```java
/**
 * COMP215-Programming Project 3: 01 Knapsack Problem Analysis.
 * DYNAMICSOLUTION is the Java implementation of the dynamic programming solution to solve the
 01 Knapsack Problem.
 * This implementation is adapted from Sahni et al.
 * @author Andrew Parsons
 * @version 09 March 2017
 */
class DynamicSolution implements Solution {

    /* --- INSTANCE VARIABLES --- */
    static double[] value;
    static double[] weight;
    static int numberOfObjects;

    /* --- METHODS --- */

    /**
     * KNAPSACK returns the solution to the 01 Knapsack Problem, using a dynamic 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) {

        value = valuePass;
        weight = weightPass;
        numberOfObjects = value.length-1;

        if (value.length == 0) {
            return 0;
        }
        return helper(0, capacity);
    }

    /**
     *HELPER is a helper method for the dynamic algorithm's solution. It is recursively called.
     * @param i, recursion initiator
     * @param capacity, the capacity
     * @return double, the Knapsack solution
     */
    private static double helper(int i, double capacity) {

        if (i == numberOfObjects) {
            return (capacity < weight[numberOfObjects]) ? 0 : value[numberOfObjects];
        }

        if (capacity < weight[i]) {
            return helper(i+1, capacity);
        }

        double r = Math.max(helper(i+1, capacity), helper(i+1, capacity-weight[i]) + value[i]);

        if (KnapsackMain.debug)
            assert assertionFunctioning(r, capacity, i);

        return r;
    }

    private static boolean assertionFunctioning(double r, double capacity, int i) {

        double first = helper(i+1, capacity);
        double second = helper(i+1, capacity-weight[i]) + value[i];
```

```
        double checkedMaximum = first;
        if (checkedMaximum < second)
            checkedMaximum = second;

        if (checkedMaximum == r)
            return true;
        return false;

    }

}
```