# 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: [Safwan Hossain —SS]

Student ID: [400252391 —SS]

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]

- A proper metric system for productivity is hard to create as the measurement of productivity is usually a relative value.
- Using ”average lines of codes written per day” is an ambiguous term.
- ”average lines of codes written per day” can mean different amounts for different people and projects.
- Productivity measured using lines of code is not effective because more code does not mean more productive and neither does less code mean less productive
- For example if Bob has comes up with a solution for a problem using 200 lines, while Jim solves the same problem with 50 lines of code in the same time frame. We cannot measure who was more productive using amount of code because they both solved the same problem in the same amount of time.

**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 is abstract because "easy to use" does not contain any specific details about the application and rather focuses on a key objective.
- **Unambiguous** - The specification is NOT unambiguous as the phrase "easy to use" is ambiguous. "Easy to use" is also hard to metricize because "easy" can have different criteria for different audiences. For example an "easy to use" application can be different for a teenager compared to a professional who wants details.
- **Validatable** - The specification is not validatable because there is no metric system defined that measures the simplicity of the app (or how "easy to use" it is). If a criteria for a measurement is not given then the measurement cannot be validated.
- **How to improve** - We can start with defining the target audience and using that target audience we can define the "easiness" of the app. Instead of "easy to use" we can set an unambiguous criteria that can validate the application.

**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 module is Software Decision Hiding because it is hiding the implementation for the application. Algorithm and Data structures are software decisions.

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

   - Because we want to be able to change the data structure and algorithm independent of each other and in this design we can't do that. If instead we have a module that hides the algorithm and another that hides the data structure we would be able to do that.

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

   - It requires the maze as the environment variable

**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.**

   - We can start by trying to fake the requirements with our current knowledge, and then try implmenting those requirements. After the first implementation attempt we can go back to our original requirements, modify the requirements based on discovered obstacles and then go implement the modified requirements. We can incrementally develop the software.

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

   - Mutation testing can be used for verification

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

   - We can install it across different operating systems and check if the game functions properly.

**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.**

   - We will have 400 hours in total available for verification. (5 people) * (4 hours per person) * (4 work weeks) * (5 days per week) = 400 hours.

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.**

   - We can go through a list of inputs and outputs and check if the given inputs influences the correct outputs.

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 implementation of our game. There is a very large combination of moves that can be made in the game which makes verification difficult. We can use Metamorphic Testing for verification to overcome this problem. The test cases/program will have specifications that determines how an input can change the output of the game, which can be used to verify each move.

**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?

[$D_i$ can be the set of numbers that are a multiple of 5, the set of numbers that are a multiple of 7, set of numbers that are a multiple of 5 and 7 or a set of numbers that is neither the multiple of 5 nor 7 —SS]

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

[I would test every input domain to check if they are one of the sets as stated in the previous question. I would also make sure the input domain contains numbers within the given range (1 - 1000) —SS]

**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, InvalidMove-Exception |
| 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

[$\forall(i, j : \mathbb{N} \mid 0 \leq i, j < SIZE : b[i, j] \in cellT)$ —SS]

### 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:

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

- 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: $[(Xturn \Rightarrow X \mid True \Rightarrow O)$ —SS]

- 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: $[\exists(c : cellT | \neg(c = FREE) : horizontal\_win(c) \vee vertical\_win(c) \vee diagonal\_win(c)) \vee \neg(\exists(i,j : \mathbb{N} \mid 0 \leq i,j < SIZE : b[i,j] = FREE)$ —SS]

- exception: none

**Local Types**

boardT = sequence [SIZE, SIZE] of cellT

**Local Functions**

**InvalidPosition**: $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{B}$

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

**count**: cellT $\rightarrow \mathbb{N}$

[count(c) $\equiv +(i, j : \mathbb{N} \mid 0 \leq i, j < SIZE \wedge getb(i, j) = c : 1)$ —SS]

**horizontal_win** : cellT $\times$ boardT $\rightarrow \mathbb{B}$

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 $\rightarrow \mathbb{B}$

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 $\rightarrow \mathbb{B}$

[diagonal_win$(c, b) \equiv (b[0, 0] = b[1, 1] = b[2, 2] = c \vee b[0, 2] = b[1, 1] = b[2, 0] = c)$ —SS]

**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 webpage 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:

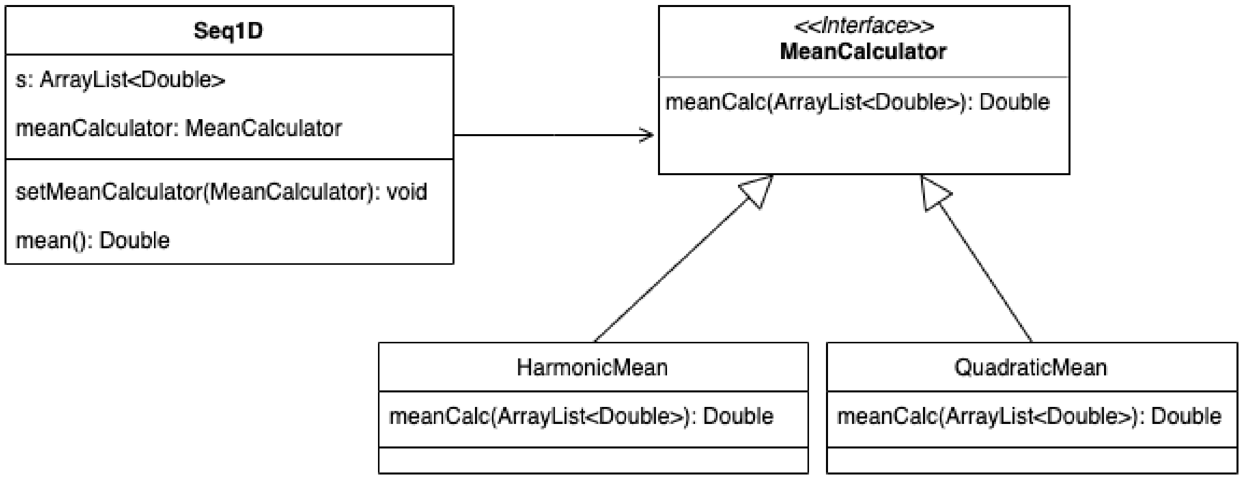
Seq1D
s: ArrayList<Double>
meanCalculator: MeanCalculator
setMeanCalculator(MeanCalculator): void
mean(): Double
<<Interface>>
MeanCalculator
meanCalc(ArrayList<Double>): Double
HarmonicMean
meanCalc(ArrayList<Double>): Double
QuadraticMean
meanCalc(ArrayList<Double>): Double

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

setMeanCalculator($m$):

- transition: meanCalculator $:= m$
- exception: none

mean():

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

## Code for MeanCalculator.java

## Code for HarmonicMean.java

## Code for QuadraticMean.java

## Code for Seq1D.java