COMP SCI 2ME3 and SFWR ENG 2AA4 Final Examination
McMaster University

DAY CLASS, **Version 1** Dr. S. Smith
DURATION OF EXAMINATION: 2.5 hours (+ 30 minutes buffer time)
MCMASTER UNIVERSITY FINAL EXAMINATION April 28, 2021

NAME: Alan Scott
Student ID: 400263658

This examination paper includes 21 pages and 8 questions. You are responsible for ensuring that your copy of the examination paper is complete. Bring any discrepancy to the attention of your instructor.

By submitting this work, I certify that the work represents solely my own independent efforts. I confirm that I am expected to exhibit honesty and use ethical behaviour in all aspects of the learning process. I confirm that it is my responsibility to understand what constitutes academic dishonesty under the Academic Integrity Policy.

Special Instructions:

1. For taking tests remotely:
   - Turn off all unnecessary programs, especially Netflix, YouTube, games like Xbox or PS4, anything that might be downloading or streaming.
   - If your house is shared, ask others to refrain from doing those activities during the test.
   - If you can, connect to the internet via a wired connection.
   - Move close to the Wi-Fi hub in your house.
   - Restart your computer, 1-2 hours before the exam. A restart can be very helpful for several computer hiccups.
   - Use a VPN (Virtual Private Network) since this improves the connection to the CAS servers.
   - Commit and push your tex file, compiled pdf file, and code files frequently. As a minimum you should do a commit and push after completing each question.
   - Ensure that you push your solution (tex file, pdf file and code files) before time expires on the test. The solution that is in the repo at the deadline is the solution that will be graded.
   - If you have trouble with your git repo, the quickest solution may be to create a fresh clone.
2. It is your responsibility to ensure that the answer sheet is properly completed. Your examination result depends upon proper attention to the instructions.
3. All physical external resources are permitted, including textbooks, calculators, computers, compilers, and the internet.
4. The work has to be completed individually. Discussion with others is strictly prohibited.

5. Read each question carefully.
6. Try to allocate your time sensibly and divide it appropriately between the questions. Use the allocated marks as a guide on how to divide your time between questions.
7. The quality of written answers will be considered during grading. Please make your answers well-written and succinct.
8. The set $\mathbb{N}$ is assumed to include 0.

Question 1 [5 marks] What are the problems with using "average lines of code written per day" as a metric for programmer productivity?

[Provide your reasons in the itemized list below. Add more items as required. —SS]

- ALOC/day becomes innacurate for reusable code, as this code can often be copy-pasted, which is minimal effort on the coder's part, but produces the same number of lines.
- ALOC/day encourages the overabundance of code, which can usually lead to unnecessary and/or difficult to understand code.
- ALOC/day can vary depending on the program or the language (ie Java programs will normally lead to more lines than Python code due to the differnce in method inclusion, since Java programs use an extra line for a '{' character.
- ALOC/day does not take into account the actual content of the code, as coding a more complex algorithm may take fewer lines than coding something basic (for example a 30 line algorithm will likely take more time than 50 lines of System.out.println()).

Question 2 [5 marks] Critique the following requirements specification for a new cell phone application, called CellApp. Use the following criteria discussed in class for judging the quality of the specification: abstract, unambiguous, and validatable. How could you improve the requirements specification?

"The user shall find CellApp easy to use."

[Fill in the itemized list below with your answers. Leave the word in bold at the beginning of each item. —SS]

- **Abstract** - The specification specifies a given app, rather than referring to an app abstractly.
- **Unambiguous** - The specification is very ambiguous, as the term "easy to use" is not defined.
- **Validatable** - The specification is not validatable, as there is no given metric with which ease of use is to be measured.
- **How to improve** - The specification could be improved by further defining the term "easy to use" and giving a way for it to be measured.

Question 3 [5 marks] The following module is proposed for the maze tracing robot we discussed in class (L20). This module is a leaf module in the decomposition by secrets hierarchy.

Module Name find_path

Module Secret The data structure and algorithm for finding the shortest path in a graph.

[Fill in the answers to the questions below. For each item you should leave the bold question and write your answer directly after it. —SS]

A. **Is this module Hardware Hiding, Software Decision Hiding or Behaviour Hiding? Why?**

The given module is Software Decision Hiding, since it hides an algorithm and data structure within a software module.

B. **Is this a good secret? Why?**

This is not a good secret as it refers to two separate things: a data structure and an algorithm.

C. **Does the specification for maze tracing robot require environment variables? If so, which environment variables are needed?**

The robot would need some environment variables representing the current position of the robot, as well as the layout of the map (along with the start and end points).

Question 4 [5 marks] Answer the following questions assuming that you are in doing your final year capstone in a group of 5 students. Your project is to write a video game for playing chess, either over the network between two human opponents, or locally between a human and an Artificial Intelligence (AI) opponent.

[Fill in the answers to the questions below. For each item you should leave the bold question and write your answer directly after it. —SS]

A. **You have 8 months to work on the project. Keeping in mind that we usually need to fake a rational design process, what major milestones and what timeline for achieving these milestones do you propose? You can indicate the time a milestone is reached by the number of months from the project's start date.**

The milestones can be modelled after the stages of the design process, with the time assigned to each relative to the time it would take to undertake them. The first half-month would be dedicated to determining what problems would be solved, and what requirements would be met. This milestone would be met with the completion of the SRS. The next month would be dedicated to coming up with the design for the program, and would reach its milestone with the completion of the MIS document. The largest portion of the time would come from the implementation phase, which would take place over a period of three months following the completion of the design stage. The milestone for this phase would be reached with a running implementation of the game (ie the code), whether or not it functions optimally. The following three months would be assigned to verification, involving a full verification and testing of the code. This milestone is complete when a full VnV report is produced. The last half month is allocated for delivery and maintenance, a milestone which is automatically met at the end of the 8 month project time.

B. **Everything in your process should be verified, including the verification. How might you verify your verification?**

The process of verification can be verified with a nV report, which will verify that our verification is indeed verifying correct behaviour.

C. **How do you propose verifying the installability of your game?**

The installability of the game can be verified by attempting to install it on multiple different hardwares and operating systems to verify that it works on a variety of platforms. The program should also be tested by several people, in order to verfity that the install process is intuitive and easy to use.

Question 5 [5 marks] As for the previous question, assume you are doing a final year capstone project in a group of 5 students. As above, your project is to write a video game for playing chess, either over the network between two human opponents, or locally between a human and an Artificial Intelligence (AI) opponent. The questions below focus on verification and testing.

[Fill in the answers to the questions below. For each item you should leave the bold question and right your answer directly after it. —SS]

A. **Assume you have 4 work weeks (a work week is 5 days) over the course of the project for verification activities. How many collective hours do you estimate that your team has available for verification related activities? Please justify your answer.**

Given that there are 5 students per group, 4 work weeks and 5 work days per week, we can say that the the collective time for the group is 80 times the average student's daily available work time. Considering that a work day implies the student has other classes (or even work) happening concurrently, we can say that the student does not have the full day to work on the project. We can assume that the average student has about 2-3 hours that they are able and willing to work on this project per week, bring the total collective time to a lowball estimate of approximately 160 hours total for testing.

B. **Given the estimated hours available for verification, what verification techniques do you recommend for your team? Please list the techniques, along with the number of hours your team will spend on each technique, and the reason for selecting this technique.**

Given the relatively short amount of time to test a relatively complicated program,

C. **Is the oracle problem a concern for implementing your game? Why or why not? If it is a concern, how do you recommend testing your software?**

The oracle problem is a concern for the game, specifically for the AI portion of the game. Since the AI is meant to emulate a human player, it is very difficult to decide exactly what actions the AI should take, especially since the best moves in chess are not always immediately clear to an average person. To test the software with this concern, we could use prevous GM level games of chess, and see how closely the AI determines its moves relative to the real people, identifying any faults as we do so.

Question 6 [5 marks] Consider the following natural language specification for a function that looks for resonance when the input matches an integer multiple of the wavelengths 5 and 7. Provided an integer input between 1 and 1000, the function returns a string as specified below:

- If the number is a multiple of 5, then the output is "resonance 5"
- If the number is a multiple of 7, then the output is "resonance 7"
- If the number is a multiple of both 5 and 7, then the output is "resonance 5 and 7"
- Otherwise, the output is "no resonance"

You can assume that inputs outside of the range 1 to 1000 do not occur.

A. What are the sets D_i that partition D (the input domain) into a reasonable set of equivalence classes?
The sets that partition D are numbers are: $x \in \{x/7 \wedge \neg x/5\}$, $x \in \{x/5 \wedge \neg x/7\}$, $x \in \{\neg x/5 \wedge \neg x/7\}$, and $x \in \{x/5 \wedge x/7\}$, where '/' represents integer divisibility.

B. Given the sets D_i, and the heuristics discussed in class, how would you go about selecting test cases?

The test cases would be select that there would be at least one input from each of the sets D_i, as this would cover the full range of normal operation. In addition, additional test cases would be conducted for inputs at and close to 1 and 1000, in order to test the expected performance at the bounds. Since the function assumes that the numbers are within [1,1000], no other numbers need be tested.

Question 7 [5 marks] Below is a partial specification for an MIS for the game of tic-tac-toe (`https://en.wikipedia.org/wiki/Tic-tac-toe`). You should complete the specification.

[The parts that you need to fill in are marked by comments, like this one. You can use the given local functions to complete the missing specifications. You should not have to add any new local functions, but you can if you feel it is necessary for your solution. As you edit the tex source, please leave the `wss` comments in the file. You can put your answer immediately following the comment. —SS]

Syntax

Exported Constants

SIZE = 3 *//size of the board in each direction*

Exported Types

cellT = { X, O, FREE }

Exported Access Programs

Routine name	In	Out	Exceptions
init			
move	$\mathbb{N}$, $\mathbb{N}$		OutOfBoundsException, InvalidMoveException
getb	$\mathbb{N}$, $\mathbb{N}$	cellT	OutOfBoundsException
get_turn		cellT	
is_valid_move	$\mathbb{N}$, $\mathbb{N}$	$\mathbb{B}$	OutOfBoundsException
is_winner	cellT	$\mathbb{B}$	
is_game_over		$\mathbb{B}$	

Semantics

State Variables

b: boardT
Xturn: $\mathbb{B}$

State Invariant

[Place your state invariant or invariants here —SS] None

Assumptions

The init method is called for the abstract object before any other access routine is called for that object. The init method can be used to return the state of the game to the state of a new game.

Access Routine Semantics

init():

- transition:

$$Xturn, b := \text{true}, < \begin{array}{c} < \text{FREE}, \text{FREE}, \text{FREE} > \\ < \text{FREE}, \text{FREE}, \text{FREE} > \\ < \text{FREE}, \text{FREE}, \text{FREE} > \end{array} >$$

- exception: none

move(i, j):

- transition: $Xturn, b[i, j] := \neg Xturn, (Xturn \Rightarrow \text{X} | \neg Xturn \Rightarrow \text{O})$
- exception

  $exc := (\text{InvalidPosition}(i, j) \Rightarrow \text{OutOfBoundsException} | \neg \text{is\_valid\_move}(i, j) \Rightarrow \text{InvalidMoveException})$

getb(i, j):

- output: $out := b[i, j]$
- exception $exc := (\text{InvalidPosition}(i, j) \Rightarrow \text{OutOfBoundsException})$

get_turn():

- output: [Return the cellT that corresponds to the current turn —SS]
  $out := (XTurn \implies X | True \implies O)$
- exception: none

is_valid_move(i, j):

- output: $out := (b[i][j] = \text{FREE})$
- exception $exc := (\text{InvalidPosition}(i, j) \Rightarrow \text{OutOfBoundsException})$

is_winner(c):

- output: $out := \text{horizontal\_win}(c, b) \vee \text{vertical\_win}(c, b) \vee \text{diagonal\_win}(c, b)$
- exception: none

is_game_over():

- output: [Returns true if X or O wins, or if there are no more moves remaining —SS]
  $out := (isWinner(X) \vee isWinner(O) \implies True) | \forall (x : cellT | x \in b : x \neq \text{FREE}) \implies True | True \implies False$
- exception: none

Local Types

boardT = sequence [SIZE, SIZE] of cellT

Local Functions

InvalidPosition: $\mathbb{N} \times \mathbb{N} \to \mathbb{B}$

InvalidPosition$(i, j) \equiv \neg((0 \leq i < \text{SIZE}) \wedge (0 \leq j < \text{SIZE}))$

count: cellT $\to \mathbb{N}$

[For the current board return the number of occurrences of the cellT argument —SS]
count$(c) \equiv +(x : cellT | x \in b \wedge x = c : 1)$

horizontal_win : cellT $\times$ boardT $\to \mathbb{B}$

horizontal_win$(c, b) \equiv \exists(i : \mathbb{N} | 0 \leq i < \text{SIZE} : b[i, 0] = b[i, 1] = b[i, 2] = c)$

vertical_win : cellT $\times$ boardT $\to \mathbb{B}$

vertical_win$(c, b) \equiv \exists(j : \mathbb{N} | 0 \leq j < \text{SIZE} : b[0, j] = b[1, j] = b[2, j] = c)$

diagonal_win : cellT $\times$ boardT $\to \mathbb{B}$

[Returns true if one of the diagonals for the board has all of the entries equal to cellT —SS]
diagonal_win$(c, b) \equiv (\forall(i : \mathbb{N} | 0 \leq i < \text{SIZE} : b[i, i] = c)) \vee (b[2, 0] = b[1, 1] = b[0, 2] = c)$

Question 8 [5 marks] For this question you will implement in Java an ADT for a 1D sequence of real numbers. We want to take the mean of the numbers in the sequence, but as the following web-page shows, there are several different algorithms for doing this: `https://en.wikipedia.org/wiki/Generalized_mean`

Given that there are different options, we will use the strategy design pattern, as illustrated in the following UML diagram:

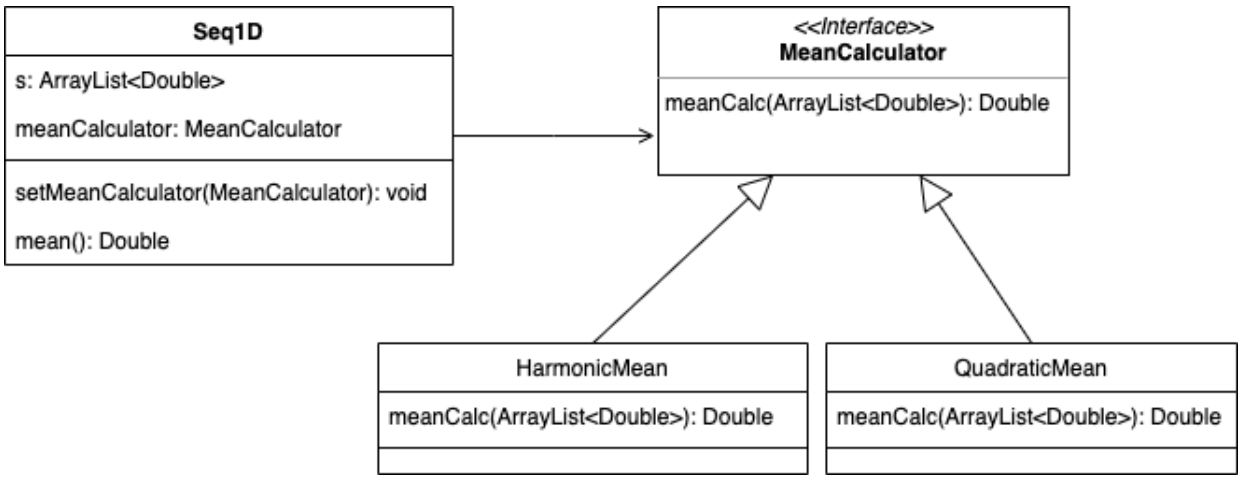

Figure 1: UML Class Diagram for Seq1D with Mean Function, using Strategy Pattern

You will need to fill in the following blank files: `MeanCalculator.java`, `HarmonicMean.java`, `QuadraticMean.java`, and `Seq1D.java`. Two testing files are also provided: `Expt.java` and `TestSeq1D.java`. The file `Expt.java` is pre-populated with some simple experiments to help you see the interface in use, and do some initial testing. You are free to add to this file to experiment with your work, but the file itself isn't graded. The `TestSeq1D.java` is also not graded. However, you may want to create test cases to improve your confidence in your solution. The stubs of the necessary files are already available in your `src` folder. The code will automatically be imported into this document when the `tex` file is compiled. You should use the provided Makefile to test your code. You will NOT need to modify the Makefile. The given Makefile will work for `make test`, without errors, from the initial state of your repo. The `make expt` rule will also work, because all lines of code have been commented out. Uncomment lines as you complete work on each part of the modules relevant to those lines in `Expt.java` file. As usual, the final test is whether the code runs on mills. You do not need to worry about doxygen comments.

Any exceptions in the specification have names identical to the expected Java exceptions; your code should use exactly the exception names as given in the spec.

Remember, your code needs to implement the given specification so that the interface behaves as specified. This does NOT mean that the local functions need to all be implemented, or that the types used internally to the spec need to be implemented exactly as given. If you do implement any local functions, please make them private. The real type in the MIS should be implemented by `Double` (capital D) in Java.

[Complete Java code to match the following specification. —SS]

Mean Calculator Interface Module

Interface Module

MeanCalculator

Uses

None

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
meanCalc	seq of $\mathbb{R}$	$\mathbb{R}$	

Considerations

meanCalc calculates the mean (a real value) from a given sequence of reals. The order of the entries in the sequence does not matter.

Harmonic Mean Calculation

Template Module inherits MeanCalculator

HarmonicMean

Uses

MeanCalculator

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
meanCalc	seq of $\mathbb{R}$	$\mathbb{R}$	

Semantics

State Variables

None

State Invariant

None

Assumptions

None

Access Routine Semantics

meanCalc(v)

- output: $out := \frac{|x|}{+(x:\mathbb{R}|x \in v:1/x)}$
- exception: none

Quadratic Mean Calculation

Template Module inherits MeanCalculator

QuadraticMean

Uses

MeanCalculator

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
meanCalc	seq of $\mathbb{R}$	$\mathbb{R}$	

Semantics

State Variables

None

State Invariant

None

Assumptions

None

Access Routine Semantics

meanCalc(v)

- output: $out := \sqrt{\frac{+(x:\mathbb{R}|x \in v : x^2)}{|x|}}$
- exception: none

Seq1D Module

Template Module

Seq1D

Uses

MeanCalculator

Syntax

Exported Types

Seq1D = ?

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
new Seq1D	seq of $\mathbb{R}$, MeanCalculator	Seq1D	IllegalArgumentException
setMaxCalculator	MaxCalculator		
mean		$\mathbb{R}$	

Semantics

State Variables

s: seq of $\mathbb{R}$
meanCalculator: MeanCalculator

State Invariant

None

Assumptions

- The Seq1D constructor is called for each object instance before any other access routine is called for that object. The constructor can only be called once. All real numbers provided to the constructor will be zero or positive.

Access Routine Semantics

new Seq1D(x, m):

- transition: s, meanCalculator := x, m
- output: *out* := *self*
- exception: *exc* := ($|x| = 0 \Rightarrow$ IllegalArgumentException)

setMeanCalculator(m):

- transition: meanCalculator := m
- exception: none

mean():

- output: *out* := meanCalculator.meanCalc()
- exception: none

Code for MeanCalculator.java

```
package src;
import java.util.ArrayList;

public interface MeanCalculator {

        public Double meanCalc(ArrayList<Double> v);

}
```

Code for HarmonicMean.java

```
package src;

import java.util.ArrayList;

public class HarmonicMean implements MeanCalculator{

        @Override
        public Double meanCalc(ArrayList<Double> v) {
                Double hsum = 0.0;
                for (Double i : v)
                        hsum += 1/i;
                return v.size()/hsum;
        }

}
```

Code for QuadraticMean.java

```
package src;

import java.util.ArrayList;

public class QuadraticMean implements MeanCalculator{

        @Override
        public Double meanCalc(ArrayList<Double> v) {
                Double qsum = 0.0;
                for (Double i : v)
                        qsum += Math.pow(i,2);
                return Math.sqrt(qsum/v.size());
        }

}
```

Code for Seq1D.java

```
package src;

import java.util.ArrayList;

public class Seq1D{
        ArrayList<Double> s;
        MeanCalculator meanCalculator;

        public Seq1D(ArrayList<Double> x, MeanCalculator m) {
                if (x.size() == 0)
                        throw new IllegalArgumentException();
                s = x; meanCalculator = m;
        }

        public void setMeanCalculator(MeanCalculator m) {
                meanCalculator = m;
        }

        public Double mean() {
                return meanCalculator.meanCalc(s);
        }
}
```