UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

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

COS212 - Data structures and algorithms

Practical 1 Specifications: Recursion and Linked Lists

Release date: 17-02-2025 at 06:00

Due date: 21-02-2025 at 23:59

Total marks: 70

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, as no extension 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.
- Read the entire specification before you start coding.
- **Ensure your code compiles with Java 8**
- The usage of ChatGPT and other AI-Related software is strictly forbidden and will be considered as plagiarism.
- 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/`.
- Please note that there is a late deadline which is 1 hour after the initial deadline, but you will lose 20% of your mark.

2 Warning

For this practical, you are **only allowed to use recursion**. The following rules must be adhered to for the entirety of this practical. Failure to comply with any of these rules **will result in a mark of 0**. The words "for","do" and "while" may not appear anywhere in any of the files you upload. Make sure you do not use any variable or function names that contain these words, and also **do not use these words in your comments**. For safety reasons, before uploading make sure to search (Ctrl F) through your files to make sure you don't find these keywords.

3 How to convert iteration to recursion

If you are struggling to implement some of the functions recursively, try to first create an iterative solution, as this might be more natural for you to implement. After you have a working iterative solution, then use the following examples, to convert the iterative solution to a recursive solution.

3.1 For loop

Assuming we have a basic for loop, like the one given below.

```
for(int i = 10 ; i < 20 ; i++){
    System.out.println(i);
}
```

The steps to convert this to a recursive function are listed below:

- The looping variable initialization is done in the normal function call.
- The looping variable should be passed as a parameter.
- The looping condition is the base case of the recursive function.
- The looping update statement is passed down in the recursive call.

Thus the recursive version is given below

```
// This is the normal function. It is used to initialize the variables
public static void recursivePrint(){
    recursivePrint(10); // The recursive function is called with the initialization condition
}
// This is the recursive function. Note that the return type and parameters can be different from the original
private static void recursivePrint(int i){ // All loop variables are passed in
    if(!(i < 20)){ // The loop condition is used as the base case.
        return;
    }
    System.out.println(i); // The body of the loop is called here
    recursivePrint(i+1); // The function is then recursively called with the updated loop variable
}
```

3.2 While loop

Let's assume we have a while loop, with a variable outside of the loop which we want to update, like the one below.

```java
int i = 0;
String print = "";
while(i <= max){
    if(i != 0){
        print += ",";
    }
    print += String.valueOf(i);
    i += 1;
}
System.out.println(print);
```

The same concept as the **for loop** can be used. The loop condition is the base case. The loop variable is the parameter passed into the recursive function. Techinically you can pass the value which you want to update (String print) as a parameter, but for EO's this is not always possible, as you are not allowed to add helper functions or change the signature. Thus, it is recommended to rather return the value which you want to update. The solution to this is to return the value in the recursive function.

```java
// This is the normal function.
public static void recursiveString(int max){
    // Since we want to update a String variable, the recursive helper function
       returns a String
    // Thus we print out the result of the recursive call.
    System.out.println(recursiveString(max,0));
}

// The max variable is passed in, which won't change and the i variable is the
   loop variable
private static String recursiveString(int max,int i){
    // This is not quite the loop condition. This is because with the normal
       loop condition, it is
    difficult to remove the extra , at the end. Thus we end one earlier and
       leave the comma out.
    if(i == max){
        return String.valueOf(i);
    }
    // Note that we can append the recursive call to the back of the string.
    return String.valueOf(i) + "," + recursiveString(max,i + 1);
}
```

4 Overview

Recursion is an important tool for programming, as some problems have solutions that can easily be solved using recursion. Recursion occurs when a function calls itself directly or indirectly through other functions. Linked Lists are data structures that are usually traversed using iteration but this can be converted to recursion, and it is a good idea to familiarize yourself with the idea of traversing data structures using recursion as a tree data structure is usually traversed using recursion.
For this practical, you will be implementing a Linked List which will be used to store student academic records. These academic records will be sorted using a specific set of rules built into the Linked List.

5 Tasks

You are tasked with creating a program that can store, delete and gather statistics for student's marks in a module. Student marks will be stored in a `TestMark` object, which acts as a node in a linked list, and this will be stored in a `Gradebook` to calculate statistics on the marks. Please note that using any iterative loops will be seen as a fail and will result in a mark of 0, as was explained in Section 2.

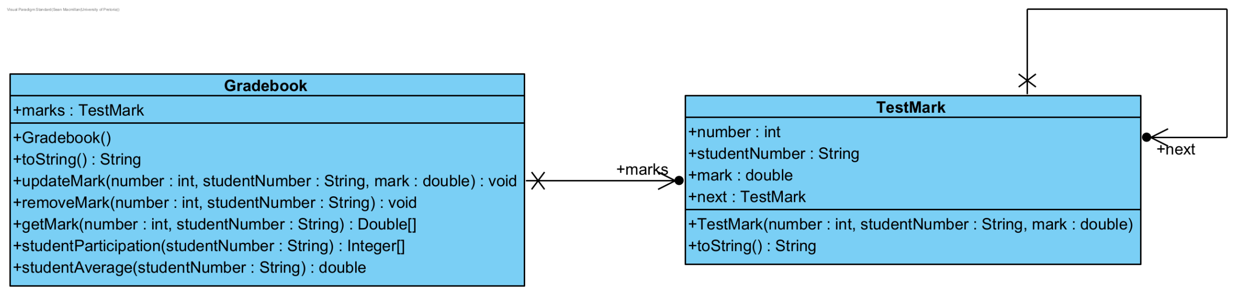
Gradebook
+marks : TestMark
+Gradebook()
+toString() : String
+updateMark(number : int, studentNumber : String, mark : double) : void
+removeMark(number : int, studentNumber : String) : void
+getMark(number : int, studentNumber : String) : Double[]
+studentParticipation(studentNumber : String) : Integer[]
+studentAverage(studentNumber : String) : double
+marks
TestMark
+number : int
+studentNumber : String
+mark : double
+next : TestMark
+TestMark(number : int, studentNumber : String, mark : double)
+toString() : String
+next

Figure 1: UML

5.1 TestMark

This is a class that will represent a node in a linked list.

5.1.1 Members

- +number: int
  - This is the test number for this specific mark.
- +studentNumber: String
  - This is the student number for this specific mark.
- +mark: double
  - This is the mark that the student achieved for this test.

5.1.2 Functions

- +TestMark(number: int, studentNumber: String, mark: double)
  - This is the constructor to create a `TestMark` object. Initialise all member variables to the passed-in parameters.

- +toString(): String
  - This returns a String representation of the node, and all nodes following this node in the list.
  - The format for each `TestMark` is as follows. Note that there are no spaces or newlines in the format.
    - Start with the word "Test".
    - Add the test number.
    - Add a colon (:).
    - Add the student number.
    - Add the student's mark inside brackets. Note that the mark must be formatted to 2 decimal places (Hint: *String format: "%.2f"*).
  - There should be an arrow (->) between each node. Note that there should not be an arrow after the last mark.

5.2 Gradebook

5.2.1 Members

- +marks: TestMark
  - This is the first mark in the gradebook. If there are no marks in the gradebook then this will be `null`.

5.2.2 Functions

- +Gradebook()
  - The constructor should initialise the gradebook to show that there are no marks recorded yet.
- +toString(): String
  - Return a string representation of the gradebook.
  - If the gradebook is empty, then return an empty string.
  - Otherwise, the format is as follows (Note that there are no spaces or newlines in the format):
    - Inside square brackets, put the number of marks in the gradebook.
    - Call the `toString` function on the first mark and add this after the closing square bracket.

- updateMark(number: int, studentNumber: String, mark: double): void
  - This should add a new mark to the gradebook.
  - Duplicate marks are allowed, since students are allowed multiple attempts for each test in this scenario.
  - The marks should first be sorted by test number. The smaller the test number the earlier in the gradebook the new mark should be.
  - Inside each test number, the marks should be sorted by the mark the student achieved. Students with a higher mark should be earlier in the gradebook than students with a lower mark.
  - If both test number and mark is the same, then the order should be the same as which the mark was read in. Thus, marks that were read in first should be earlier in the gradebook.
- removeMark(number: int, studentNumber: String): void
  - This should remove all marks with a matching number and matching student number. Only delete `TestMarks` where *both* of these conditions are met.
- getMark(number: int, studentNumber: String): Double`[]`
  - This should return an array of student marks.
  - This should return all marks with a matching number and matching student number. Only return `TestMarks` where *both* of these conditions are met.
  - The marks should be sorted in descending order.
- studentParticipation(studentNumber: String): Integer`[]`
  - This should return an array with all of the test numbers for which the student with the passed-in student number got a mark.
  - This array should be sorted in ascending order.
- studentAverage(studentNumber: String): double
  - This should return the average mark the student recieved for this gradebook.
  - For each test number, only the best mark a student achieved will be counted.
  - Example:
    - If a student recieved the following marks:
      - Test 1: 30, 20, 10
      - Test 3: 30, 15, 3
      - Test 4: 60, 5
    - Then the student's average is 40

6 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.

7 Upload checklist

The following files should be in the root of your archive

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

8 Allowed libraries

- No imports allowed.

9 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 5 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!**