CPSC 233: Assignment 3 (*Due March 8 at 4 PM)*

New learning concepts: Creating and managing linked lists. Aside from 'main' you cannot implement other static methods. (Don't worry if you don't know the significance of the word 'static' yet, an explanation will come later in the term). You will be given the code for the **`Driver`** and **`Event`** classes and you must implement the methods in the **`EventList`** class. You may add new methods and change the implementation of Event.java, but don't change the **`Driver`** classes because the same test cases will be used by the markers to test your program

Problem Statement:

In this assignment, you will complete a program that simulates the processing of events (in the form of a list) in a simple operating system. New events are added to the end of the list. The system iterates through the list from the first event to the last event. In each iteration each event will execute for one time unit. The remaining time of an event is updated after each execution; and an event will be removed from the list as soon as its remaining time reaches zero (see the output of 'Driver3.java').

Program description:

In this assignment, you will work with an **`Event`** class (in *Event.java*) and the skeleton code for **`EventList`** class (in *EventList.java*). Please DO NOT change the class name and the file name, as this will result in the **`Driver`** classes failing to compile during marking. The **`Event`** class defines individual nodes in the linked list of events, whereas the **`EventList`** class defines the implementation of the linked list. The basic implementation of the **`Event`** class is provided with this assignment. The members of this class are defined as follows:

- **`private int remainTime;`**
  The remaining time of the event.
- **`private Event nextEvent;`**
  The pointer/reference to the next event in the linked list
- **`public Event (int duration)`**
  Creates an event object and initialize the remainTime to the duration of the event. The pointer/reference to the next event, nextEvent, is initialized to null.
- **`public int run()`**
  Simulates the execution of the event by decrementing the remaining time of one event in the list by one. This method returns the remaining time of the event.
- **`public void setNext (Event e)`**
  Sets the pointer/reference to the next event to the given Event e.
- **`public Event getNext ()`**
  Returns the pointer/reference to the next event.
- **`public String toString()`**
  Returns the string representation of the event.

Your task is to complete the implementation of the skeleton of the **`EventList`** class according to the following descriptions.

- **`EventList()`**
  The default constructor creates an empty linked list.
- **`public boolean addEvent(int duration)`**
  This method creates a new event object and adds it to the end of the linked list. The **`Driver`** class calls this method and passes it the duration of the new event. You may assume the duration will always be a positive integer number.
- **`public boolean process ()`**
  This method iterates through the list and runs each event/node by calling the **`run()`** method in the Event class. An event is removed from the list if the remaining time returned by the **`run()`** method reaches 0. This method returns false if the event list is empty, true otherwise.
- **`public String toString ()`**
  This special method returns a string representation of the linked list. The recommended format is "**`<event>, <event>,`** …", in which each event is separated by a comma and whitespace. Hint: you may use the **`Event.toString()`** method here to get the string representation of each event.
- **`public void printBanner ()`**
  This method prints the banner with student information.

The **`Driver`** class reads the duration of each event from an input file and creates an object of the **`EventList`** class consisting of these events. It then repeatedly calls the **`process()`** methods in the **`EventList`** class to execute the events until all events are finished, *i.e.*, the program terminates when the list is empty. Three versions of the **`Driver`** class are available to test your implementation of the **`EventList`** class at different levels. The **`Driver`** classes get information about events in the operating system from an input file provided as a command-line argument. The file should contain the duration of events, and the duration for individual events is on its own. For example, in *events.txt*, there are four events with durations 10, 5, 4, and 15. Like the previous assignment the user will enter the name of the input file in the form of a command line argument (e.g., **`java Driver1 events.txt`**). For the sample input file (*events.txt*), the behavior and the output of each version of the **`Driver`** class are as follows:

- *Driver1.java*
  This class creates the linked list of the events according to the input file (*events.txt*) and displays the linked list. This class can be used to test the **`printBanner()`**, **`EventList()`**, **`addEvent()`**, and **`toString()`** methods in the **`EventList`** class.
  Output: 10, 5, 4, 9
- *Driver2.java*
  This class creates the linked list of the events according to the input file (*events.txt*), calls the **`process()`** method once, and then display the linked list. This class can be used to test the **`printBanner()`**, **`EventList()`**, **`addEvent()`**, and **`toString()`** methods in the **`EventList`** class. It also partially tests the **`process()`** method.
  Output: 9, 4, 3, 8
- *Driver3.java*
  This class creates the linked list of the events according to the input file (*events.txt*), processes all events, and then displays the linked list. This class can be used to test all

methods in the **`EventList`** class. Of particular note, it actually 'runs' the simulation by advancing time (note how the duration counts down and completed events are removed from the list once they are complete).
Output:

```
9, 4, 3, 8
8, 3, 2, 7
7, 2, 1, 6
6, 1, 5
5, 4
4, 3
3, 2
2, 1
1
```

Programming Requirements:

In this assignment, you **may**:

- modify the Event class, including new data members and methods
- create your own input files to test difference scenarios
- implement the bodies of the existing methods in the **`EventList`** class
- write your own version of the **`Driver`** class to test your **`EventList`** class

You **may NOT**:

- add new methods to the **`EventList`** class
- change the method signature of the public methods in the **`EventList`** class
- change the three versions of the **`Driver`** class provided. Your submissions will be marked using these classes with different input files.

External libraries/methods that can be used (you don't write yourself)

You can use pre-created code for console input and output (*e.g.*, **`print`** and **`Scanner`**), classes for file input and to methods convert from **`String`** to numeric types.

Important reminders

- **Assignment Guidelines:** When working on your assignments, please adhere strictly to the generic assignment submission guidelines (which apply to all assignments) as well as the ones listed in this assignment.
- **Handing in your assignment**: Use the UNIX 'submit' program. Make sure you submit both *Event.java* and *EventList.java*, with the exact file names. The submit command is:

  `submit -c 233 -a 3 Event.java EventList.java`
- **Collaboration**: Assignments must reflect individual work, group work is not allowed in this class nor can you copy the work of others.