```java
import java.util.ArrayList;

/**
 * COMP215-Programming Project 3: 01 Knapsack Problem Analysis.
 * KNAPSACKMAIN is the main method for testing 3 algorithmic solutions to the 01 Knapsack
 Problem.
 * @author Andrew Parsons
 * @version 09 March 2017
 */
public class KnapsackMain {

    static boolean debug = true;
    static int capacity = 30;     // knapsack capacity
    static MultiFileWriter multiFileWriter = new MultiFileWriter();
    static QuickSort quickSort;

    public static void main(String args[]) {

        /** --- SET VALUES HERE ------------------------------------------------------------ */
        int maxValue = 120;    // this sets the largest element value in an array
        int repetitions = 1;   // this sets the number of repetitions for the mean calculation
        int startArraySize = 0;    // this sets the starting array's size
        int endArraySize = 20;   // this sets the ending array's size
        int increment = 1;  // this sets the increment size
        /** -------------------------------------------------------------------------------- */

        // instantiate the necessary objects: a data generator and the sorter
        DoublesDataGenerator doublesDataGenerator = new DoublesDataGenerator();
        quickSort = new QuickSort();

        GreedySolution greedySolution = new GreedySolution();
        DynamicSolution dynamicSolution = new DynamicSolution();
        BruteforceSolution bruteforceSolution = new BruteforceSolution();

        for (int dataSize = startArraySize; dataSize < (endArraySize + increment); dataSize =
        dataSize + increment) {

            for (int r = 0; r < repetitions; r++) {

                double[] valuesArray = doublesDataGenerator.createDataSetOfDoubles(dataSize,
                maxValue);
                double[] weightsArray = doublesDataGenerator.createDataSetOfDoubles(dataSize,
                maxValue);

                System.out.println("Length: " + valuesArray.length);
                System.out.println("-----------------------------------------------");

                AlgorithmTester algorithmTester = new AlgorithmTester(greedySolution);
                algorithmTester.testSolution(valuesArray, weightsArray, repetitions);

                algorithmTester = new AlgorithmTester(dynamicSolution);
                algorithmTester.testSolution(valuesArray, weightsArray, repetitions);


                if (dataSize < 31) {
                    algorithmTester = new AlgorithmTester(bruteforceSolution);
                    algorithmTester.testSolution(valuesArray, weightsArray, repetitions);
                }
            }
            System.out.println();
        }
    }

    /**
```

```
     * Calculates the mean time to perform an operation.
     * @param arrayOfTimes, an array of times.
     * @return long, the mean time listed in the parametrized array.
     */
    private static long calculateMean(ArrayList<Long> arrayOfTimes) {

        long sum = 0;
        long size = arrayOfTimes.size();

        if (arrayOfTimes.isEmpty())
            return 0;

        for (long time: arrayOfTimes) {
            sum += time;
        }
        return sum / size;
    }
    //GreedySolution greedySolution = new GreedySolution();
    //DynamicSolution dynamicSolution = new DynamicSolution();
            /*ArrayList<Long> timeList = new ArrayList<>();
            for (int r = 0; r < repetitions; r++) {
                stopwatch = new Stopwatch();
                DynamicSolution dynamicSolutionMAIN = new DynamicSolution();
                System.out.println("Dynamic: " + dynamicSolutionMAIN.knapsack(valuesArray,
                weightsArray, capacity));
                timeList.add(stopwatch.elapsedTime());
            }
            System.out.println(calculateMean(timeList)/1E6);
            timeList = new ArrayList<>();
            for (int r = 0; r < repetitions; r++) {
                stopwatch = new Stopwatch();
                GreedySolution greedySolutionMAIN = new GreedySolution();
                System.out.println("GreedyP: " + greedySolutionMAIN.knapsack(valuesArray,
                weightsArray, capacity));
                timeList.add(stopwatch.elapsedTime());
            }
            System.out.println(calculateMean(timeList)/1E6);
            timeList = new ArrayList<>();
            System.out.println();*/
}
```