Assignment 1

This assignment is worth 15% of the total course marks.

Due date: The assignment is due at **4pm on Tuesday 9 September.** Late assignments will attract a penalty of 10% of the total assignment mark per day, and submissions more than 5 days late will not be marked. Only extensions on Medical Grounds or Exceptional Circumstances will be considered, and in those cases students need to submit an application for extension of progressive assessment form (http://www.uq.edu.au/myadvisor/forms/exams/progressive-assessment-extension.pdf) to the lecturer (email is acceptable) or the ITEE Enquiries desk on Level 4 of GPSouth Building) prior to the assignment deadline.

Resources:

The Assignment 1 zip file, available from Blackboard (under Learning Resources/Course Materials), contains the Java classes and interfaces that are mentioned in this handout. You will need those source files to complete this assignment.

Your Java Build Path will need to include the JUnit 4 Library to run the JUnit tests.

Part 1: Testing and implementing stacks and lists [3 marks]

The purpose of this part of the assignment is to familiarise you with array and linked-based implementations of linear abstract data types (ADTs), and to accustom you to using JUnit tests to uncover implementation errors.

Question 1(a) [1 mark]:

Class `part1.ArrayStack` (that is, class `ArrayStack` in the `part1` package) is an implementation of the `adt.IStack` interface.

Using the JUnit tests in `part1.test.ArrayStackTest` discover and correct the 2 ERRORS in the `part1.ArrayStack` implementation of the `adt.IStack` interface. When you find an error, insert a brief one-line "//" comment at that location, indicating where the error had been found, and why it occurred.

You may not modify the code other than to fix the 2 ERRORS and insert the required comments.

Question 1(b) [2 mark]:

Class `part1.LinkedList` is a doubly-linked list implementation of the `adt.PositionList` interface, that does not use header and trailer sentinels. It uses the `part1.Node` implementation of the `adt.Position` interface.

Using the JUnit tests in `part1.test.LinkedListTest` discover and correct the 4 ERRORS in the `part1.LinkedList` implementation of the `adt.PositionList` interface. When you find an error, insert a brief one-line "//" comment at that location, indicating where the error had been found, and why it occurred.

You may not modify the code other than to fix the 4 ERRORS and insert the required comments. Do not modify `part1.Node` in any way.

<u>Part 2:</u> Trees [12 marks for COMP3506, 15 marks for COMP7505]
The purpose of this part of the assignment is for you to gain experience writing and analysing tree algorithms.

Types of expenditure for an organisation can be classified using a hierarchical classification scheme. In Figure 1, for instance, we give a simplistic example of how expenditures for a university could be classified.

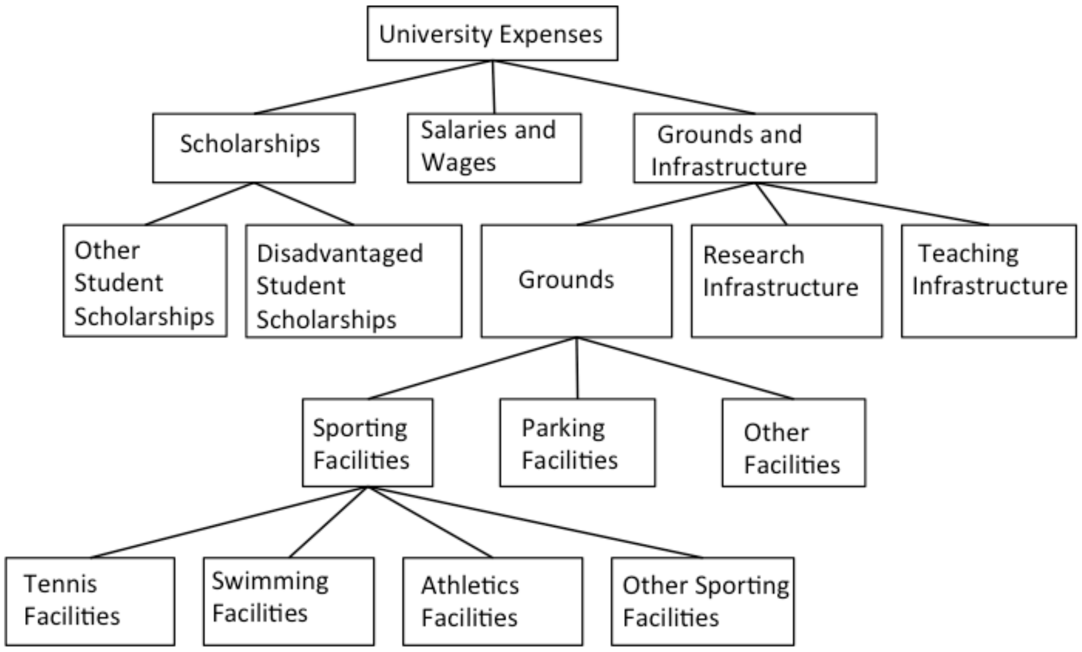

Figure 1: A simple hierarchical classification scheme of expenditures for a university.

In general, such a classification scheme is described as a tree, where each node can have either zero, one or more children. Each node represents a type of expenditure, which is a subtype of its parent, if any, and of its other ancestors. For example, in Figure 1, *Disadvantaged Student Scholarship* expenses are a subtype of *Scholarship* expenses, which is itself a subtype of all *University Expenses*. Each type of expenditure can only appear once in a classification tree. For instance, there couldn't be two different nodes in the tree with the name *Research Infrastructure*.

When donations are made to an organisation, it is typical for the donor to specify the types of expenditures that donation can be used for with respect to that organisation's expenditure classification scheme. For instance, a donor to a university using the scheme in Fig. 1 may specify that their donation may be used for *Scholarships*, *Sporting Facilities* or *Research Infrastructure.*

If such a donation can be used for a particular type of expense, then it may implicitly be used for any subtype of that expense. For example, if a donation can (w.r.t. Fig. 1) be spent on *Grounds*, then it can be used for any expense classified more specifically as either *Sporting Facilities, Parking Facilities, Other Facilities, Tennis Facilities, Swimming Facilities, Athletics Facilities,* or *Other Sporting Facilities*. Also, if a donation may be used for every subtype of an expense type, then it may also be used for that type itself. For example, a donation (w.r.t. Fig.1) that could be spent on *Tennis Facilities*, *Swimming Facilities*, *Athletics Facilities* or *Other Sporting Facilities* could be used to fund any project classified more generally as a *Sporting Facility* (since a *Sporting Facility* expense must be one of those four kinds of expenses according to the tree).

Given a set `S` of types of expenses that a donation can be spent on we refer to the **closure** of `S`, written as **closure**(`S`), to be the set of all types of expenses that that donation can be spent on. The closure is taken with respect to the relevant classification tree. For example, for the tree in Fig. 1 we have that

> **closure**({*Scholarships*, *Grounds and Infrastructure*})
>
> =
>
> **closure**({*Disadvantaged Student Scholarships*, *Scholarships*, *Athletics Facilities*, *Sporting Facilities*, *Parking Facilities*, *Other Facilities*, *Research Infrastructure*, *Teaching Infrastructure* })
>
> =
>
> {*Other Student Scholarships, Disadvantaged Student Scholarships, Scholarships, Tennis Facilities, Swimming Facilities, Athletics Facilities, Other Sporting Facilities, Sporting Facilities, Parking Facilities, Other Facilities, Grounds, Research Infrastructure, Teaching Infrastructure, Grounds and Infrastructure*}

When donors specify the types of project that their donation may be spent on they are not always as concise as they could be. For instance, a donation that could be spent on either *Disadvantaged Student Scholarships*, *Scholarships*, *Athletics Facilities*, *Sporting Facilities*, *Parking Facilities*, *Other Facilities*, *Research Infrastructure* or *Teaching Infrastructure* (w.r.t. Fig. 1), could more concisely be described as a donation that could be spent on *Scholarships* or *Grounds and Infrastructure*.

Given a set `S` of expense types from a classification tree, that is used by a donor to describe the types of expenditures that a donation may be used for, we define the ***summary*** of that set `S` to be the smallest set `X` of expense types from the classification tree such that

1) the **closure** of `S` equals the **closure** of `X`
2) for each type `y` in the **closure** of S, either `y` is in `X` or an ancestor of `y` is in `X`.

Using this terminology we have that the **summary** of `S` = {*Disadvantaged Student Scholarships*, *Scholarships*, *Athletics Facilities*, *Sporting Facilities*, *Parking Facilities*, *Other Facilities*, *Research Infrastructure*, *Teaching Infrastructure*} is `X` = {*Scholarships*, *Grounds and Infrastructure*} with respect to the classification scheme in Figure 1.

We say that a set `S` of expenditure types is **concise** with respect to a classification tree, if the **summary** of `S` equals `S`.

Some donors chose to describe what their donation may be spent on by giving a list of expenditure types that they forbid their money to be spent on.

We define the **negation** of a set of expenditure types `S` to be the concise set `X` such that the **closure** of `X` contains all of the types in the given classification scheme tree other than those types that are either in the set `S`, or are an ancestor or descendent of an element of `S`.

For example, the **negation** of the set {*Parking Facilities, Scholarships*} is the concise set {*Salaries and Wages, Sporting Facilities, Other Facilities, Research Infrastructure, Teaching infrastructure*}.

Question 2(a)[2 marks]:

Implement method `summary` in the class `part2.ExpenditureTrees` efficiently according to its specification in that file.

You may use the linear data structures in `java.util` from the Java 7 SE API in your implementation (e.g. ArrayList, LinkedList, ArrayDeque etc.), but no other libraries should be used. (It is not necessary and makes marking hard.)

You may write supporting methods, but they must be declared private and included in the `part2.ExpenditureTrees` file. Similarly, if you choose to write any supporting classes, then they must be written as private nested classes and included in that file.

To help you get your solution right, you should write you own JUnit tests in `part2.test.ExpenditureTreeTest` (some basic tests are included to get you started). Tests that you write will not be marked. We will test your implementation with our own tests.

Question 2(b)[2 marks]:

Let `n` be the number of nodes in the hierarchical classification scheme `tree` of expenditures, `m` be the number of expenditure types in the parameter list `types`, and `p` be the maximum length of any string used to represent an expenditure type in the tree. What is the worst-case time complexity and the worst-case space complexity of your `summary` method in terms of `n`, `m` and `p`? Justify your answer.

Express your answers in big-Oh notation, making the bounds as tight as possible, and simplifying your answer as much as possible.

(N.B. The space complexity of your method should be measured in terms of how much additional space your method consumes. It does not include the space already consumed by the method parameters.)

Question 3(a)[2 marks]:

Implement method `contains` in the class `part2.ExpenditureTrees` efficiently according to its specification in that file.

You may use the linear data structures in `java.util` from the Java 7 SE API in your implementation (e.g. ArrayList, LinkedList, ArrayDeque etc.), but no other libraries should be used. (It is not necessary and makes marking hard.)

You may write supporting methods, but they must be declared private and included in the `part2.ExpenditureTrees` file. Similarly, if you choose to write any supporting classes, then they must be written as private nested classes and included in that file.

To help you get your solution right, you should write you own JUnit tests in `part2.test.ExpenditureTreeTest` (some basic tests are included to get you started). Tests that you write will not be marked. We will test your implementation with our own tests.

Question 3(b)[2 marks]:

Let `n` be the number of nodes in the hierarchical classification scheme `tree` of expenditures, and `m` be the number of expenditure types in the parameter list `types`. Assuming that the length of each string used to represent an expenditure type in the tree is bound above by $\log_2 n$, what is the worst-case time complexity and the worst-case space complexity of your `contains` method in terms of `n` and `m`? Justify your answer.

Express your answers in big-Oh notation, making the bounds as tight as possible, and simplifying your answer as much as possible.

(N.B. The space complexity of your method should be measured in terms of how much additional space your method consumes. It does not include the space already consumed by the method parameters.)

Question 4(a)[2 marks]:

Implement method `negation` in the class `part2.ExpenditureTrees` efficiently according to its specification in that file.

You may use the linear data structures in `java.util` from the Java 7 SE API in your implementation (e.g. ArrayList, LinkedList, ArrayDeque etc.), but no other libraries should be used. (It is not necessary and makes marking hard.)

You may write supporting methods, but they must be declared private and included in the `part2.ExpenditureTrees` file. Similarly, if you choose to write any supporting classes, then they must be written as private nested classes and included in that file.

To help you get your solution right, you should write you own JUnit tests in `part2.test.ExpenditureTreeTest` (some basic tests are included to get you started). Tests that you write will not be marked. We will test your implementation with our own tests.

Question 4(b)[2 marks]:

Let `n` be the number of nodes in the hierarchical classification scheme `tree` of expenditures. Assuming that the length of each string used to represent an expenditure type in the tree is bound above by $\log_2 n$, what is the worst-case time complexity and the worst-case space complexity of your **`negation`** method in terms of `n`? Justify your answer.

Express your answers in big-Oh notation, making the bounds as tight as possible, and simplifying your answer as much as possible.

(N.B. The space complexity of your method should be measured in terms of how much additional space your method consumes. It does not include the space already consumed by the method parameters.)

Question 4(c)[3 marks]: COMP7505 ONLY:

Perform an experimental analysis of the worst-case running time of your method `negation`, given that the length of each string used to represent an expenditure type in the tree is bound above by $\log_2 n$, where `n` is the number of nodes `n` in the parameter `tree`.

Using the method `System.currentTimeMillis()`, you should measure the running time of your method `negation` for a suitably large range of inputs. Chose inputs for which your algorithm should exhibit worst-case behaviour. Use your results to plot a graph of measured running time against input size, where you measure the input size of the program in terms of the number of nodes `n` in the parameter `tree`. Clearly label the axes of your graph. Explain whether or not your experimental analysis confirms your theoretical analysis from Q4(b) of the worst-case running time of this method. If there is a discrepancy between your theoretical and experimental result, discuss why that might be the case.

Explain how you collected your results. As part of this explanation, include a copy of your code that you used to run the experiment (so that we can see how you did it).

Practical considerations:

If necessary, there may be some small changes to the files that are provided, up to 1 week before the deadline, in order to make the requirements clearer, or to tweak test cases. These updates will be clearly announced on the Announcements page of the course Blackboard site, and during the lectures.

For the coding tasks:

- Don't change the class names, specifications provided, or alter the method names, parameter types or return types or the packages to which the files belong.
- Don't write any code that is operating-system specific (e.g. by hard-coding in newline characters etc.), since we will batch test your code on a Unix machine.
- Your source files should be written using ASCII characters only.

Submission: Submit each of your source code files `ArrayStack.java`, `LinkedList.java`, `ExpenditureTrees.java` as well as your written answers to questions 2(b), 3(b), 4(b) [and 4(c) if you are COMP7505] in `report.pdf` electronically using Blackboard according to the exact instructions on the Blackboard website:

https://learn.uq.edu.au/

Answers to each of the questions 2(b), 3(b), 4(b) [and 4(c) if you are COMP7505] should be clearly labelled and included in file `report.pdf`.

You can submit your assignment multiple times before the assignment deadline but only the last submission will be saved by the system and marked. Only submit the files listed above. You are responsible for ensuring that you have submitted the files that you intended to submit in the way that we have requested them. You will be marked on the files that you submitted and not on those that you intended to submit. Only files that are submitted according to the instructions on Blackboard will be marked.

Evaluation: Your assignment will be given a mark according to the following marking criteria. For COMP3506 the total marks possible is 15, and for COMP7505 the total marks possible is 18. The assignment is worth 15% for both courses.

Code must be clearly written, consistently indented, and commented. Java naming conventions should be used, and lines should not be excessively long (> 80 chars). Marks will be deducted in code analysis questions if we cannot read and understand your solution to check your analysis.

Part 1

Q1(a) [1 mark]:

• solution compiles and passes ALL part1.test.ArrayStackTest tests, and location and cause of all errors correctly marked	1 mark
• otherwise or work has no academic merit	0 marks

Q1(b) [2 marks]:

• solution compiles and passes ALL part1.test.LinkedListTest tests, and location and cause of all errors correctly marked	2 marks
• otherwise or work has no academic merit	0 marks

Part 2

Q2(a)[2 marks]:

Given that the solution does not violate any restrictions (using forbidden libraries etc), [0.5 marks] will be allocated for efficiency and the remaining [1.5 marks] will be allocated based on the outcome of running test cases:

• All of our tests pass	1.5 marks
• At least 2/3 of our tests pass	1 marks
• At least 1/3 of our tests pass	0.5 marks
• Work with little or no academic merit	0 marks

Note: code submitted with compilation errors will result in zero marks in this section. Code that violates any stated restrictions will receive zero marks.

Q2(b)[2 marks]

Time analysis is correct and justified. [1 mark]
Space analysis is correct and justified. [1 mark]

A mark of 0 will be given for this part if no reasonable attempt was made to complete part (a) or the solution to part (a) is obviously incorrect or difficult to read and comprehend.

Q3(a)[2 marks]:

Given that the solution does not violate any restrictions (using forbidden libraries etc), [0.5 marks] will be allocated for efficiency and the remaining [1.5 marks] will be allocated based on the outcome of running test cases:

• All of our tests pass	1.5 marks
• At least 2/3 of our tests pass	1 marks
• At least 1/3 of our tests pass	0.5 marks
• Work with little or no academic merit	0 marks

Note: code submitted with compilation errors will result in zero marks in this section. Code that violates any stated restrictions will receive zero marks.

Q3(b)[2 marks]

Time analysis is correct and justified. [1 mark]
Space analysis is correct and justified. [1 mark]

A mark of 0 will be given for this part if no reasonable attempt was made to complete part (a) or the solution to part (a) is obviously incorrect or difficult to read and comprehend.

Q4(a)[2 marks]:

Given that the solution does not violate any restrictions (using forbidden libraries etc), [0.5 marks] will be allocated for efficiency and the remaining [1.5 marks] will be allocated based on the outcome of running test cases:

• All of our tests pass	1.5 marks
• At least 2/3 of our tests pass	1 marks
• At least 1/3 of our tests pass	0.5 marks
• Work with little or no academic merit	0 marks

Note: code submitted with compilation errors will result in zero marks in this section. Code that violates any stated restrictions will receive zero marks.

Q4(b)[2 marks]

Time analysis is correct and justified. [1 mark]
Space analysis is correct and justified. [1 mark]

A mark of 0 will be given for this part if no reasonable attempt was made to complete part (a) or the solution to part (a) is obviously incorrect or difficult to read and comprehend.

Question 4(c)[3 marks]: COMP7505 ONLY:

[2 marks] will be allocated for providing and properly graphing experimental results (for a range of well-chosen inputs) that were collected using the method you explained (including code). If it is not clear how you obtained your data, then no marks will be given for this part.

[1 mark] will be allocated for providing a valid comparison of your experimental and theoretical analysis, including an explanation of any apparent discrepancies between them.

A mark of 0 will be given for this part if no reasonable attempt was made to complete part (a) or the solution to part (a) is obviously incorrect or difficult to read and comprehend.

School Policy on Student Misconduct: You are required to read and understand the School Statement on Misconduct, available on the School's website at:

http://ppl.app.uq.edu.au/content/3.60.04-student-integrity-and-misconduct

This is an individual assignment. If you are found guilty of misconduct (plagiarism or collusion) then penalties will be applied.

If you are under pressure to meet the assignment deadline, contact the course coordinator **as soon as possible**.