```java
/**
 * COMP215-Programming Project 3: 01 Knapsack Problem Analysis.
 * BRUTEFORCESOLUTION is the Java implementation of the bruteforce algorithm to solve the 01
 Knapsack Problem.
 * This implementation is adapted from http://www.micsymposium.org/mics_2005/papers/paper102.pdf
 * @author Andrew Parsons
 * @version 09 March 2017
 */
class BruteforceSolution implements Solution {

    /* --- INSTANCE VARIABLES --- */
    private double[] value;
    private double[] weight;

    /* --- METHODS --- */
    /**
     * KNAPSACK returns the solution to the 01 Knapsack Problem, using a bruteforce algorithm.
     * @param valuePass, the passed values
     * @param weightPass, the passed weights
     * @param capacity, the knapsack capacity
     * @return double, the best value taken
     */
    public double knapsack(double[] valuePass, double[] weightPass, double capacity) {

        if (valuePass.length == 0) { return 0; }
        value = valuePass;
        weight = weightPass;
        double[] subsets = new double[value.length];
        double takenValue = 0;
        double takenWeight = 0;

        // look at each subset of the entire set
        for (int i = 0; i < Math.pow(2, value.length); i++) {

            /* INVARIANT: the value/weight values are greater than 0 */
            if (KnapsackMain.debug)
                assert assertionValueWeightGreaterThanZero(takenValue, takenWeight);

            // variables for the inner loop
            int itemToTake = value.length - 1;
            double temporaryValue = 0;
            double temporaryWeight = 0;

            // wh
            while (subsets[itemToTake] != 0 && itemToTake > 0) {
                subsets[itemToTake] = 0;
                itemToTake = itemToTake - 1;
            }

            subsets[itemToTake] = 1;

            for (int j = 0; j < value.length; j++) {

                // if the item is marked in the subset, then add it to the takenValue
                if (subsets[j] == 1 ) {
                    temporaryValue = temporaryValue + value[j];
                    temporaryWeight = temporaryWeight + weight[j];
                }
            }

            if (temporaryValue > takenValue && temporaryWeight <= capacity) {
                takenValue = temporaryValue;
                takenWeight = temporaryWeight;
            }
        }
        return takenValue;
    }

    private boolean assertionValueWeightGreaterThanZero(double bestValueYet, double
    bestWeightYet) {

        return bestValueYet >= 0 && bestWeightYet >= 0;
    }
}
```