Program 4: ArrayList (50 Points)

**Objectives:**

The focus of this assignment is understanding of the ArrayList structure and Collections utilities. Use the ArrayList and Collections classes that are defined in the java API, in the java.util package. The StringTokenizer class will also be used to parse simple Strings, this is also found in java.util.

**Program Background:**

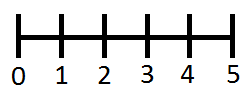
The problem of optimal scheduling has many applications, from ordering processor tasks in a CPU or scheduling real life tasks in order to maximize the profit of available assets.

**Program Description:**

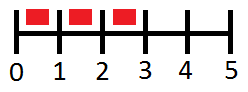
A schedulable task will be defined by having a deadline, by when it needs to be completed, and a value, which represents the importance or profit of the task. The user will input a number of these tasks and the program will schedule them such that the maximum value is achieved. Often with these scheduling problems it is not possible to schedule all tasks.

This can be done by a fairly simply Greedy Algorithm. A Greedy Algorithm is one that makes choices based upon what appears to be optimal at the time of choosing. This job scheduling algorithm is greedy because it schedules the most valuable remaining task next. So the first step is to organize our tasks in reverse order, making it easy to linearly process them.

To make the problem a bit easier, we will use integer time units, starting at time 0. Time can then be viewed as an integer number line starting at 0.



All tasks will only take one unit of time, so if there is a task that has a deadline of 3 it could be scheduled in the following locations.



That is, it can be scheduled at time 0, 1, or 2, so that it is completed by the deadline time of 3.

The algorithm is to schedule the highest valued unscheduled task in the latest available time slot by its deadline. If there is no available time slot before the deadline the task is not completed, and its value lost.

If we have the following tasks:

Task # Deadline Value

1 3 2

2 4 10

3 1 5

4 4 12

5 2 3

6 5 7

7 3 10

The first step is to sort them in reverse order by value (the order of those of the same value does not matter):

Task # Deadline Value

4 4 12

2 4 10

7 3 10

6 5 7

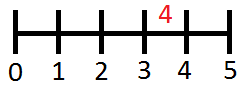
3 1 5

5 2 3

1 3 2

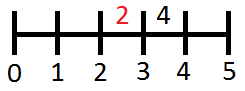
Then, starting with the most valuable, schedule each in the latest possible time such that their deadline is met.

SCHEDULING TASK 4



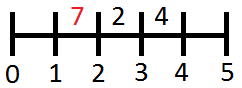
Task 4 is placed at time 3 (such that it is completed by time 4). This is the latest it can be placed.

SCHEDULING TASK 2



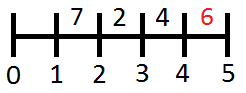
Next is Task 2. It has the same deadline as the previous task, so initially it attempts to schedule it at time 3, however there is already a task there, so it goes back a time slot, where there is room and gets scheduled at that time.

SCHEDULING TASK 7



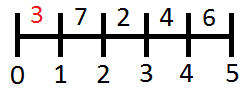
Task 7 is similar to the previous. It initially is attempted to place it at time 2, however that slot is taken, so checks the previous which is available.

SCHEDULING TASK 6



Task 6 can go at the last position 4.

SCHEDULING TASK 3



Task 3 must be done at time step 0, which is free.

The remaining tasks cannot be scheduled. The total value is the sum of the values of the scheduled tasks, in this case 44.

In this example the schedule was complete before we encountered an unschedulable task, but this will not always be the case, so you cannot just order the tasks and add the values of the first N tasks.

The following classes are required:

* InvalidDeadlineException
* InvalidValueException
* Task
* Scheduler
* SchedulerDriver

**UML DIAGRAMS FOR AND DISCUSSION FOR InvalidDeadlineException and InvalidValueException**

There will be two things that can cause a Task to be invalid, if either the deadline or the value of the task is not a positive number. Each of these cases will have their own Exception so when a Task is created with incorrect values it can report more specifically what was wrong.

|  |
| --- |
| InvalidDeadlineException extends RuntimeException |
|  |
| <<constructor>> InvalidDeadlineException(deadline : int) |

Methods:

InvalidDeadlineException(deadline : int) – Class constructor. Calls upon super class constructor with the String, “Non-positive number given for deadline: “, followed by the int that is passed.

|  |
| --- |
| InvalidValueException extends RuntimeException |
|  |
| <<constructor>> InvalidValueException(value : int) |

Methods:

InvalidValueException(value : int) – Class constructor. Calls upon super class constructor with the String, “Non-positive number given for value: “, followed by the int that is passed.

**UML DIAGRAM FOR AND DISCUSSION FOR Task**

|  |
| --- |
| Task **implements Comparable<Task>** |
| - maxDeadline : int  - deadline : int  - value : int |
| <<constructor>> Task(deadline: int, value : int)  + getDeadline( ) : int  + setDeadline(deadline : int)  + getValue( ) : int  + setValue(value : int)  + getMaxDeadline( ) : int  + toString( ) : String  + compareTo( t : Task) : int |

An instance of Task represents a single task to be scheduled. Each task has two values, one for the deadline and one for the value of that that task. Additionally, the class has a static data member (denoted by the underline in the UML diagram). A static data member in a class shares the same value for **all** instances of that class. This is necessary as we will need to keep track of the latest deadline of all Tasks so we know how big to make our time line.

Data Members:

maxDeadline – static int, must be kept up to date whenever a new Task is created

deadline – Deadline of what time step the Task must be completed by in order to be scheduled

value– Value of the Task

Methods:

Task(deadline: int, value : int) – Class constructor. Sets the non-static data members by calling upon the appropriate set methods.

getDeadline( ) : int – Returns deadline

setDeadline(deadline : int) – Attempts to set deadline data member to the int passed. If int passed is non-positive it will throw an InvalidDeadlineException. If the deadline is valid will update the value of maxDeadline if greater.

getValue( ) : int – Returns value

setValue(value : int) – Attempts to set value data member to the int passed. If int passed is non-positive it will throw an InvalidValueException.

getMaxDeadline( ) : int – Returns maxDeadline

compareTo(t : Task) : int – Override the compareTo method from implementing the Comparable interface. This determines the logical order of Tasks which will be used to sort them. Tasks with higher values should be considered greater than those with lower values.

toString( ) : String – Return a String of the following form:

“Deadline: DEADLINE Value: VALUE”, where DEADLINE and VALUE are replaced by the instances deadline and value.

**UML DIAGRAM FOR AND DISCUSSION FOR Scheduler**

|  |
| --- |
| Scheduler |
| - inputTasks : ArrayList<Task>  - outputTasks : Task[] |
| <<constructor>> Scheduler( )  + addTask( t : Task )  + scheduleTasks( )  + calculateValue( ) : int  + printInputTasks( )  + printOutputTasks ( ) |

Scheduler( ) – Class constructor. Creates a new empty ArrayList of Tasks for inputTasks.

addTask(t : Task) – Adds the passed Task to the ArrayList.

scheduleTasks( ) – Implements scheduling algorithm as discussed above.

calculateValue( ) : int – Sums the total value of all Tasks that have been scheduled.

printInputTasks( ) – Prints the Task with the highest value and the Task with the lowest value (see example output for format) followed by all inputted Tasks.

printOutputTasks( ) – Prints the scheduled Tasks in the order they should be performed as determined by scheduleTasks method.

The Collections class can be used to easily perform actions on ArrayLists and other similarly defined data structures. In particular, you will need to use sort, reverse, max, and min.

**DISCUSSION FOR ScheduleDriver**

See expected output below discussion, if your output is different in any way you are subject to losing points.

The Driver will create a Scheduler object and read Task values from the user, adding them to the scheduler as they are entered. These will be input as Strings of two whole number values, the deadline followed by the value. The user will enter the String, “DONE”, when they are finished entering Tasks.

Since the Task values are entered as a single String it will need to be parsed into two int values to pass them to the Task constructor. This can be done with the StringTokenizer class. The constructor for StringTokenizer that you will use takes a String to be parsed. Calling upon the nextToken() method of the StringTokenizer will return the next token in the String, delimited (separated) by white space characters.

If the user enters the String:

“5 8”

The first call to nextToken() will return the String, “5”, then next call will return the String, “8”. Note these are Strings, not ints. The Integer class’ parseInt() method can be used to convert a String the represents an int value into an int. This method throws a NumberFormatException if the String cannot be converted, this must be handled by outputting “Invalid Task”. If the user does not enter at least 2 tokens, this will cause a NoSuchElementException which must be handled by outputting “Invalid Task”. If they attempt to give non-positive values for a Task, the appropriate Exception as defined above must be handled by outputting the Exception.

Once the user has entered all of their Tasks output the Tasks as they were entered, run the scheduling algorithm, output the outputTasks, and finally output the value of those Tasks. If no Tasks were entered output, “No tasks given.” See expected output in the examples below.

**Example Runs (User input underlined)**

Example 1:

Input Tasks: DEADLINE VALUE

Type DONE to schedule

3 2

4 10

1 5

4 12

2 3

5 7

3 10

DONE

Task with Max value: Deadline: 4 Value: 12

Task with Min value: Deadline: 3 Value: 2

Deadline: 3 Value: 2

Deadline: 4 Value: 10

Deadline: 1 Value: 5

Deadline: 4 Value: 12

Deadline: 2 Value: 3

Deadline: 5 Value: 7

Deadline: 3 Value: 10

Scheduled Tasks:

Deadline: 1 Value: 5

Deadline: 4 Value: 10

Deadline: 3 Value: 10

Deadline: 4 Value: 12

Deadline: 5 Value: 7

Maximum Value of Tasks is 44

Example 2:

Input Tasks: DEADLINE VALUE

Type DONE to schedule

-5 8

InvalidDeadlineException: Non-positive number given for deadline: -5

5 -8

InvalidValueException: Non-positive number given for value: -8

this is not valid

Invalid Task

7 15

2 20

5 30

3 18

4 18

5 10

2 23

7 16

3 25

DONE

Task with Max value: Deadline: 5 Value: 30

Task with Min value: Deadline: 5 Value: 10

Deadline: 7 Value: 15

Deadline: 2 Value: 20

Deadline: 5 Value: 30

Deadline: 3 Value: 18

Deadline: 4 Value: 18

Deadline: 5 Value: 10

Deadline: 2 Value: 23

Deadline: 7 Value: 16

Deadline: 3 Value: 25

Scheduled Tasks:

Deadline: 2 Value: 20

Deadline: 2 Value: 23

Deadline: 3 Value: 25

Deadline: 4 Value: 18

Deadline: 5 Value: 30

Deadline: 7 Value: 15

Deadline: 7 Value: 16

Maximum Value of Tasks is 147

Example 3:

Input Tasks: DEADLINE VALUE

Type DONE to schedule

DONE

No tasks given.

**Assignment Questions**

Provide the answers to these questions in your submission directory within a file called Assignment4Questions.

1. What advantages do you gain from using an ArrayList for inputTasks rather than an array?

2. For outputTasks, why is an array a better option than an ArrayList?

3. Create six of your own scheduling test cases. What were your test cases and their results?