```
/**
 * COMP215-Programming Project 3: 01 Knapsack Problem Analysis.
 * GREEDYSOLUTION is the Java implementation of the greedy algorithm to solve the 01 Knapsack
 Problem.
 * @author Andrew Parsons
 * @version 09 March 2017
 */
class GreedySolution implements Solution {

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

    /* --- METHODS -- */

    /**
     * KNAPSACK returns the solution to the 01 Knapsack Problem, using a greedy 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 valueTaken = 0;
        KnapsackItem[] knapsackItems = new KnapsackItem[value.length];
        KnapsackItem[] takenItemsArray = new KnapsackItem[0];

        for (int v = 0; v < value.length; v++) {
            knapsackItems[v] = new KnapsackItem(value[v], weight[v]);
        }

        KnapsackMain.quickSort.sort(knapsackItems, 0, knapsackItems.length - 1);
        double remainingCapacity = capacity;

        if (KnapsackMain.debug)
            takenItemsArray = new KnapsackItem[value.length];

        /* INVARIANT: (1) still under capacity, (2) items taken have greater density than
        untaken items */

        for (int n = knapsackItems.length - 1; n >= 0; n--) {

            /*INIT: takenItem[n]'s density > knapsackItems[all other ones] 's density */
            if (KnapsackMain.debug)
                assert assertionCapacityAndDensityCheck(remainingCapacity, capacity,
                knapsackItems, takenItemsArray, n);

            if (knapsackItems[n].getWeight() <= remainingCapacity) {

                if (KnapsackMain.debug)
                    takenItemsArray[n] = knapsackItems[n];
                valueTaken = valueTaken + knapsackItems[n].getValue();
                remainingCapacity = remainingCapacity - (knapsackItems[n].getWeight());
            }

            /*MAIN: takenItem[n]'s density > knapsackItems[all other ones] 's density */
            if (KnapsackMain.debug)
```

```
                assert assertionCapacityAndDensityCheck(remainingCapacity, capacity,
                knapsackItems, takenItemsArray, n);
        }

        /*TERM: takenItem[n]'s density > knapsackItems[all other ones] 's density */
        if (KnapsackMain.debug)
            assert assertionCapacityAndDensityCheck(remainingCapacity, capacity, knapsackItems,
            takenItemsArray, 0);

        return valueTaken;
    }

    private boolean assertionCapacityAndDensityCheck(double remainingCapacity, double
    totalCapacity, KnapsackItem[] knapsackItemsArray, KnapsackItem[] takenItemsArray, int n) {

        if (remainingCapacity > totalCapacity)
            return false;
        for (int a = 0; a < n; a++) {
            if (!(takenItemsArray[n].getPriceDensity() > knapsackItemsArray[a].getPriceDensity())
                    || !(knapsackItemsArray[a].getWeight() <= remainingCapacity)) {
                System.out.println("a" + a);
                return false;
            }
        } return true;
    }
}
```