateFactorial(5));
        assertEquals(24, Factorial.calculateFactorial(4));
    }
}

class Factorial {
    public static int calculateFactorial(int n) {
        int result = 1;
        for (int i = 1; i <= n; i++) {
            result *= i;
        }
        return result;
    }
}
```

```
~

╷
├─ JUnit Jupiter ✔
│  └─ TestFactorial ✔
│     └─ testFactorial() ✔
└─ JUnit Vintage ✔

Test run finished after 93 ms
[         3 containers found      ]
[         0 containers skipped    ]
[         3 containers started    ]
[         0 containers aborted    ]
[         3 containers successful ]
[         0 containers failed     ]
[         1 tests found           ]
[         0 tests skipped         ]
[         1 tests started         ]
[         0 tests aborted         ]
[         1 tests successful      ]
[         0 tests failed          ]
```

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

```
~

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

public class Testrectangle {

    @Test
    public void testRectangle() {
        Rectangle rectangle = new Rectangle(5, 3);
        assertEquals(15, rectangle.area());
        assertEquals(16, rectangle.perimeter());
    }
}

class Rectangle {
    private double length;
    private double width;

    public Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    public double area() { return length * width; }

    public double perimeter() { return 2 * (length + width); }
}
```

```
~

Thanks for using JUnit! Support its development at https://junit.org/sponsoring

┌─ JUnit Jupiter ✔
│  └─ Testrectangle ✔
│     └─ testRectangle() ✔
└─ JUnit Vintage ✔

Test run finished after 85 ms
[         3 containers found      ]
[         0 containers skipped    ]
[         3 containers started    ]
[         0 containers aborted    ]
[         3 containers successful ]
[         0 containers failed     ]
[         1 tests found           ]
[         0 tests skipped         ]
[         1 tests started         ]
[         0 tests aborted         ]
[         1 tests successful      ]
[         0 tests failed          ]
```

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

```
~
```

```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class TestShape {
    @Test
    public void testShape() {
        Shape circle = new Circle(5);
        assertEquals(Math.PI * 5 * 5, circle.area());
        Shape rectangle = new Rectangle(4, 6);
        assertEquals(24, rectangle.area());
    }
}

class Shape {
    public double area() { return 0; }
}

class Circle extends Shape {
    private double radius;
    public Circle(double radius) { this.radius = radius; }

    @Override
    public double area() { return Math.PI * radius * radius; }
}

class Rectangle extends Shape {
    private double length;
    private double width;

    public Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    @Override
    public double area() { return length * width; }
}
```

```
~
Thanks for using JUnit! Support its development at https://junit.org/sponsoring

╷
├─ JUnit Jupiter ✔
│  └─ TestShape ✔
│     └─ testShape() ✔
└─ JUnit Vintage ✔

Test run finished after 88 ms
[         3 containers found      ]
[         0 containers skipped    ]
[         3 containers started    ]
[         0 containers aborted    ]
[         3 containers successful ]
[         0 containers failed     ]
[         1 tests found           ]
[         0 tests skipped         ]
[         1 tests started         ]
[         0 tests aborted         ]
[         1 tests successful      ]
[         0 tests failed          ]
```