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: [Cassidy Baldin —SS]

Student ID: [400251130 —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]

- The number of lines does not necessarily mean that more work has been completed for a project, as it could just be longer due to less efficient code to accomplish the same tasks. For example you could use a for loop in its normal format, or you could use list comprehension to accomplish the same task. Both are equivalent in terms of functions but the person who created the for loop would seem like they did more work over the list comprehension person when in reality it was the same amount of work, showing that more lines does not mean more total work done in that day.
- If the worker was getting paid more for each line that wrote, then that would give the programmer incentive to make longer and less efficient code than necessary to be able to receive more pay. This would make the code much more cluttered and disorganized as there are unnecessary parts to the code that could have been cut down to make a more simple and easy to read version of the same function.
- The number of codes written per day metric also does not include a programmer cleaning up code (like they would have to do for the earlier example) where lines might not be added but instead taken out or simply modified. If the average lines written was your salary and you subtracted lines, would you have to pay the company instead. No! That's absurd. This is why this cannot be taken as a good measurement of programmer productivity, as the cleanup and maintaining of code also allows for programmers to be productive, which is very necessary for the quality of software programs and would not be included in this metric.

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** - This specification is a little abstract as it is telling the programmer an important requirement that the client has which is to make the app easier to use. There is not much detail to this statement but it is not adding the smaller and less relevant details to the specification which allows it to be abstract as it is not giving a specific example of how this requirement should be done, but is just stating the requirement that it needs to be easy to use.

- **Unambiguous** - This specification is very ambiguous as it only has one request which is to make the app easy to use but it does not give any metrics to determine how this should be done and checked. It is ambiguous because it does not tell the programmer how they want to make it easy to use and who the users of this app will be. THis information would be very helpful to the programmer following this specification as it will allow them to cater the design to be what the client is looking for, therefore. This is why this specification is ambiguous.

- **Validatable** - By using the examples above in the unambiguous section, this could make the specification more validatable, as an ambiguous specification cannot be validated. You cannot check to see if the right product has been built in terms of the requirements if there are vague requirements as there will be no right answer in this process. So this specification cannot be validatable as it is too ambiguous given the lack of metrics and other information about the user that needs to find the app easy to use. If more details were given about the users then it would be easier to make the specification validatable.

- **How to improve** - To make this specification less ambiguous and more validatable as a result, I would set more requirements for how the app was to be used. For example, since the specification wants to make it easy to use, I would specify what aspects of the design could be added to make the app easier to use. An example of this would be to include in the specification something like make icons in the app with labels for calling, voice mail, contacts etc so that way the user will be able to know that in those icons it will have that functionality, perhaps making it more clear and therefore easy to use. Another way to make the specification less ambiguous would be to make it clear in the specification who will be the primary user of this app. This way the programmer making the app will know who to gear the app towards. For example if it was to be used by teenagers, there might be able to be less explanations or labels as teens usually know what to expect out of a cell phone app from their previous experience using other similar apps. If the primary user was a grandmother who did not grow up in the technology age, then the app would need to have much more basic functions and clear and concise explanations of how to use certain functions in the app to be able to make it easier to use for that age group. This could be added into the spec to make it less ambiguous and more validatable.

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 as the secret is the structure being use and the algorithm used which can be found in the software of the implementation. The data structure that is being used in the design is a decision made by the software team when they decided how they were going to implement the maze tracing robot. The algorithm they used is also software based. This is not hiding the hardware used nor the behavior of the robot, but is hiding the software design aspects of the robot.

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

   This is a good secret as they are hiding the algorithm that finds the shortest path which is the task that the robot is trying to accomplish. If this was not hidden, then someone could go and change this algorithm and make it a slower or less efficient algorithm instead as well as change the data structure, which would cause the robot to not function the way it is supposed to. THis would be a disaster, which is why it is a good secret to have and keep.

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

   No, it does not require environment variables as all the functionality of the robot is in the software and going to the hardware that makes it perform its required actions. Environment variables are used to allow a program have have external interaction outside of its class/module (for example the plot function from one of the assignments), but since the software is interacting with the hardware it does not need to have external functionality and use environmental variables.

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.**

   Some major milestones for this project would be to first think of how the project will be implemented in terms of what data structure to use and the general sense of what methods that will be used and how the project will work which should be done in the first week or so. Then the next milestone would be to write a specification indicating in an abstract and general way how the project will complete the task of playing chess which should be done about two weeks after the project starts. The next milestone would be to write the code and get the game to "work" meaning that it has the functionality of playing the game of chess with one user (the programmer) which could take 1-2 months from the last milestone depending on the functionality. The next milestone would be to make sure the game can be played by testing it, then then testing it with two human players which should take about a week or two of testing and fixing the problems. The next milestone would be write the code for the AI to work properly which could take about 1 month after the last milestone. Then the next milestone would be to try to make the game work for the player and AI together which will be completed by a month before the project is due. Then bug fixes and other tests should be run in the remaining month before the project is finished.

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

   Verification is the ease with which the product's properties such as correctness and performance can be verified. You might be able to verify your verification by writing down requirements in which those requirements specify what the game should be able to do and what the program must be able to do by the end of the project. This checklist can be verified by other members of the group agreeing on these requirements, the supervisor agreeing that these are the requirements that need to be met, as well as other groups which may be working on similar problems to make sure that what you are trying to verify in your project is indeed what would be considered correct in terms of its performance and functionality.

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

   You might be able to verify the installibility of your game by having your group members test installing it, as well as having other people not just from your software class, but from other disciplines and programs try to install it too. Then they can give you a score out of whatever metric you decide to give you a more accurate assessment of how your program;s installibility is and can give you feedback on how to improve it.

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.**

   - answer here

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.**

   - answer here

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?**

   - answer here

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]

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]

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

[Place your state invariant or invariants here —SS]

SIZE = 3 *# The size of the game board is always 3x3*

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]

- 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: $(if\_winner(c[0]) \Rightarrow True | if\_winner(c[1]) \Rightarrow True |$ # *Where c[0] is X and c[1] is O*

  [Returns true if X or O wins, or if there are no more moves remaining —SS]

- exception: none

Local Types

boardT = sequence [SIZE, SIZE] of cellT

Local Functions

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

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

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

[For the current board return the number of occurrences of the cellT argument —SS]

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

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

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

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

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

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 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:

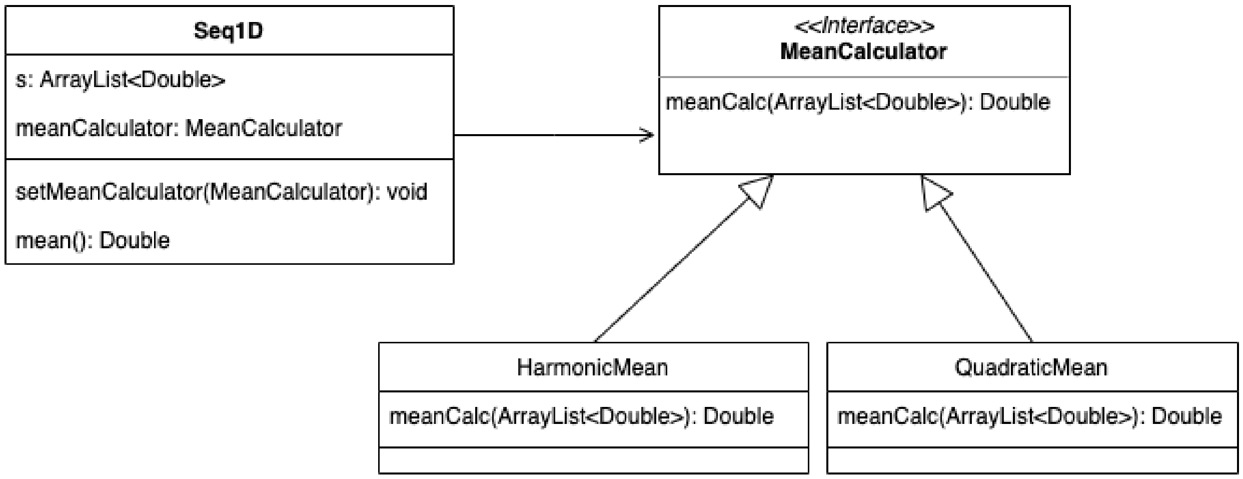
Seq1D
s: ArrayList<Double>
meanCalculator: MeanCalculator
setMeanCalculator(MeanCalculator): void
mean(): Double
<<Interface>>
MeanCalculator
meanCalc(ArrayList<Double>): Double
HarmonicMean
meanCalc(ArrayList<Double>): Double
QuadraticMean
meanCalc(ArrayList<Double>): Double

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

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

public interface MeanCalculator
{
    public Double meanCalc(ArrayList<Double> seq);
}
```

Code for HarmonicMean.java

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

public class HarmonicMean implements MeanCalculator
{
    public Double meanCalc(ArrayList<Double> v) {
        double hmean = 0;
        double recip = 0;

        for (double x : v) {
            recip = recip + 1/x;
        }

        hmean = v.size()/recip;
        return hmean;
    }
}
```

Code for QuadraticMean.java

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

public class QuadraticMean implements MeanCalculator
{
    public Double meanCalc(ArrayList<Double> v) {
        double qmean = 0;
        double quad = 0;

        for (double x : v) {
            quad = quad + x*x;
        }

        qmean = Math.sqrt(quad/v.size());
        return qmean;
    }
}
```

Code for Seq1D.java

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

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

    public Seq1D(ArrayList<Double> x, MeanCalculator m) {
        if (x.size() == 0) {
            throw new IllegalArgumentException("Sequence must be
               greater than zero");
        }
        s = x;
        meanCalculator = m;
    }

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

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

THE END.