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: [Enter your name here —SS] Aamina Hussain

Student ID: [Enter your student number here —SS] 400263927

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]

- Programmers will make sure to use more lines than necessary to make it seem like they are more productive.
- Using more lines than needed makes it more likely for errors to occur, since there are more things to keep track of and make sure are correct. This means the code would be less reliable.
- Having excess lines of code reduces the readability and understandability of the code. Other programmers who look at the code might have trouble following the code or deciphering it.
- More lines of code impacts performance. The compiler will need to compile more code, which means it will take more time to compile.

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** - It is abstract because it does not specify what types of users will use CellApp. Also, it does not tell the user how they should use the app or why they would find the app easy to use.
- **Unambiguous** - It is ambiguous because it does not explain what easy to use means, and does not say what type of user will use the app.
- **Validatable** - It is not validatable because an ambiguous specification cannot be validated. Also, easy to use is qualitative and subjective to the user, so it cannot directly be validated.
- **How to improve** - Let's define easy to use as the app loads pages quickly after it receives user input. "CellApp updates the display (a 2-D sequence of colored pixels) less that 1 second after receiving user input."

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?**

   This module is software decision hiding. This is because it actually actively does something; in this case, it finds the shortest path in a graph. This is a generic service that are likely to be used by other projects as well.

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

   No, it is not a good secret. Usually, one module contains only one secret, but in this case, this module contains both a data structure and an algorithm, which are two separate things and therefore two secrets.

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

   Yes, the specification does require environment variables. This includes the hardware interface to the robot. This means things like the motor and sensors in the robot arm. The list of environental variables given in the publication include i_mazeWalls, i_mazeStart, i_mazeEnd, i_stopButton, i_homeButton, i_backButton, i_mazeFile, o_penPos, o_penDown, o_powerOn, and o_message.

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 requirements of the project, a rough outline of the design specification, and ideas for the test cases for the modules should be completed by the end of month 1. The part of the video game where you play over the network between two human opponents should be completed by the end of month 2. The part of the video game where you play locally between a human and an AI opponent should be completed by the end of month 3. Months 4 and 5 should be used to come up with more test cases and test the code. Update the code accordingly, and come up with a way to verify the verification (test cases). Month 6 should be used to make some adjustments to the code as needed. By the end of month 8, the design specification and code should be finalized. There is some extra time left over (month 7) in case we need to redo the rational design process.

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

   We could verify the verification by getting the group of five together and doing a code walkthrough and a code inspection. We could also test the test cases using mutation testing.

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

   We can verify the installability of the game using a virtual machine. Having a virtual machine is essentially like having a computer with a clean slate. This gives you the perfect environment to test the installability of the game.

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.**

   Let's assume the average work day (or school day) is 8 hours long. Since it is our final year, we will be taking courses alongside with doing this project. Let's assume the classes for these courses take about 4 hours a day. Let's also assume the other work we do for those courses takes about 2 hours a day. This leaves us 2 hours a day for 5 days * 4 weeks, which is 2 hours a day for 20 days, for the verification activities. This is about 40 hours for one person. 40 hours/student * 5 students in the group gives an estimated amount of 200 hours for verification related activities. However, keep in mind that this is when each student in the group is working on the project individually. If the group were to work together for an hour, then that would only be 1 hour, not 5 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.**

   Blackbox testing: _ hours. I selected blackbox testing because we need to ensure we are including test cases based on our specification. This is also a technique that is relatively simple and takes less time and resources, and it ensures our code won't be missing an access routine, etc. Whitebox testing: _ hours. I selected this because this creates test cases based on the code itself, which is a bit more specific that the specification in terms of how we are implementing it. Like blackbox testing, it is also a simple technique. Stress testing: _ hours. This is so that we can test if it still functions correctly with test cases that push it to its limits. Execution testing: _ hours. This is so that we can test the performance of the code (how fast/slow it is). Mutation testing: _ hours. This is a way to test the tests. Code walkthrough and code inspection: _ hours. This is another way to test the tests, and verify the code. This would be done with the whole group.

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?**

   No, the oracle problem is not a concern for implementing this game. This is because we can easily figure out what the expected output. For example, after one player makes a move, we know what the board will look like afterwards, as well as the possibilities of moves the next player can make.

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?

[answer here - you can answer in natural language, or using mathematical notation. —SS]

The set D_0 is the set of integers ranging from 1 to 1000 that are multiples of 5. (This means there is no remainder when the integer is divided by 5.)
The set D_1 is the set of integers ranging from 1 to 1000 that are multiples of 7. (This means there is no remainder when the integer is divided by 7.)
The set D_2 is the set of integers ranging from 1 to 1000 that are in BOTH set D_0 and set D_1.
The set D_3 is the set of integers ranging from 1 to 1000 that are NOT in set D_0 or set D_1.

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

[answer here - you don't need specific test cases; your answer should characterize how all significant test cases are to be chosen. —SS]

The sets represent subdomains of the domain of integers ranging from 1 to 1000. I would choose one element to represent each subdomain. (One element of each set described above will represent each subdomain.) These elements will be used to test each subdomain. This means one element/test case from set D_0, D_1, D_2, and D_3. We can also test the boundaries/limits by having both 1 and 1000 as test cases. Since we are assuming that inputs outside of the range 1 to 1000 do not occur, we do not need to test for this case.

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]
Total number of X, O, and FREE should equal 9. There can either be an equal number of X and O cells, or one more X than O, or one more O then X (depending on who started the game).

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] $out := (Xturn \Rightarrow X | \neg Xturn \Rightarrow O)$

- 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] $out := (\text{is\_winner}(X) \vee \text{is\_winner}(O) \vee (\forall i, j : [0 ... SIZE - 1] | b[i][j] \neq \text{FREE}) \Rightarrow True | True \Rightarrow False)$

- 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 $\rightarrow \mathbb{N}$

[For the current board return the number of occurrences of the cellT argument —SS]
count$(i) \equiv (+x, y : [0...SIZE - 1] | b[x][y] = i : 1)$

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 $\rightarrow \mathbb{B}$

[Returns true if one of the diagonals for the board has all of the entries equal to cellT —SS]
diagonal_win$(c, b) \equiv (b[0, 0] = b[1, 1] = b[2, 2] = c) \vee (b[0, 2] = b[1, 1] = b[2, 0] = c)$

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:

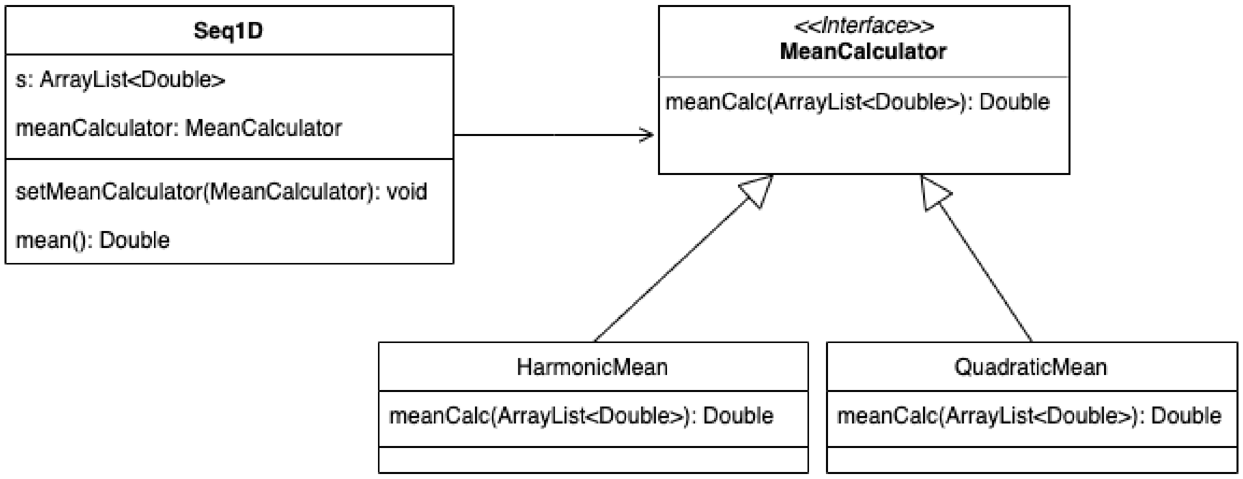
Seq1D
s: ArrayList<Double>
meanCalculator: MeanCalculator
setMeanCalculator(MeanCalculator): void
mean(): Double
<<Interface>>
MeanCalculator
meanCalc(ArrayList<Double>): Double
HarmonicMean
meanCalc(ArrayList<Double>): Double
QuadraticMean
meanCalc(ArrayList<Double>): Double

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: $\text{meanCalculator} := m$
- exception: none

mean():

- output: $out := \text{meanCalculator.meanCalc}()$
- exception: none

Code for MeanCalculator.java

```
package src;
```

Code for HarmonicMean.java

```
package src;
```

Code for QuadraticMean.java

```
package src;
```

Code for Seq1D.java

```
package src;
```