Folder src/test/java

2 printable files

(file list disabled)

src/test/java/MatrixInitTest.java

```java
/**
 * @author Aubry Antoine
 * @author Faria dos Santos Dani Tiago
 */

import ch.heigvd.dai.Matrix;
import org.junit.jupiter.api.Test;

import java.io.ByteArrayInputStream;
import java.util.Scanner;

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


public class MatrixInitTest {

    public static Matrix[] createMatrices(Scanner scanner) {
        try {
            System.out.print("Entrez le nombre de lignes pour la première matrice: ");
            int nbRow1 = scanner.nextInt();
            System.out.print("Entrez le nombre de colonnes pour la première matrice: ");
            int nbCol1 = scanner.nextInt();
            System.out.print("Entrez le nombre de lignes pour la deuxième matrice: ");
            int nbRow2 = scanner.nextInt();
            System.out.print("Entrez le nombre de colonnes pour la deuxième matrice: ");
            int nbCol2 = scanner.nextInt();

            if (nbRow1 <= 0 || nbCol1 <= 0 || nbRow2 <= 0 || nbCol2 <= 0) {
                throw new RuntimeException("Les dimensions entrées ne sont pas des nombres positifs.");
            }

            System.out.print("Entrez une valeur pour le modulo de la première matrice: ");
            int modulo1 = scanner.nextInt();
            System.out.print("Entrez une valeur pour le modulo de la deuxième matrice: ");
            int modulo2 = scanner.nextInt();

            if (modulo1 <= 0 || modulo2 <= 0) {
                throw new RuntimeException("L'un des modulos entrés n'est pas un nombre positif.");
            }

            if (modulo1 != modulo2) {
                throw new RuntimeException("Les modulos sont différents.");
            }

            Matrix m1 = new Matrix(nbRow1, nbCol1, modulo1);
            Matrix m2 = new Matrix(nbRow2, nbCol2, modulo2);

            System.out.print("Entrer les valeurs manuellement (oui/non) ? ");
            String choice = scanner.next();

            if (choice.equalsIgnoreCase("oui")) {
                m1.randomMatrixInitiation(false, scanner);
                m2.randomMatrixInitiation(false, scanner);
            } else if (choice.equalsIgnoreCase("non")) {
                m1.randomMatrixInitiation(true, scanner);
                m2.randomMatrixInitiation(true, scanner);
```

```
            } else {
                throw new RuntimeException("Réponse incorrecte.");
            }
            return new Matrix[]{m1, m2};

        } catch (Exception e) {
            throw new RuntimeException("Erreur lors de la création des matrices : " + e.getMessage());
        }

    }
    @Test
    public void testCreateMatricesWithValidInput() {
        String dimMat1 = "2\n3\n";
        String dimMat2 = "2\n4\n";
        String moduloMat1= "10\n";
        String moduloMat2= "10\n";
        String manuelValues = "oui\n";
        String valuesMat1 = "9\n4\n7\n2\n8\n1\n";
        String valuesMat2 = "3\n1\n2\n5\n3\n8\n4\n9\n";
        String input = dimMat1+dimMat2+moduloMat1+moduloMat2+manuelValues+valuesMat1+valuesMat2;
        Scanner scanner = new Scanner(new ByteArrayInputStream(input.getBytes()));

        Matrix[] matrices = createMatrices(scanner);

        assertNotNull(matrices);
        assertEquals(2, matrices.length, "correct");
        assertEquals(10, matrices[0].getModulo());
        assertEquals(10, matrices[1].getModulo());
        assertEquals(9, matrices[0].getValue(0, 0));
    }

    @Test
    public void testCreateMatricesWithInvalidDimensions() {
        String input = "-1\n2\n2\n2\n10\n10\nnon\n";
        Scanner scanner = new Scanner(new ByteArrayInputStream(input.getBytes()));

        assertThrows(RuntimeException.class, () -> createMatrices(scanner));
    }

    @Test
    public void testCreateMatricesWithDifferentModulos() {
        String input = "2\n2\n2\n2\n10\n5\nnon\n";
        Scanner scanner = new Scanner(new ByteArrayInputStream(input.getBytes()));

        assertThrows(RuntimeException.class, () -> createMatrices(scanner));
    }
}
```

src/test/java/MatrixTest.java

```
/**
 * @author Aubry Antoine
 * @author Faria dos Santos Dani Tiago
 */

//package ch.heigvd.dai;

import ch.heigvd.dai.*;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

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

public class MatrixTest {
```

```
private Matrix matrix1;
private Matrix matrix2;
private Matrix result;

@BeforeEach
public void setUp() {
    matrix1 = new Matrix(2, 2, 10);
    matrix2 = new Matrix(2, 2, 11);

    matrix1.setValue(0, 0, 1);
    matrix1.setValue(0, 1, 2);
    matrix1.setValue(1, 0, 3);
    matrix1.setValue(1, 1, 4);
    matrix2.setValue(0, 0, 5);
    matrix2.setValue(0, 1, 6);
    matrix2.setValue(1, 0, 7);
    matrix2.setValue(1, 1, 8);
}

@Test
public void testAdditionOperation() {
    MatrixOperation addition = new Addition();
    result = matrix1.operate(matrix2, addition);

    assertEquals(6, result.getValue(0, 0), "1 + 5 [10] should equal 6");
    assertEquals(8, result.getValue(0, 1), "2 + 6 [10] should equal 8");
    assertEquals(0, result.getValue(1, 0), "3 + 7 [10] should equal 10");
    assertEquals(2, result.getValue(1, 1), "4 + 8 [10] should equal 12");
}

@Test
public void testSubtractionOperation() {
    MatrixOperation subtraction = new Subtraction();
    result = matrix1.operate(matrix2, subtraction);

    assertEquals(6, result.getValue(0, 0), "1 - 5 should equal -4");
    assertEquals(6, result.getValue(0, 1), "2 - 6 should equal -4");
    assertEquals(6, result.getValue(1, 0), "3 - 7 should equal -4");
    assertEquals(6, result.getValue(1, 1), "4 - 8 should equal -4");
}

@Test
public void testMultiplicationOperation() {
    MatrixOperation multiplication = new Multiplication();
    result = matrix1.operate(matrix2, multiplication);

    assertEquals(5, result.getValue(0, 0), "1 * 5 [10] should equal 5");
    assertEquals(2, result.getValue(0, 1), "2 * 6 [10] should equal 2");
    assertEquals(1, result.getValue(1, 0), "3 * 7 [10] should equal 21");
    assertEquals(2, result.getValue(1, 1), "4 * 8 [10] should equal 32");
}

@Test
public void testMatrixInitialization() {
    assertNotNull(matrix1, "Matrix 1 should be initialized");
    assertNotNull(matrix2, "Matrix 2 should be initialized");
}

@Test
public void testValueSettingAndGetting() {
    assertEquals(1, matrix1.getValue(0, 0), "Value at (0, 0) should be 1");
    assertEquals(4, matrix1.getValue(1, 1), "Value at (1, 1) should be 4");
    matrix1.setValue(0, 0, 13);
    assertEquals(3, matrix1.getValue(0, 0), "Value at (0, 0) should now be 10");
}
```

```
    @Test
    public void testDisplay() {
        // While you can't assert the output of a print directly, you can test that display doesn't throw an
error
        assertDoesNotThrow(() -> matrix1.display(), "Display method should not throw an exception");
    }
}
```