Modular Programming Design:

The program breaks down into 5 classes, SparseMatrix.java, Vector.java, Jacobi.java, Test.java, and Main.java.

The first two classes are designed for the purpose of isolation between different data structures. These two classes also implemented the basic operations for their own data structures, including matrix's addition/subtraction, multiplication, element-retrieving, element's value-setting, and Jacobi-Decomposition in SparseMatrix.java, and vector's addition/subtraction in Vector.java.

Jacobi.java uses the matrixes generated from SparseMatrix.java as input, and implemented the iterative steps for solving Ax = b. The functions in SparseMatrix.java are designed to maximize its reusability in the future, by generalizing the function's interfaces as much as possible.

Test.java is discussed below.

Main.java serves as the main entry for the entire program, and calls testing functions to run the tests.

Testing Design:

Test.java class tests all functions in SparseMatrix.java and Vector.java (except for their class constructors, because constructors only simply assign values to the class variants, and the errors generated by assigning values are more likely coming from wrong input data instead of wrong constructor implementations, therefore constructors' tests are neglected), and also the convergence property for Jacobi.java.

The outline for testing:

Start with the most basic and independent functions (retrieveElement(), matrixSetter(), Vector.add() for instance), and then move on to the functions that have more dependencies on other functions. Therefore, before testing with Jacobi-solver, the other functions have to be tested firstly to provide a verified foundation for Jacob-solver's convergence testing.

The methods for testing:

All SparseMatrix and Vector testing are based on **Wilkinson Principle**, which is to check whether two methods generate the same outputs. The basic and simpler functions (retrieveElement(), matrixSetter(), Vector.add() for instance) are tested with small in-code testing data, and they are white-box tested, because smaller and less dependent functions are more robust and don't need a complex test for verification; The more complex functions are tested with Python generated large data inputs (having the same rank as mat1), and they are black-box tested, because if they are capable of providing foundations for the ultimate Jacobi-Iterative-Solver, which uses mat1(a matrix without knowing the ground truth) as input, they have to be able to correctly solve a black-boxed matrix that's as large as the mat1, to ensure ultimate Jacobi-Solver test is based upon functions that's robust enough for mat1's size.

The Jacobi-Iterative-Solver function is tested by looking at its **convergence property**, which is to check whether the ||Ax - b|| / ||b|| decrease with iterations.

The testing procedures:

Each function is tested on both of its **input** (whether the inputs satisfy the function's invariants, for instance, for retrieving element(i, j) from a matrix with rank of 5, (0 <= i, j < 5) has to be satisfied. When the function invariants are violated, the tests make sure the functions would throw assertion errors to terminate the program to prevent further error cascading), and **output correctness** (whether this function generated desired correct outcome, by comparing its result with a result generated by another verified method, if their second norm difference is zero, then this function passed the test)