Ćwiczenie 1: Prosta Klasa Kalkulatora

1. Stwórz klasę **Calculator** zawierającą metody dodawania i odejmowania.
2. Napisz testy jednostkowe za pomocą JUnit 5 do sprawdzenia poprawności działania obu metod.

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

public class CalculatorTest {

    @Test
    void testAddition() {
        assertEquals(5, Calculator.add(2, 3));
    }

    @Test
    void testSubtraction() {
        assertEquals(2, Calculator.subtract(5, 3));
    }
}
```

Rozwiązanie:

```java
public class Calculator {

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

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

Ćwiczenie 2: Obsługa Błędów

1. Rozwiń klasę **Calculator**, dodając metodę dzielenia.
2. Zabezpiecz metodę dzielenia przed dzieleniem przez zero.
3. Napisz test jednostkowy, aby sprawdzić, czy metoda rzuci wyjątkiem przy próbie dzielenia przez zero.

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

public class CalculatorTest {

    @Test
    void testDivisionByZero() {
        assertThrows(ArithmeticException.class, () -> Calculator.divide(5, 0));
    }
}
```

Rozwiązanie:

```java
public class Calculator {

    public static int divide(int a, int b) {
        if (b == 0) {
            throw new ArithmeticException("Division by zero");
        }
        return a / b;
    }
}
```

Ćwiczenie 3: Parametryzowane Testy

1. Rozwiń klasę **StringUtils** z metodą **capitalize**, która zamienia pierwszą literę ciągu znaków na wielką.
2. Napisz parametryzowany test jednostkowy, aby przetestować różne przypadki użycia metody **capitalize**.

```java
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;
import static org.junit.jupiter.api.Assertions.*;

public class StringUtilsTest {

    @ParameterizedTest
    @CsvSource({"apple, Apple", "banana, Banana", "orange, Orange"})
    void testCapitalize(String input, String expected) {
        assertEquals(expected, StringUtils.capitalize(input));
    }
}
```

Rozwiązanie:

```java
public class StringUtils {

    public static String capitalize(String input) {
        if (input == null || input.isEmpty()) {
            return input;
        }
        return Character.toUpperCase(input.charAt(0)) + input.substring(1);
    }
}
```

Ćwiczenie 4: Kalkulator BMI

Treść ćwiczenia:
Stwórz klasę **BMICalculator**, która będzie zawierała metodę do obliczania wskaźnika masy ciała (BMI). BMI można obliczyć na podstawie wzoru: BMI = masa ciała (kg) / (wzrost (m) * wzrost (m)). Napisz testy jednostkowe przy użyciu JUnit 5 dla metody obliczającej BMI, sprawdzając różne przypadki, takie jak poprawne obliczenia, ujemne wartości, oraz testy dla wartości granicznych.

Rozwiązanie

```
public class BMICalculator {

    public double calculateBMI(double weight, double height) {
        if (weight <= 0 || height <= 0) {
```

```
            throw new IllegalArgumentException("Weight and height
must be positive values");
        }
        return weight / (height * height);
    }
}

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

public class BMICalculatorTest {

    @Test
    void testCalculateBMI() {
        BMICalculator calculator = new BMICalculator();
        assertEquals(22.22, calculator.calculateBMI(70, 1.75), 0.01);
    }

    @Test
    void testCalculateBMIWithNegativeValues() {
        BMICalculator calculator = new BMICalculator();
        assertThrows(IllegalArgumentException.class,       ()       ->
calculator.calculateBMI(-70, 1.75));
    }

    @Test
    void testCalculateBMIWithZeroValues() {
        BMICalculator calculator = new BMICalculator();
        assertThrows(IllegalArgumentException.class,       ()       ->
calculator.calculateBMI(70, 0));
    }
}
```

Ćwiczenie 5: Zarządzanie Listą Zakupów

Treść ćwiczenia:
Stwórz klasę **ShoppingList**, która pozwala dodawać, usuwać i zwracać produkty na liście zakupów. Napisz testy jednostkowe przy użyciu JUnit 5 do sprawdzenia poprawności funkcji dodawania, usuwania i zwracania produktów. Upewnij się, że testujesz różne scenariusze, takie jak próba usunięcia produktu, który nie istnieje, czy dodanie pustego produktu.

Rozwiązanie:

```
import java.util.ArrayList;
import java.util.List;

public class ShoppingList {
```

```
    private List<String> products = new ArrayList<>();

    public void addProduct(String product) {
        if (product != null && !product.trim().isEmpty()) {
            products.add(product);
        } else {
            throw new IllegalArgumentException("Product name cannot
be empty");
        }
    }

    public void removeProduct(String product) {
        if (!products.remove(product)) {
            throw new IllegalArgumentException("Product not found");
        }
    }

    public List<String> getProducts() {
        return new ArrayList<>(products);
    }
}
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class ShoppingListTest {

    @Test
    void testAddProduct() {
        ShoppingList shoppingList = new ShoppingList();
        shoppingList.addProduct("Apple");
        shoppingList.addProduct("Banana");
        assertEquals(2, shoppingList.getProducts().size());
    }

    @Test
    void testAddEmptyProduct() {
        ShoppingList shoppingList = new ShoppingList();
        assertThrows(IllegalArgumentException.class,      ()      ->
shoppingList.addProduct(""));
    }

    @Test
    void testRemoveProduct() {
        ShoppingList shoppingList = new ShoppingList();
        shoppingList.addProduct("Apple");
        shoppingList.addProduct("Banana");
        shoppingList.removeProduct("Banana");
```

```
        assertEquals(1, shoppingList.getProducts().size());
    }

    @Test
    void testRemoveNonexistentProduct() {
        ShoppingList shoppingList = new ShoppingList();
        assertThrows(IllegalArgumentException.class,       ()       ->
shoppingList.removeProduct("Orange"));
    }
}
```

Ćwiczenie 6: Obsługa Książek w Bibliotece

Treść ćwiczenia:
Stwórz klasę **Library**, która będzie zawierała książki i pozwalała na dodawanie, usuwanie i wypożyczanie książek. Napisz testy jednostkowe przy użyciu JUnit 5 do sprawdzenia poprawności funkcji zarządzania książkami. Sprawdź różne scenariusze, takie jak próba wypożyczenia nieistniejącej książki czy dodanie pustej książki.

Rozwiązanie:

```
import java.util.ArrayList;
import java.util.List;

public class Library {

    private List<Book> books = new ArrayList<>();

    public void addBook(Book book) {
        if (book != null && !book.getTitle().trim().isEmpty()) {
            books.add(book);
        } else {
            throw new IllegalArgumentException("Book title cannot be
empty");
        }
    }

    public void removeBook(Book book) {
        if (!books.remove(book)) {
            throw new IllegalArgumentException("Book not found");
        }
    }

    public void borrowBook(Book book) {
        if (!books.contains(book)) {
            throw new IllegalArgumentException("Book not found");
        }
        if (!book.isAvailable()) {
```

```
            throw new IllegalStateException("Book is already
borrowed");
        }
        book.setAvailable(false);
    }

    public List<Book> getBooks() {
        return new ArrayList<>(books);
    }
}
public class Book {

    private String title;
    private boolean available;

    public Book(String title) {
        this.title = title;
        this.available = true;
    }

    public String getTitle() {
        return title;
    }

    public boolean isAvailable() {
        return available;
    }

    public void setAvailable(boolean available) {
        this.available = available;
    }
}
```