IT 314 – Software Engineering

Lab 7

Akshat Jindal | 202201299

Knapsack Code

Program Inspection

Question 1. How many errors are there in the program? Mention the errors you have identified.
Uninitialized vector opt is one of the errors. We need to initialize opt with 0, the total profit in case of weight of the bag to be 0 or total items to be 0. Similarly, we need to initialize sol with F, since we won't take items in the above boundary conditions.

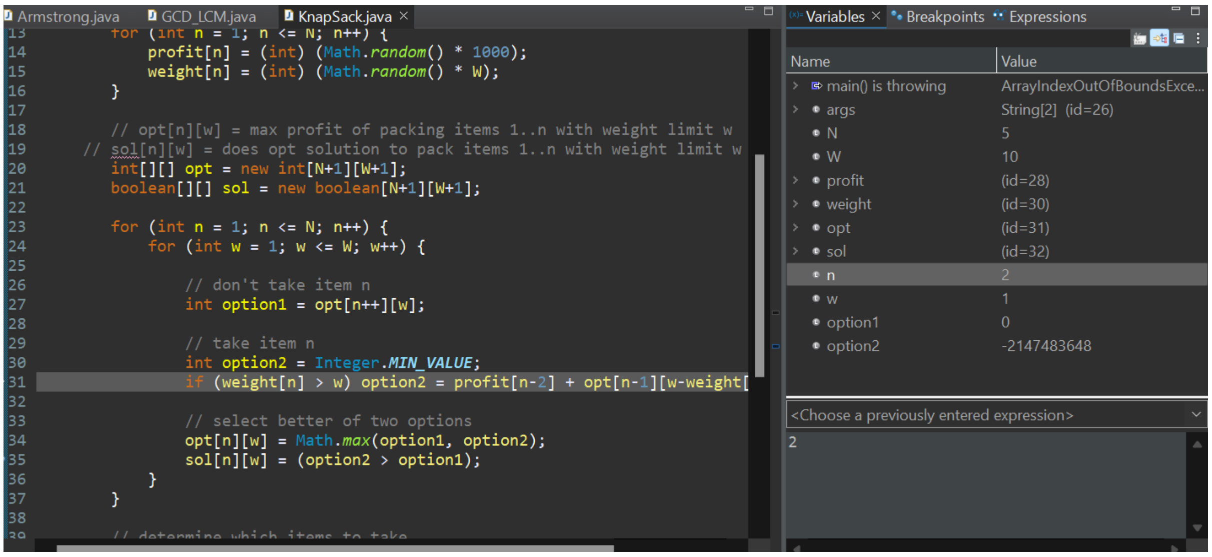
The next error is in the logic, with option1 being equal to opt[n++][w] instead of opt[n-1][w] and with option 2 being equal to profit[n-2] + opt[n-1][w-weight[n]] instead of profit[n] + opt[n-1][w-weight[n]]. Also, the condition for the option2 should be weight[n] <= w.

Question 2. Which category of program inspection would you find more effective?
For this question, the Category A of the program inspection, the Data Reference Errors were the most effective, as the failure of the code was due to the use of wrong referenced values and uninitialized values.

Question 3. Which type of error you are not able to identified using the program inspection?
In this example, all the errors in the document were identifiable using the program inspection method.

Question 4. Is the program inspection technique is worth applicable?
Even though the code was small, it was a bit cumbersome to find the errors for the failure in the program, with the program inspection method.

Code Debugging


Armstrong.java GCD_LCM.java KnapSack.java

```java
        for (int n = 1; n <= N; n++) {
            profit[n] = (int) (Math.random() * 1000);
            weight[n] = (int) (Math.random() * W);
        }

        // opt[n][w] = max profit of packing items 1..n with weight limit w
     // sol[n][w] = does opt solution to pack items 1..n with weight limit w
        int[][] opt = new int[N+1][W+1];
        boolean[][] sol = new boolean[N+1][W+1];

        for (int n = 1; n <= N; n++) {
            for (int w = 1; w <= W; w++) {

                // don't take item n
                int option1 = opt[n++][w];

                // take item n
                int option2 = Integer.MIN_VALUE;
                if (weight[n] > w) option2 = profit[n-2] + opt[n-1][w-weight[

                // select better of two options
                opt[n][w] = Math.max(option1, option2);
                sol[n][w] = (option2 > option1);
            }
        }

        // determine which items to take
```

Variables Breakpoints Expressions

Name	Value
main() is throwing	ArrayIndexOutOfBoundsExce...
args	String[2] (id=26)
N	5
W	10
profit	(id=28)
weight	(id=30)
opt	(id=31)
sol	(id=32)
n	2
w	1
option1	0
option2	-2147483648

<Choose a previously entered expression>

2

Question 1. How many errors are there in the program? Mention the errors you have identified.

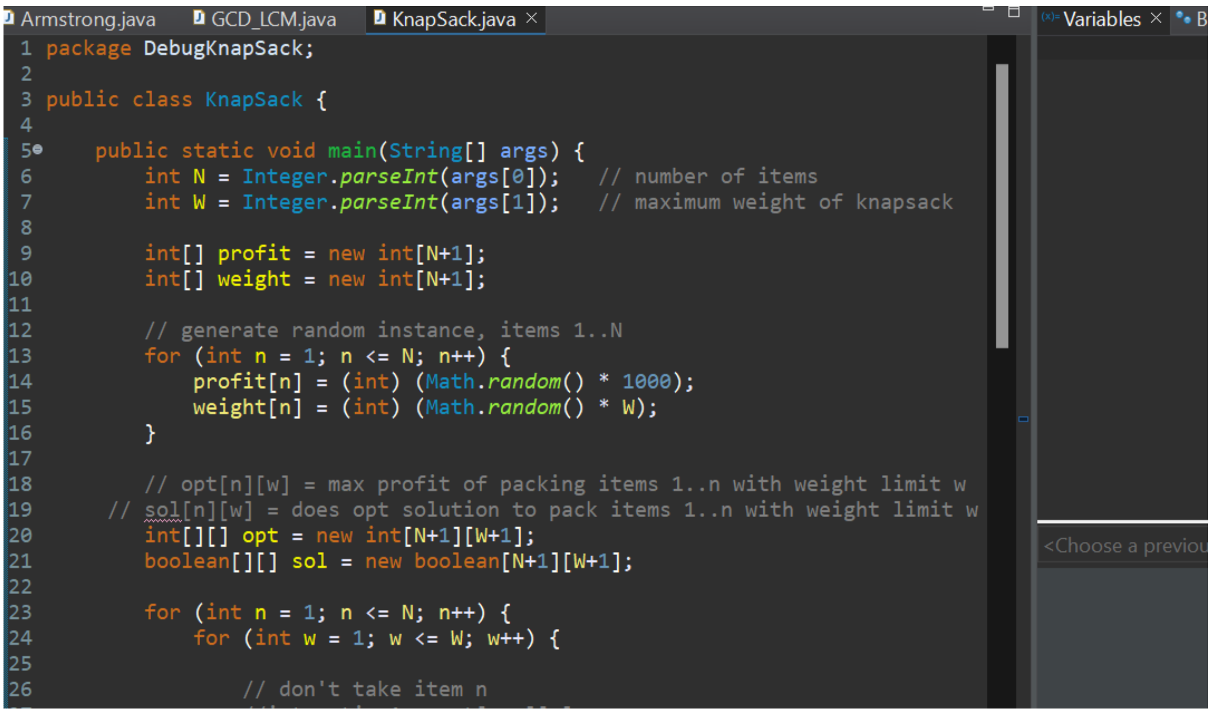
The errors were mostly the uninitialized 2d arrays opt and sol. Though they do not give an error while running in Java and are initialized correctly, it's a good practice to initialize these arrays. The next mistake was with the logic, in calculation of both option1 and option2, with the conditions attached.

Question 2. How many breakpoints you need to fix those errors? What are the steps you have taken to fix the error you identified in the code fragment?

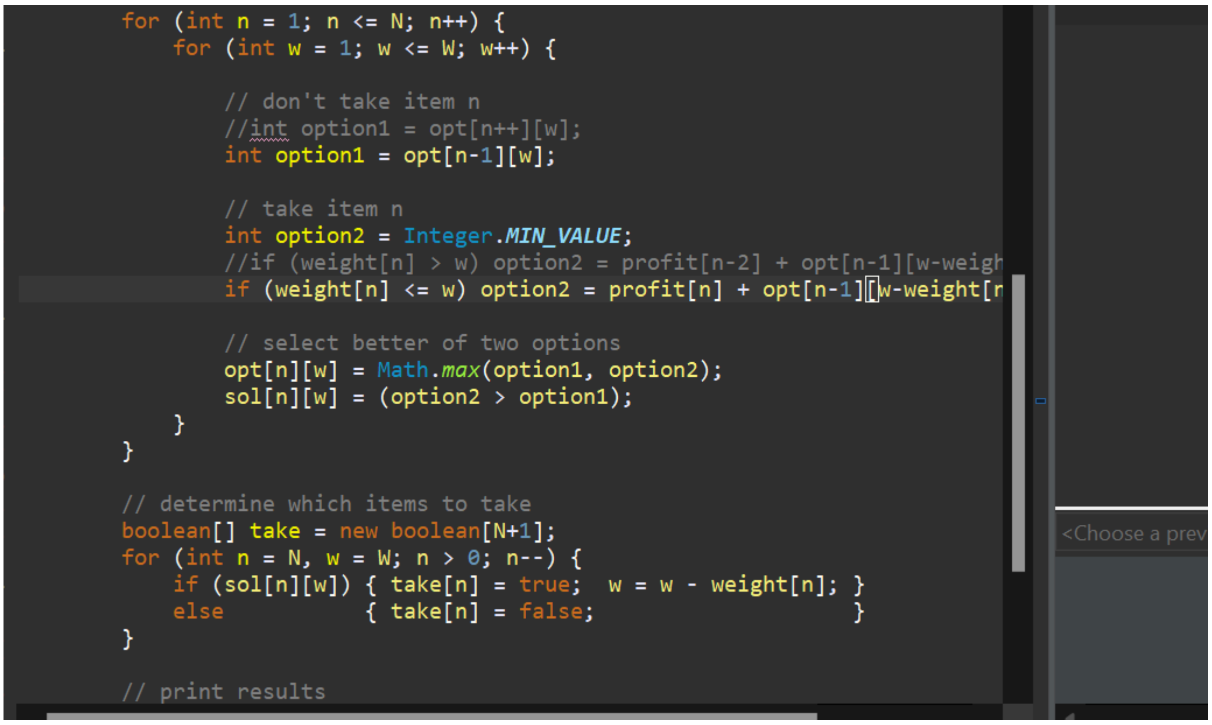
The breakpoints were added inside the double for loop, and were added each time to check for 2 to 3 iterations. The mistakes were then easily identified.

These errors can be fixed by simply initializing the 2d arrays and correcting the logic for calculation of knapsack.

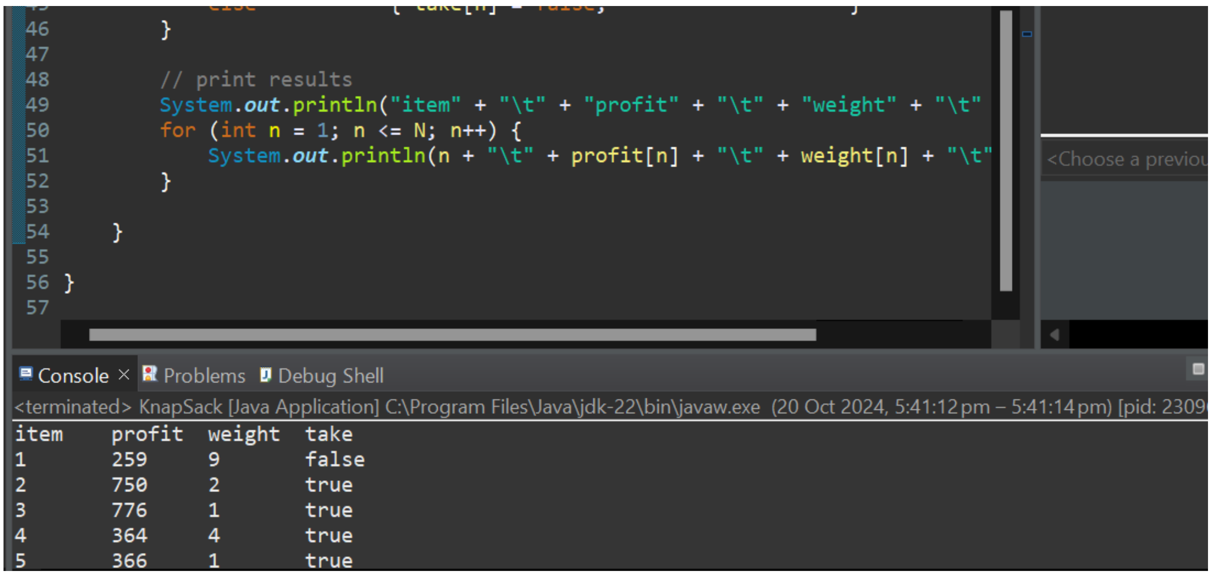
Question 3. Submit your complete executable code.



```java
package DebugKnapSack;

public class KnapSack {

    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);   // number of items
        int W = Integer.parseInt(args[1]);   // maximum weight of knapsack

        int[] profit = new int[N+1];
        int[] weight = new int[N+1];

        // generate random instance, items 1..N
        for (int n = 1; n <= N; n++) {
            profit[n] = (int) (Math.random() * 1000);
            weight[n] = (int) (Math.random() * W);
        }

        // opt[n][w] = max profit of packing items 1..n with weight limit w
     // sol[n][w] = does opt solution to pack items 1..n with weight limit w
        int[][] opt = new int[N+1][W+1];
        boolean[][] sol = new boolean[N+1][W+1];

        for (int n = 1; n <= N; n++) {
            for (int w = 1; w <= W; w++) {

                // don't take item n
```



```java
        for (int n = 1; n <= N; n++) {
            for (int w = 1; w <= W; w++) {

                // don't take item n
                //int option1 = opt[n++][w];
                int option1 = opt[n-1][w];

                // take item n
                int option2 = Integer.MIN_VALUE;
                //if (weight[n] > w) option2 = profit[n-2] + opt[n-1][w-weigh
                if (weight[n] <= w) option2 = profit[n] + opt[n-1][w-weight[n

                // select better of two options
                opt[n][w] = Math.max(option1, option2);
                sol[n][w] = (option2 > option1);
            }
        }

        // determine which items to take
        boolean[] take = new boolean[N+1];
        for (int n = N, w = W; n > 0; n--) {
            if (sol[n][w]) { take[n] = true;  w = w - weight[n]; }
            else           { take[n] = false;                    }
        }

        // print results
```



```java
        }

        // print results
        System.out.println("item" + "\t" + "profit" + "\t" + "weight" + "\t"
        for (int n = 1; n <= N; n++) {
            System.out.println(n + "\t" + profit[n] + "\t" + weight[n] + "\t"
        }

    }

}
```

```
<terminated> KnapSack [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe  (20 Oct 2024, 5:41:12 pm – 5:41:14 pm) [pid: 2309
item    profit  weight  take
1       259     9       false
2       750     2       true
3       776     1       true
4       364     4       true
5       366     1       true
```