Australian National University

2017 Semester 1 - Final Examination

Software Design Methodologies

(COMP2100/COMP6442)

Writing Period: 3 hour duration
Study Period: 15 minutes duration
Permitted Materials: One A4 page with notes on both sides.
Note also the standard lab tools are available including: Java, eclipse, gedit, vim, emacs, git, umbrello, dia, gcc, man, the calculator on the computer, ...
The Java API is available at file:///usr/share/doc/openjdk-7-jre-headless/api/index.html

NO calculator permitted (physical electronic device).

Please Read The Following Instructions Carefully.

This exam will be marked out of 100 and consists of 4 questions. Questions are of unequal value. The value of each question is shown in square brackets. Questions that are partitioned into parts show the number of marks given to each part within square brackets.

Students should attempt all questions. Answers must be saved into the question's directory (Q1p1, Q1p2, Q2, Q3, Q4) using the file(s) described in the question statement. Marks may be lost for giving information that is irrelevant.

Network traffic may be monitored for inappropriate communications between students, or attempts to gain access to the Internet.

The marking scheme will put a high value on clarity so, as a general guide, it is better to give fewer answers in a clear manner than to outline a greater number of less clear answers.

Question 1

Part I – Multiple Choice [30 marks]

The multiple choice questions are available via a program in the 'Q1p1' directory. This program is also used for entering your answers. To run the multiple choice program either start it by *double clicking the icon on the desktop* or by starting it in a terminal. This is done by:

- opening a Terminal,
- from the command line change directory (cd) into the Q1p1 directory, and
- run the java MultipleChoice program.

```
% cd ~/Q1p1
% java -jar MultiChoice.jar
```

Your answers are automatically saved every time you press a button. Hence once you have completed your answers you can simply exit from the program (either by pressing the window closing “x” or by selecting the “exit” menu item). Note at any time you can restart the program and change your answers. **Do not edit files in this directory.** Also you can only have one instance of the MultiChoice program running at any time.

There are 15 questions. Each answer you get correct **gains you 2 marks**, each incorrect answer **loses 0.5 marks**. Questions left unanswered neither lose nor gain marks.

If the question statement contains an underline, that is “_____”, then the selected answer should correctly complete the question statement. Some question statements may contain two underlined sections, in which case the answer has two parts separated by a comma. The first part corresponds to the first underlined section and the second part corresponds to the second underlined section.

Part II - Short Answer [10 marks]

Highest marks are gained by providing clear, concise, and short answers. Save your answers in the text file 'Q1p2answers.txt' in the directory Q1p2 (this file is already created with places to add your answers, so edit the file 'Q1p2answers.txt' and save your answers directly into it). Use your favourite editor e.g. vim, gedit etc. Please make sure that this file is saved both as you progress through the exam and before the exam ends.

i. [4 marks] In “bash” what symbols are used for “pipes” and “file redirection”? Pipes and file redirection provide a simple yet powerful way of combining different programs within a command. Provide an illustration of this. What are some limitations of “pipes” and “file redirection” when attempting to combine the efforts of different programs.

ii. [4 marks] Explain the “factory” design pattern. What is the difference between the “factory method pattern” and the “abstract factory pattern”?

iii. [3 marks] What is a finite state machine? Explain the relationship between finite state machines and Turing machines.

Question 2 [30 marks]

The aim of this question is to complete a mini software development project. This involves:

i) Understanding the requirements for a simple application,

ii) Creating a software design in UML,

iii) Implementing the application using Java,

iv) Demonstrating the use of some software engineering tools such as git, Makefiles

v) Writing JUnit test cases and testing the implementation,

vi) Writing appropriate documentation.

Your code and answers must all be included in the Q2 directory.

Suppose you are given the task of writing a program that helps a radio station manage and analyze their daily list of songs and advertisements. Each day a list of playable items is constructed. These playable items are either songs or advertisements.

Songs have: a title, the name of the band, and the number of seconds it takes to play the song (this includes any time it takes to announce the song).

Advertisements have: the number of seconds it takes to play the advertisement, the company name the advertisement comes from, and the amount of money the radio station will earn from playing the advertisement.

Part of a day's play list may look like:

- Song - title : Vertigo band : U2 time : 210 sec
- Song - title : Bad band : Michael Jackson time : 230 sec
- Advertisement - company : AGL time : 30 sec earn: $320.00
- Advertisement - company : Holden time : 20 sec earn: $210.00
- Song - title : Little Lamb band : Mary time : 210 sec
- etc...

To help build the day's playlist create methods "addSong" and "addAdvertisement" that appends the current with playlist with another song or advertisement respectively.

The radio station manager wishes to be able to analyse a day's playlist to help work out the content for the day. Your program must include 2 methods:

- 'longestSong' which returns the name of the longest song played during the day.

- 'tooManyAds' which returns true if and only if there are too many advertisements in any one hour period. (The radio station is only permitted to have at most 12mins of advertisements in any one hour period. This one hour period may start at any time in the day (so it is not just starting on the

hour))

The interface that your BasicPlayList should implement is PlayList. A "DemoPlayList" runs a simple demo of your implementation. Note these class give you a starting point for your design.

In answering this question you may constrain the problem making some reasonable assumptions about the problem. Please state any assumption you make in answering this question in the DesignSummary.txt file.

Your tasks are:

a) [10 marks] Write a basic working version of the playlist program using Java. Put your code in the Q2 directory.

b) [3 marks] Creating a software design for the program in UML. Export the diagram to a file called UML.png in the Q2 directory. Provide a short summary explaining your design in the file named DesignSummary.txt provided in the Q2 directory. You can also state assumptions you make regarding the problem in this DesignSummary.txt file.

c) [2 marks] Setup and use git within the Q2 directory for revision control and make at least 3 commits.

d) [3 marks] Write JUnit unit tests for your program.

e) [2 marks] Write a Makefile for building the program with 3 targets: make, make clean, and make rundemo.

f) [10 marks] Overall quality of submission. This includes aspects such as properly formatted code and UML diagram, enough detailed comments within code, appropriately named variables/fields/methods/classes, meaningful comments for git commits and use of appropriate design.

Hints:

- keep the design simple,
- Check that your code/makefile works from the command line within the Q2 directory.

Question 3 [20 marks]

A simple functional programing language has been created. The representation and execution of programs within this language has been implemented. What is yet to be implemented is the parser that uses a simple tokenizer to take the source code and turn it into Java objects that represents the programs. Your task for this exam question is to implement the parser.

The grammar for this language is:

```
<exp> ::= 0 | inc ( <exp> ) | dec ( <exp> ) | <variable> |
          <function name> ( <exps> ) | ( <exp> ? <exp> : <exp> )
<exps> ::= <exp> | <exp> , <exps>
<function> ::= <function name> ( <vars> ) = <exp>
<functions> ::= <function> | <function> <functions>
<vars> ::= <variable> | <variable> , <vars>
```

The Q3 directory contains the code for this project. Put your code for completing the parser in the Exp.java file in the Q3 directory. You only need to add code to complete the methods *parseExp, parseExps, parseFunction, parseFunctions*, and *parseVars*.

Question 4 [10 marks]

An AVL binary search tree for representing a set of integers has been implemented. Originally a simple binary search tree approach was used, however, its performance was poor as there would often be a sequence of ascending integers added to the set making the tree unbalanced, hence the developers thought an AVL approach would work better. Their implementation has a significant problem which meant that the original simple binary search tree approach worked better then their AVL implementation. The code for the AVL implementation is given in the Q4 directory, your task is to investigate and solve this problem with their implementation.

[3 marks] What is this significant problem? Describe and analyze the problem. Put your answer in a comment in "NETree.java" in the Q4 directory.

[7 marks] Modify the implementation fixing this problem. (Modify the files in the Q4 directory)