CSC 364 Homework #1 Instructor: Jeff Ward
Assigned: Monday, January 23, 2017 Due: 11:59pm, Monday, February 6
Covers: Dynamic Programming Worth 40 points

Multidimensional Knapsack Problem

The file Data5.txt, provided on Blackboard, contains a list of five prospective projects for the upcoming year for a particular company:

```
Project0 13 34 12
Project1 22 54 18
Project2 5 34 44
Project3 16 49 48
Project4 9 30 8
```

Each line in the file provides four pieces of information:

1) The name of the project;
2) The amount of employee labor that will be demanded by the project, measured in work weeks;
3) The amount of company liquidity required by the project, measured in thousands of dollars;
4) The net profit that the company can expect from engaging in the project, measure in thousands of dollars.

Your task is to write a program that (1) prompts the user for the number of work weeks available (an integer); (2) prompts the user for the amount of liquidity available (also an integer), (3) prompts the user for the names of the input and output files; (4) reads the available projects from the input file; (5) solves the corresponding knapsack problem, without repetition of items; and (6) writes to the output file a summary of the results, including the expected profit and a list of the best projects for the company to undertake.

Here is a sample session with the program:

```
Enter the number of employee work weeks available: 30
Enter the amount of liquidity available: 80
Enter the name of input file: Data5.txt
Enter the name of output file: Output5.txt
```

For the above example, here is the output that should be written to Output5.txt:

```
Number of projects available: 5
Employee work weeks available: 30
Liquidity available: 80
Number of projects chosen: 2
Total profit: 56
Project0 13 34 12
Project2 5 34 44
```

The file Data1000.txt, also provided on Blackboard, contains one thousand prospective projects. Your program should also be able to handle this larger problem as well. The corresponding output file (WardOutput1000.txt) is also on Blackboard (work weeks available = 40, liquidity available = 100).

With a thousand prospective projects to consider, it will be impossible for your program to finish in a reasonable amount of time if it uses a "brute-force search" that explicitly considers every possible combination of projects. You are required to use a dynamic programming approach to this problem as described in the Chapter22/DynamicProgramming – HW1.ppt notes on Blackboard. However, note that the knapsack problem discussed in those notes is single-dimensional: there is only one weight constraint considered. In your problem there are two weight constraints: work weeks available and liquidity available. Thus, rather than the two-dimensional array (corresponding to the two parameter function K) shown in the notes, you will want to use a three-dimensional array. Give your array a more descriptive name than 'K', and use the Java naming conventions when naming it: Something like `profit`, `totalProfit`, or `payoff` would be a good name for it.

What to turn in:
Submit your .java file on Blackboard. Be sure to follow the coding conventions described below. (Students often lose points for neglecting the "Required Comments" and "Naming Conventions", especially).

CODING CONVENTIONS (IMPORTANT):
For this homework, and each subsequent homework, you are required to follow the coding conventions listed below. Failure to follow these conventions in full may result in a loss of up to 20% of your grade on the assignment, even if your code works perfectly. These conventions appears in the textbook and will be in force for this assignment and for all subsequent assignments in the course.

Required comments:
Each file should have comments at the beginning that state your name, the course number, and a brief description of what the program does. Of course, you are always free to provide additional comments that you feel make the program code easier to understand.

Indentation and spacing:
As stated in Section 1.9.2: "Indent each subcomponent or statement at least two spaces more than the construct within which it is nested." Also, make sure that statements that are nested at the same level have their left-most characters line up evenly.

Good:

```
if (avg >= 92.0)
{
   grade = "A";
   System.out.println("Great job!");
}
```

Bad:

```
if (avg >= 92.0)
{
grade = "A";
   System.out.println("Great job!");
}
```

Bad:

```
if (avg >= 92.0)
```

```
{
   grade = “A”;
      System.out.println(“Great job!”);
}
```

Furthermore, “a single space should be added on both sides of a binary operator”.

Good:

```
int i = 3 + 4 * 4;
```

Bad:

```
int i= 3+4 * 4;
```

Block styles:
Review Section 1.9.3. You may use either the “Next-line style” or the “End-of-line style” of creating code blocks. Each has its advantages and disadvantages. In a given file, you should use one style or the other: do not mix block styles within a single source code file.

Next-line style:

```
public class Test
{
   public static void main(String[] args)
   {
      System.out.println(“Block Styles”);
   }
}
```

End-of-line style:

```
public class Test {
   public static void main(String[] args) {
      System.out.println(“Block Styles”);
   }
}
```

Naming conventions:
As specified in Section 2.8 of the textbook, you should:

- Begin the name of each variable and method with a lower case letter. “If a name consists of several words, concatenate them into one, making the first word lowercase and capitalizing the first letter of each subsequent word – for example, the variables **radius** and **area** and the method **showInputDialog**.”
- “Capitalize the first letter of each word in a class name – for example, the class names **WindChill**, **Math**, and **JOptionPane**.”
- “Capitalize every letter in a constant, and use underscores between words – for example, the constants **PI** and **MAX_VALUE**.”

Tip: Always start the name of a .java source file with a capital letter. E.g., **WindChill.java**, *NOT* **windChill.java**. This will at least force you to start the names of your public classes with capital letters.