**LA7 – Pair Programming  
Course Registration System**

|  |
| --- |
| **Due Date** |
| **Wednesday 12/04/19 @ 11:59pm** |

**Objectives**

* Priority Queues
* LinkedList
* Generics

**Problem Specification**

University of Parkview provides inexpensive education to students. The tuition rates are very low and this attracts more students than can be accommodated in the available classes. This causes a problem as the university cannot hire enough faculty members to offer enough courses for all students. Popular classes run out of space very soon after the course registration system is open. The following rules have therefore been put in place to decide which students have priority in registering for specific classes.

* Students who are majors in the department offering a class have priority to register this class over students from other departments.
* After that, a student who is closer to graduation has priority to register for the class. For example, seniors have priority over juniors, sophomores have priority over freshmen, etc.
* If two students from the same department are also in the same year (or level), the priority is given to the student with the higher GPA.
* If the priority cannot be decided by applying the above rules, the “first come, first served” rule will be applied to decide who has priority to register for the class.

Develop a Java application to implement this Course Registration System.

The courses offered are contained in a text file in the format shown below. Use this file to create a list of **Course** objects (more details specified in the **Design Requirements** section).

**Example**: Input file for available courses (named **course.txt**)

// This is a **comma-separated** input file.

// Data stored in the file is in the order: courseDept, courseNumber, capacity (maximum number of

// students that can be accommodated in the class)

CS,1040,10

CS,1060,2

You will also be keeping a **priority queue** of registration requests – this means that the items (**Requests**) in the queue will be stored in order according to the priority rules specified above (not strictly in the “first in, first out” manner of a typical queue. You will first read in a series of registration requests from a second text file, and then put each one into the queue according to the priority rules. The item with the highest priority should be at the front of the queue.

**Example**: Input File containing information about students waiting /requesting to be registered for courses (input file is named **request.txt**)

// Data stored in the file in order: studentName, studentLevel, studentMajor, courseDept,

// courseNumber, list of grades in four courses with each grade followed by the corresponding number

// of credits (to enable calculation of the student’s GPA),. Format for the grades and credits is:

// grade1,credit1, grade2,credit2,grade3,credit3,grade4,credit4

Jeremy Anderson,Senior,ECE,CS,1040,3.2,3,3.1,2,4.0,3,3.6,2

Jeremy Anderson,Senior,ECE,CS,1060,3.2,3,3.1,2,4.0,3,3.6,2

Peter Bond,Junior,CS,CS,1060,3.0,3,3.0,2,3.9,3,3.5,2

Peter Bond,Junior,CS,CS,1040,3.0,3,3.0,2,3.9,3,3.5,2

Jonah Fish, Senior,CS,CS,1060,3.1,2,3.4,2,4.0,3,3.7,2

Jonah Fish, Senior,CS,CS,1040,3.1,2,3.4,2,4.0,3,3.7,2

Jonathan Fin,Freshman,CS,CS,1040,3.9,2,3.8,3,4.0,3,3.8,3

Jonathan Fin,Freshman,CS,CS,1060,3.9,2,3.8,3,4.0,3,3.8,3

Then you will process each request in the ordered queue (starting from the front and going in order).

For each request:

* print out the request;
* either add the student to the course’s class list if there is room, or print a message indicating that the student cannot be registered for the course.

**You also need to print the queue contents before and after the processing of all requests (the queue should be empty after all requests have been processed).**

**Hint**: You may want to create a toString method for the Request class to make printing easier.

Finally, print out the class list for each course (after all requests have been processed).

**Design Requirements**

A **Priority Queue** is similar to a typical queue where we insert an element at the rear and remove an element from the front; but there is one difference: the order of storing elements in the priority queue depends on the **priority** of the elements. The element with highest priority will be moved to the front of the queue and one with lowest priority will move to the back of the queue. Thus, it is possible that when you enqueue an element at the rear of the queue, it can move to the front because of its higher priority.

You are to implement a **generic** **priority queue** data structure in which the allowable objects should belong to a class that implements the **Comparable** interface. **You must not use the built-in PriorityQueue collection in the Java API** (i.e. **you should not have an import statement for this class in your program** since you will be writing your own Priority Queue class). Your priority queue class should use the **Node** class (which you will also write) for storing objects. In your implementation of a priority queue, you should have methods to **enqueue** an object, **dequeue** an object, determine if the queue **is empty**, and **print** the queue contents**.** Specifically, your Priority Queue class should implement the following methods:

// Determine if the priority queue is empty.

- public boolean isEmpty();

// Add object received to the priority queue taking into consideration the rules governing priority.

- public void enqueue(E data);

// Remove the next object to be processed from the priority queue.

- public E dequeue();

// Print the contents of the queue

-public void Qprint(); {

The **INode** interface for the Node class is given below.

public interface **INode** {

// Returns the data stored in this node.

E getData();

// Returns the node next to this node.

Node<E> getNext();

// Sets node received as the next node to this node.

void setNext(Node<E> next);

}

The **Node** class should also have the following **constructors**:

Node(E dataValue); // Constructor

Node(E dataValue, Node<E> nextNode); // Constructor

In addition, you should have a class **Course** representing a Course object, with attributes for the data contained in the courses.txt input file, along with a list of enrolled students for that course. You should use a LinkedList for this list of students. Class **Course** should also have methods for the following:

* checking if the course is full,
* adding a student name to the list, and
* printing out the class list.

The Course class should therefore implement the **ICourse** interface (given below).

public interface **ICourse**{

// Determine if the capacity for this course has been reached.

boolean isFull();

// Add this student to the **linkedlist** for enrolled students for this course.

void addStudent(String name);

// Print the classlist for this course.

void printClassList();

// Determines if this course object is the same as the object received as parameter.

boolean equals(Object arg0);

}

The course class should have a **LinkedList** of enrolled students for each course object. You are to implement a **generic LinkedList** data structure where the type parameter is not constrained. **You must not use the built-in LinkedList collection in the Java API** (i.e. you should not have an import statement for this in your program since you will be writing your own LinkedList for enrolled student). Your **LinkedList** class should use the same **Node** class (defined earlier) for storing objects. In your implementation of a **LinkedList**, you should have methods to **add** an object (whose data is the parameter received), **get** an object in specific position, and determine if the LinkedList **is empty**. Specifically, your **LinkedList** class should implement the following methods:

// Determine if the LinkedList is empty.

public boolean isEmpty();

// Add object received to the linked list if he/she has the priority.

public void add(E item);

// get the object in specific position in the LinkedList used to print the enrolled student contents inside the Course class and return the name

public E get(int position);

// Return number of elements in the list.

public int size();

You should also have a **Request** class representing a registration request, which includes attributes for the data contained in the request.txt input file. This class represents the objects that you will be prioritizing in your queue. Some requests have a higher priority than other requests. In order to compare two requests and determine the order of priority between the two, the class should implement the **Comparable<T>** interface.

**NOTE: This is an existing interface in the Java documentation, so you need not create the interface in your LA7 project**.)

The single method in the Comparable interface is shown below:

public interface Comparable<T>(// Pre-existing interface in Java documentation

int compareTo(T other);

}

In addition to implementing the Comparable interface, your Request class should also implement the following methods:

// Constructor

public Request(String studentName, String studentDept, String studentLevel,

String courseDept, int courseNumber, double[][] GPA\_Array)

// Returns number of years to graduation (0 for seniors, 1 for juniors etc.). This is determined from the

// student’s level – senior, junior, etc.

public int yearsFromGraduation(String level)

// Calculate the GPA for a particular student.

private double GPA\_Cal(double[][] GPA\_Array)

// Getters for a student’s name and department, and the department and number of a course

Finally, you should have a **Controller** class which implements the **IController** interface and runs everything. This class works with the other classes to process course registration for students. It is responsible for the following:

* adding new requests from the requests input file,
* adding courses from the course input file,
* processing all of the requests (registering students where this is possible, and then
* printing out the class list for all courses as mentioned earlier.

**public interface IController {**

// Read courses from input file and add each course to a LinkedList of courses. Close file.

void readCourseFile(); // Use a try-catch block.

// Read each request from the request input file and use it to create a Request object.

void readRequestFile(); // Use a try-catch block.

// Store the request object in the requests priority queue.

void addRequest(Request req);

// Process all the requests as follows: if request can be granted, update the relevant classlist,

// update the class/course capacity, print out a message that the student who made that

// request has been successfully registered for that course. Else, print out a message that

// student could not be registered.

void processRequests();

// Return the course object with data values that match the parameters received.

Course getCourse(String courseDept,int courseNumber);

// Print classlists for all courses.

void printClassList();

}

The **Controller** class should have the following constructor:

public Controller(PriorityQueue<Request> requestQueue, LinkedList<Course> courses,

BufferedReader fileIn, BufferedReader fileIn1)

**The main method has been provided below. It is not to be changed.**

**public** **static** **void** main(String[] args) **throws** IOException {

PriorityQueue<Request> requestQueue = **new** PriorityQueue<Request>();

LinkedList<Course> courses = **new** LinkedList<Course>();

BufferedReader fileIn = **new** BufferedReader(**new** FileReader("course.txt"));

BufferedReader fileIn1 = **new** BufferedReader(**new** FileReader("request.txt"));

IController control = **new** Controller(requestQueue, courses, fileIn, fileIn1);

control.readCourseFile();

control.readRequestFile();

control.processRequests();

control.printClassList();

}

**Pseudocode**

Write Pseudocode for the required methods mentioned above and any other methods you may include in your program.

**Consider the questions below and include answers or required information in your Pseudocode document.**

1. The data in your queue will be Request objects, but instead of restricting it to that, use generics to make your queue work with any type of data (based on the type parameter specified when the Priority Queue object is declared).
2. In addition to being generic, your PriorityQueue class needs to ensure that whatever type is used must be able to compare two items of that type. You can do this by constraining the type parameter to types which implement the Comparable<T> interface (where T represents the type parameter in question). Write the class header for this class, indicating this constraint.
3. As described above, the Request class should implement the Comparable<T> interface, which specifies the compareTo(T other) method. What does the **parameter** for this method represent, and what are the possible return values for this method (include the possible return values and specify what these values indicate)?

**Implementation Phase**

Using the pseudocode developed, write the Java code for your assignment.

**You may create new methods, but must use the provided methods (i.e. the methods provided must not be by-passed).**

**Testing Phase**

In testing your application, you may want to print out the queue every time you add a new request, so that you can verify that requests are being inserted in the correct order. Make sure to try different combinations, reading the requests in different sequences to make sure that they still end up in the correct order in the queue.

**Example Output**

CS,1040,10

CS,1060,2

<<<<<<<<<<<< Beginning of Queue Contents >>>>>>>>>>>>>>>>>

Request@70dea4e

Request@5c647e05

Request@33909752

Request@55f96302

Request@3d4eac69

Request@42a57993

Request@75b84c92

Request@6bc7c054

<<<<<<<<<<<< End of Queue Contents >>>>>>>>>>>>>>>>>

Request@70dea4e processed.

Jonah Fish successfully registered CS 1040

Request@5c647e05 processed.

Jonah Fish successfully registered CS 1060

Request@33909752 processed.

Peter Bond successfully registered CS 1040

Request@55f96302 processed.

Peter Bond successfully registered CS 1060

Request@3d4eac69 processed.

Jonathan Fin cannot register for CS 1060

Request@42a57993 processed.

Jonathan Fin successfully registered CS 1040

Request@75b84c92 processed.

Jeremy Anderson cannot register for CS 1060

Request@6bc7c054 processed.

Jeremy Anderson successfully registered CS 1040

<<<<<<<<<<<< Beginning of Queue Contents >>>>>>>>>>>>>>>>>

Queue is Empty

<<<<<<<<<<<< End of Queue Contents >>>>>>>>>>>>>>>>>

Class List for CS 1040

Jonah Fish

Peter Bond

Jonathan Fin

Jeremy Anderson

Class List for CS 1060

Jonah Fish

Peter Bond

**Additional Requirements**

**Note: The due date for LA7 submission is December 4th, 2019; the final date for submission of all assignments is December 6th (last day of classes). No more assignments will be accepted after this date (12-6-19).**

**Pair Programming**

This is a pair programming assignment. You will need to work on this assignment with your team mate(s).

As explained in the lab, there should be a visible evidence of both partners in a team contributing to the work required for this assignment. You will be graded on how well you use your GIT repository. The “commits” on GitLab will be reviewed to determine if there was input from both team members. There should be at least two commits from each person on the team. In addition, at the end of the pseudocode document, a **brief** explanation should be included explaining what aspect of the application each team member worked on.

**You must add your lab instructor to your GIT repository to provide access for grading purposes.** The “Git Guide” document posted on the Content page in Elearning includes steps for specifying how to utilize Git.

If you experience any problems with Git, get in touch with your lab instructor at the earliest opportunity. A zip file containing your submission needs to be submitted to Elearning **only** if you can’t get Git working (you will be penalized for this, so you need to make every effort to work with Git). If your code is in the repository, you will **not** have to upload it to Elearning. However, you still need to keep to the due date for submission – note that the most recent date/time of modification can be seen in Git.

**Pseudocode**

A proper *design* (with detailed pseudocode) and proper *testing* are essential. Pseudocode should be submitted by each team member to the appropriate folder in Elearning. See additional instructions for Pseudocode in the “Assignment Submission” section below.

*Note:* ***Correct pseudocode development*** *will be worth* ***40%*** *of the total LA grade.*

You will need to **generate Javadoc** for your project. Follow the steps shown in class (a copy of these steps can be found on the Content page in Elearning, or can otherwise be provided to you by your Lab instructor). If you follow the steps accurately, a “**doc**” folder (for Javadoc) should be created in your project.

**Coding Standards**

You must adhere to all conventions in the CS 1120 Java coding standards (available on Elearning for your Lab). This includes the use of white spaces and indentations for readability, and the use of comments to explain the meaning of various methods and attributes. Be sure to follow the conventions also for naming classes, variables, method parameters and methods.

**Assignment Submission**

* The final version of your project should be uploaded to the GitLab repository (a Git push) by the due date / time.
* The pseudocode document for each pair of students should be submitted to the appropriate folder in Elearning by both team members. Only **one** pseudocode document is required for each team / pair and the names of the team members should be clearly indicated on the document.

**NOTE**: The eLearning folder for LA submission will remain open but only till the last day of classes (Dec. 6th, 2019). It will indicate how many days late the pseudocode was submitted where applicable. The dropbox will be inaccessible after Dec 6th after which no more credit can be received for the assignment.

The penalty for late submissions as stated in the course syllabus will be applied in grading any assignment/pseudocode submitted late.