Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS212 - Data structures and algorithms

Assignment 1 Specifications: Linked Lists

Release date: 16-02-2026 at 06:00

Due date: 13-03-2026 at 23:59

Total marks: 220

Contents

1 General Instructions

- *Read the entire assignment thoroughly before you start coding.*
- This assignment should be completed individually; no group effort is allowed.
- **To prevent plagiarism, every submission will be inspected with the help of dedicated software.**
- Be ready to upload your assignment well before the deadline. There is a late deadline which is 1 hour after the initial deadline which has a penalty of 20% of your achieved mark. **No extensions will be granted.**
- You may not import any of the built-in Java data structures. Doing so will result in a zero mark. You may only use native 1-dimensional arrays where applicable.
- If your code does not compile, you will be awarded a mark of zero. Only the output of your program will be considered for marks, but your code may be inspected for the presence or absence of certain prescribed features.
- If your code experiences a runtime error, you will be awarded a zero mark. Runtime errors are considered unsafe programming.
- **Ensure your code compiles with Java 8**
- The usage of ChatGPT and other AI-Related software to generate submitted code is strictly forbidden and will be considered as plagiarism.
- The use of this practical, in any form, by any company/business/organisation outside the University of Pretoria is strictly forbidden. This includes, but is not limited to, the use of the practical for discussion sessions, review sessions, marketing tools, providing certain aspects as a free service, or publishing the specification on a website.
- Note that plagiarism is considered a very serious offence. Plagiarism will not be tolerated, and disciplinary action will be taken against offending students. Please refer to the University of Pretoria's plagiarism page at `https://portal.cs.up.ac.za/files/departmental-guide/`.

2 Overview

For this assignment you will be implementing a gradebook system which allows you to store student marks for assessments. There can be any number of assessments and students, and we want to be able to efficiently add more students and assessments, as well as be able to sort the practicals based on student marks. To do this we will use a system of doubly circular linked lists. Each student will be represented by a node, and for storage efficiency these nodes we will be reused by the different practicals.
An example of a gradebook is shown below. This example has 5 students and 5 practicals, and their marks are listed in this table.

	Prac1	Prac2	Prac3	Prac4	Prac5
u1	78	64	91	55	83
u2	42	88	73	70	57
u3	90	76	84	92	68
u4	61	49	70	58	81
u5	85	93	67	74	60

The connections in the linked list can be seen below. It is really difficult to see what is going on in the full gradebook because of the number of connections, which is why underneath it is split up to only show one practical at a time. Each practical is a doubly circular linked list where if you start at the head and use the next pointers, then the students are sorted according to their marks in descending order. If you start at the tail and follow the prev pointers, then it is in ascending order.

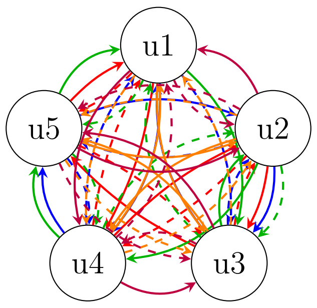


Figure 1: Full gradebook

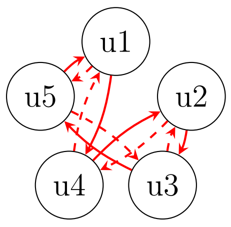


Figure 2: Prac1

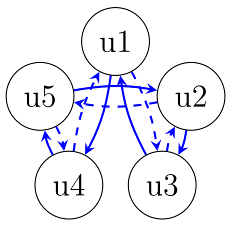


Figure 3: Prac2

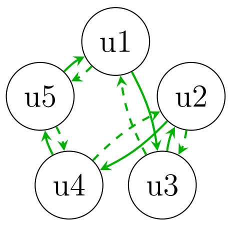


Figure 4: Prac3

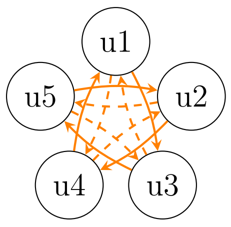


Figure 5: Prac4

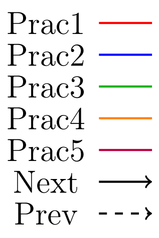


Figure 6: Legend

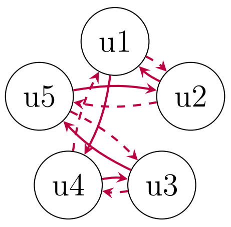


Figure 7: Prac5

Figure 8: Individual practical plots

3 Student

This class is given to you. Do not make any changes to it, since it will be overwritten on Fitchfork. This class will be used as the nodes in the lists.

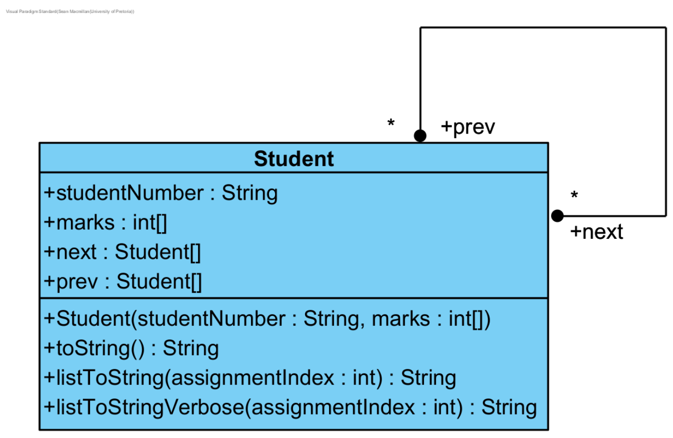

*
+prev
*
+next
Student
+studentNumber : String
+marks : int[]
+next : Student[]
+prev : Student[]
+Student(studentNumber : String, marks : int[])
+toString() : String
+listToString(assignmentIndex : int) : String
+listToStringVerbose(assignmentIndex : int) : String

Figure 9: Student UML

3.1 Members

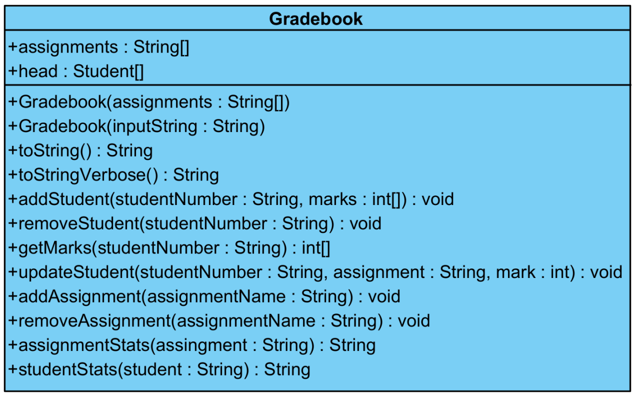
- +studentNumber: String
  - This is the student number for the current node.
- +marks: int[]
  - This is an array of ints used to store the marks for this student. Each value in the array is for a different practical. The order of the marks is determined by the Gradebook in which this student is stored.
- +next: Student[]
  - This is an array of Student pointers used to store the student that is next in the linked list. Each value in the array is for a different practical. The order of the marks is determined by the Gradebook in which this student is stored.
- +prev: Student[]
  - This is an array of Student pointers used to store the student that is before this student in the linked list. Each value in the array is for a different practical. The order of the marks is determined by the Gradebook in which this student is stored.

3.2 Functions

- +Student(studentNumber: String, marks: int[])
  - This is the constructor for Student.
  - The passed-in parameters are copied to the members, while all of the pointers are set to null.
- +toString(): String
  - This returns a string representation of the student.
  - This versions includes all of the marks, as well as the next and previous pointers.
- +listToString(): String
  - This returns a string representation of the linked list starting at this Student for an assignment with the passed-in index.
  - This will only use the next pointers, to make it easier to read.
- +listToStringVerbose(): String
  - This is similar to the previous function, except that it will show all of the marks as well as the next and prev pointers.

4 Gradebook

This class will be used to store a gradebook.



```
Gradebook
+assignments : String[]
+head : Student[]
+Gradebook(assignments : String[])
+Gradebook(inputString : String)
+toString() : String
+toStringVerbose() : String
+addStudent(studentNumber : String, marks : int[]) : void
+removeStudent(studentNumber : String) : void
+getMarks(studentNumber : String) : int[]
+updateStudent(studentNumber : String, assignment : String, mark : int) : void
+addAssignment(assignmentName : String) : void
+removeAssignment(assignmentName : String) : void
+assignmentStats(assingment : String) : String
+studentStats(student : String) : String
```

Figure 10: Greadebook UML

4.1 Members

- +assignments: String[]
  - This is an array of assignment names.
- +head: Student[]
  - This is an array of Student pointers which contains the heads of the linked lists.

4.2 Some notes on the arrays

- For a given gradebook, the assignments and head arrays have to be the same size. For all of the students associated with this gradebook, their marks, next and prev arrays also have to be the same size as these two arrays. You may assume that this will always be the case. You also have to make sure that your code never breaks this rule.
- The indexes of all of the arrays are linked. That means that assignment[0] contains the name of the first assignment. The head of the linked list for this assignment is at head[0]. To traverse this linked list, you have to use next[0] and prev[0].
- All students have to have a mark for all assignments. This means that all of the linked lists have to be the same size, and the size will be the number of students.
- It is possible that a gradebook contains no assignments. If this is the case the gradebook also can't have any students.
- It is possible that a greadebook contains no students. It is still possible that the gradebook can have assignments, but the head pointers will all be null.
- If head[0] is null, then all entries in the head array have to be null. If head[0] is not null, then non of the entries in head can be null.

4.3 Functions

- +Gradebook(assignments: String`[]`)
  - This function is given to you. Do not change it.
  - It creates a gradebook with assignments matching the passed-in parameters, and with no students.
- +Gradebook(inputString: String)
  - This function is given to you. Do not change it.
  - This is the string constructor. It can be used to reconstruct gradebooks.
  - The format of the input string is the same as the output of the toString function.
- +toString(): String
  - This function is given to you. Do not change it.
  - This returns a string representation of the gradebook.
  - Only the next pointers are used to make it easier to read.
- +toStringVerbose(): String
  - This function is given to you. Do not change it.
  - This prints all of the links in the gradebook. It can be used to make sure all of the links are correct.

- +addStudent(studentNumber: String, marks: int[]): void
  - Creates a new student using the passed-in parameters.
  - You may assume that the size of the marks array is the same as the assignments array.
  - You may assume that no duplicate studentNumbers will be used.
  - Make sure that the student is in the correct position in each linked list, to ensure the sorting is correct.
  - If two students have the same mark for a practical, then the order should be based on their student numbers. Compare the strings alphabetically and the lower student number should be first. I.e. "u1" should be before "u2".
- +removeStudent(studentNumber: String)
  - Remove the student with the passed-in student number.
  - The student should be removed from all of the assignments.
  - If the student doesn't exist then do nothing.
- +getMarks(studentNumber: String): int[]
  - Returns the array of marks for the passed-in student number.
  - If the student was not found then return null.
- +updateMarks(studentNumber: String, assignment: String, mark: int): void
  - Update the passed-in student's mark for the passed-in assignment to the new mark.
  - If the student number or assignment doesn't exist then do nothing.
  - **Important:** Updating a mark will most likely cause the order of that linked list to be wrong. You have to fix this, to ensure the lists are always in the correct order.
- +addAssignment(assignmentName: String): void
  - Add an assignment to the gradebook using the passed-in name.
  - All of the arrays associated with this gradebook have to grow by 1. The new assignment should be at the end of the arrays.
  - All of the students should get 0 for this assignment by default.
- +removeAssignment(assignmentName: String): void
  - Remove the passed-in assignment from the gradebook.
  - If the assignmentName is not found inside the gradebook, then do nothing.
  - All of the arrays associated with this gradebook have to shrink by 1. The elements in the arrays have to be shifted down to prevent gaps in the arrays.

- +assignmentStats(assignment: String): String
  - This should return a string with the statistics for a given assignment.
  - If the assignment does not exist then return an empty string.
  - See the provided main for an example of the output.
  - The format is as follows:
    - Start with the assignment name followed by a space, and then the word "stats" and a newline.
    - Then add the phrase "Min:". After this add the smallest mark for this assignment. End with a newline.
    - Then add the phrase "Max:". After this add the largest mark for this assignment. End with a newline.
    - Then add the phrase "Avg:". After this add the average mark for this assignment. Use String.format("%.2f",average) to display the answer to 2 decimal points. Use a double variable to store the answer during calculations. End with a newline.
    - If there are no students for this assignment, then replace the values with "-".
- +studentStats(student: String): String
  - This should return a string with the statistics for a given student.
  - If the student doesn't exist then return an empty string.
  - See the provided main for an example of the output.
  - The format is as follows:
    - Start with the student number followed by a space, and then the word "stats" and a newline.
    - Then add the phrase "Min:". After this add the smallest mark for this student. End with a newline.
    - Then add the phrase "Max:". After this add the largest mark for this student. End with a newline.
    - Then add the phrase "Avg:". After this add the average mark for this student. Use String.format("%.2f",average) to display the answer to 2 decimal points. Use a double variable to store the answer during calculations. End with a newline.
    - Then add the phrase "Best Position:". After this add the best position the student achieved. End with a newline. See the example below for what this means.
    - Then add the phrase "Worst Position:". After this add the worst position the student achieved. End with a newline. See the example below for what this means.
    - If there are no students for this assignment, then replace the values with "-".
  - Example: Assume the gradebook from the example at the start of Section 2 is used and that this function is called with "u3".
    - The output should be:

```
u3 stats
Min:68
Max:92
Avg:82.00
Best Position:1
Worst Position:3
```

    - The best position u3 achieved was for Prac1 and Prac4, both of which the student had the largest mark out of the group.
    - The worst position u3 achieved was for Prac2 and Prac5, both of which the student had the third smallest mark out of the group.

5 Testing

As testing is a vital skill that all software developers need to know and be able to perform daily, approximately 10% of the assignment marks will be assigned to your testing skills. To do this, you will need to submit a testing main (inside the Main.java file) that will be used to test an Instructor-provided solution. You may add any helper functions to the Main.java file to aid your testing. In order to determine the coverage of your testing the jacoco tool. The following set of commands will be used to run jacoco:

```
javac *.java
rm -Rf cov
mkdir ./cov
java -javaagent:jacocoagent.jar=excludes=org.jacoco.*,destfile=./cov/output.exec
   -cp ./ Main
mv *.class ./cov
java -jar ./jacococli.jar report ./cov/output.exec --classfiles ./cov --html
   ./cov/report
```

This will generate output which we will use to determine your testing coverage. The following coverage ratio will be used:

$$\frac{\text{number of lines executed}}{\text{number of source code lines}}$$

and we will scale this ratio according to the size of the class.
The mark you will receive for the testing coverage task is determined using table 1:

Coverage ratio range	% of testing mark
0%-5%	0%
5%-20%	20%
20%-40%	40%
40%-60%	60%
60%-80%	80%
80%-100%	100%

Table 1: Mark assignment for testing

Note the top boundary for the Coverage ratio range is not inclusive except for 100%. Also, note that only the function stipulated in this specification will be considered to determine your mark. Remember that your main will be testing the instructor-provided code and as such it can only be assumed that the functions outlined in this specification are defined and implemented.

6 Upload checklist

The following files should be in the root of your archive

- Gradebook.java
- Main.java
- Any textfiles needed by your Main

7 Allowed libraries

- No imports allowed.

8 Submission

You need to submit your source files on the FitchFork website (https://ff.cs.up.ac.za/). All methods need to be implemented (or at least stubbed) before submission. Place the above-mentioned files in a zip named uXXXXXXXX.zip where XXXXXXXX is your student number. Your code must be able to be compiled with the Java 8 standard.

For this practical, you will have 3 upload opportunities and your best mark will be your final mark. Upload your archive to the appropriate slot on the FitchFork website well before the deadline. **No late submissions will be accepted!**