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

DAY CLASS, **Version 1** Dr. S. Smith
DURATION OF EXAMINATION: 4 hours
MCMASTER UNIVERSITY FINAL EXAMINATION April 16, 2020

NAME: [Enter your name here —SS]

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

This examination paper includes 21 pages and 7 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.
   - Commit and push your tex file, compiled pdf file, and code files frequently.
   - 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.
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.
7. The set $\mathbb{N}$ is assumed to include 0.

Question 1 [5 marks] The software quality of reusability influences several other qualities. For each of the following qualities, state whether reusability impacts it positively or negatively and give a reason for your position: a) reliability, b) efficiency, and c) maintainability.

[Fill in the lists below —SS]

a) Reliability

- Positive or negative?
- Explanation

b) Efficiency

- Positive or negative?
- Explanation

c) Maintainability

- Positive or negative?
- Explanation

SAMPLE ANSWER

a) Reliability - either positive or negative can be justified (1 mark for plausible explanation)

- Positive explanation: Allows to build confidence because use successful code over again
- Negative explanation:
  - Reuse can propagate mistakes
  - Design for reuse can make the design too general for reliability, for instance in safety critical applications one may require that every statement be executed, which will not likely occur with a component that is reusable in several contexts

b) Efficiency (1 mark for negative, 1 mark for plausible explanation for negative)

- Negative
- To make a component reusable it needs to be general, which potentially means that problem specific efficiency enhancing simplifications cannot be employed

c) Maintainability (1 mark for positive, 1 mark for plausible explanation for positive)

- Positive
- If a component is easy to reuse, then it is easy to evolve for new needs
- Designed for reuse, like design for maintainability, considers likely changes

Question 2 [5 marks] When you insert a USB key into your computer you can read and write to the USB key in the same way as you read and write to your hard drive. What software engineering principles are being applied here? Please justify your answer.

[Fill in the itemized list below with your answers —SS]

- Software Engineering Principle 1 - explanation
- Software Engineering Principle 2 - explanation
- ...

SAMPLE ANSWER

The software engineering principles that are at work here are as follows:

- Generality - The algorithms for reading and writing to a file are written in such a way that they are not applied to a specific situation; the algorithm is general in that it works for many specific hardware contexts. (1 mark)
- Abstraction - The specific hardware is abstracted out of the problem - information hiding is used so that at the level of the user, we do not need to be concerned with the hardware. (1 mark)
- Modularity - The complex software of the operating system has been divided into modules where one module has the responsibility for reading and writing. (1 mark)
- Separation of concerns - The hardware concerns have been separated from the file handling concerns. (1 mark)

For the remaining 1 mark out of 5, award it if the answer is well written and easy to follow. If unanticipated principles are invoked, but well explained, full marks can still be awarded. You do not need to take off marks for incorrect principles, or incorrect explanations.

Question 3 [5 marks]
Critique the following specification for a monitoring system for a water tank. Use the criteria discussed in class for judging the quality of a specification (consistent, abstract, unambiguous, etc):

"If the difference between the measured water level and the target fluid level is under 5%, then set variable top_up_req to True."

[Fill in the itemized list below with your answers —SS]

- Criterion 1 - explanation
- Criterion 2 - explanation
- ...

SAMPLE ANSWER

- Not consistent (water level and fluid level): 1 mark
- Not abstract (set variable to True): 1 mark
- Ambiguous (5% relative to water level or target level?): 1 mark
- Not validatable (an ambiguous spec cannot be validated): 1 mark
- Succinct and easy to follow answer: 1 mark

The UML diagram below is used for Questions 4, 5, 6 and 7. You are not required, or expected, to do the questions in order. The organization of the next sections is intended to appear rationale for grading purposes; it is fine if you "fake it" and complete the parts in your preferred order. In particular, you will probably want to complete some test cases (Question 6) using the code `TestSeq1D.java` while you do the implementation (Question 5). You are encouraged to take some time to read all of the related questions before beginning your answers.

Consider the generic UML class diagram (Figure 1) for Seq1D. Seq1D contains a one dimensional sequence and provides the rank function. The rank function is defined for a given sequence of numerical or ordinal values. The rank of a value from the sequence is the position the value would appear in if the list were sorted in ascending order. For instance, the integer data sequence 3, -5, -2, 9 has the corresponding rank values of 2, 0, 1, 3. (As usual we start our index numbering at 0.) In the cases of ties, where more than one entry in the sequence, has the same value, it is not possible to assign the ranking uniquely. Therefore, the UML diagram uses the strategy design pattern to allow for changing the algorithm for handling ties. Further details on the rank function can be found at: `https://en.wikipedia.org/wiki/Ranking`.

For the ranking function to work the underlying data type T has to be sortable, and equality needs to be defined. Therefore, as shown in the diagram, T implements the Comparable interface and inherits Object, respectively.

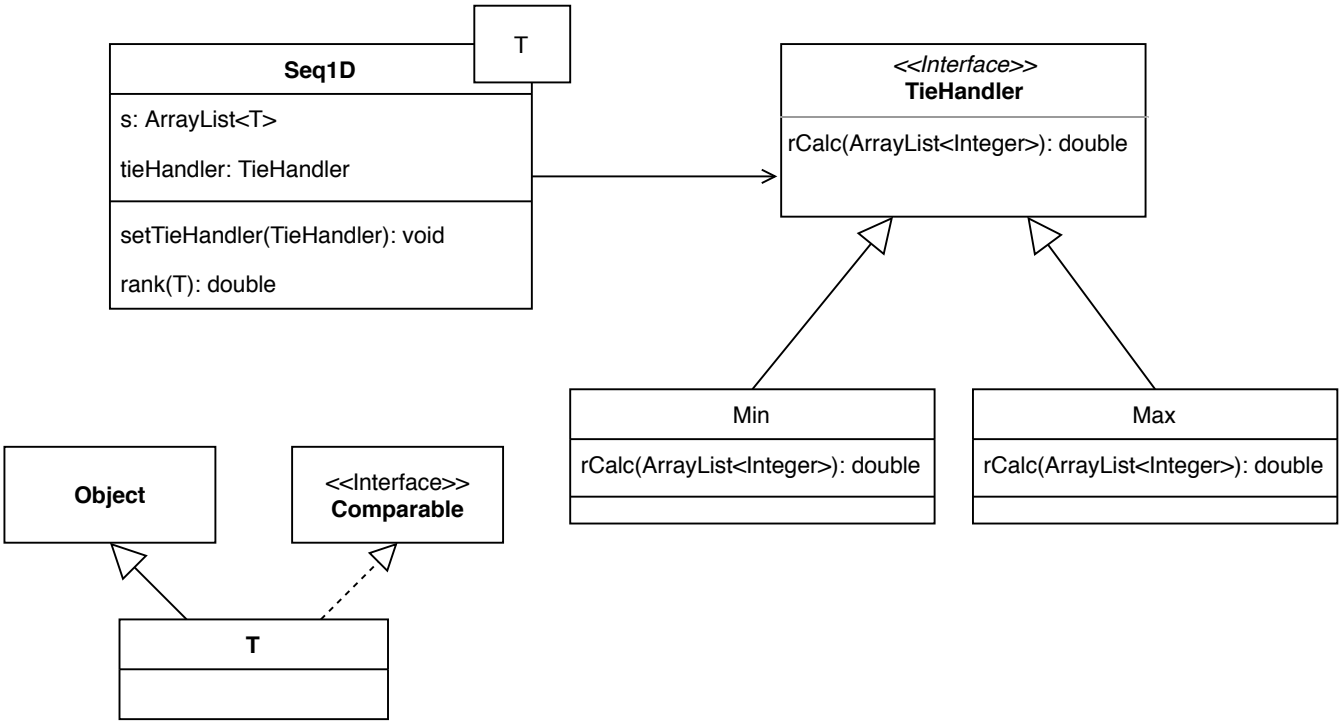

Figure 1: Generic UML Class Diagram for Seq1D with Rank Function, using Strategy Pattern for Ties

Question 4 [5 marks]
Over the next few pages is the corresponding MIS specification for the above UML diagram. Specifications for Comparable and Object are not given because they are already available in Java. As for A3, you should complete the specification where indicated by comments. (The modules that need completing are MinCalc, MaxCalc and Seq1D).

[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 fo your solution. —SS]

[As you edit the tex source, please leave the `wss` comments in the file. —SS]

GRADING
There are five spots where the spec is missing. Each one is worth 1 mark.

Tie Handler Interface Module

Interface Module

TieHandler

Uses

None

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

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

Considerations

rCalc calculates the rank (a real value) from a given sequence of indices. The order of the entries in the sequence does not matter.

Minimum Calculation for Rank with Ties Module

Module inherits TieHandler

MinCalc

Uses

TieHandler

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

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

Semantics

State Variables

None

State Invariant

None

Assumptions

None

Access Routine Semantics

rCalc(s)

- output: [The minimum element in s, where s is a sequence of natural numbers —SS] $out := m : \mathbb{N}$ such that $(m \in s) \wedge \forall(x : \mathbb{N} | x \in s | m \leq x)$
- exception: none

Maximum Calculation for Rank with Ties Module

Module inherits TieHandler

MaxCalc

Uses

TieHandler

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

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

Semantics

State Variables

None

State Invariant

None

Assumptions

None

Access Routine Semantics

rCalc(s)

- output: [The maximum element in s, where s is a sequence of natural numbers —SS] $out := m : \mathbb{N}$ such that $(m \in s) \wedge \forall(x : \mathbb{N} | x \in s | m \geq x)$
- exception: none

Generic Seq1D Module

Generic Template Module

Seq1D(T with Comparable(T))

Uses

Comparable, TieHandler

Syntax

Exported Types

[What goes here? —SS]Seq1D(T) = ?

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
new Seq1D	seq of T, TieHandler	Seq1D	
setTieHandler	TieHandler		
rank	T	$\mathbb{R}$	IllegalArgumentException

Semantics

State Variables

s: seq of T
tieHandler: TieHandler

State Invariant

None

Assumptions

- The Seq1D(T) constructor is called for each object instance before any other access routine is called for that object. The constructor can only be called once.

Access Routine Semantics

new Seq1D(S, t):

- transition: s, tieHandler := S, t
- output: $out := self$

- exception: none

setTieHandler(t):

- transition: tieHandler := t
- exception: none

rank(p):

- output: [Calculate the rank of p in the sequence, taking into account if there are ties —SS] out := tieHandler.rCalc(toSeq(indexSet(p, sort(s))))
- exception: [A p value is illegal if it does not appear in the sequence —SS] $exc := (p \notin s \Rightarrow$ IllegalArgumentException)

Local Functions

indexSet$(i, B) : T \times$ seq of T $\rightarrow$ set of $\mathbb{N}$
indexSet$(i, B) \equiv \{j : \mathbb{N} | j \in [0..|B| - 1] \wedge B[j] = i : j\}$

sort(A) : seq of $T \rightarrow$ seq of T
sort$(A) \equiv B$: seq of T, such that
$\forall(a : \mathbb{N} | a \in A : \exists(b : \mathbb{N} | b \in B : b = a) \wedge \text{count}(a, A) = \text{count}(b, B)) \wedge \forall(i : \mathbb{N} | i \in [0..|A| - 2] : B[i] \leq B[i + 1])$

count$(a, A) : T \times$ seq of T $\rightarrow \mathbb{N}$
count$(a, A) : +(x : \mathbb{N} | x \in A \wedge x = a : 1)$

toSeq(A) : set of $\mathbb{N} \rightarrow$ seq of $\mathbb{N}$
toSeq$(A) \equiv < i : \mathbb{N} | i \in A : i >$

Considerations

Implementation using Java means an equals method is available. In some cases it may be necessary to override the Java provided equals method.

Question 5 [5 marks]

[Complete Java code to match the above specification. —SS] The files you need to complete are: `TieHandler.java`, `MinCalc.java`, `MaxCalc.java`, and `Seq1D.java`. A testing file is also provided: `TestSeq1D.java`. The testing file itself isn't graded, but creating test cases will 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, without errors, from the initial state of your repo. The required imports are already given in the code. You should not make any modifications in the provided import statements. You should not delete the ones that are already there. Although you can solve the problem without adding any imports, if your solution requires additional imports, you can add them. As usual, the final test is whether the code runs on mills.

You do not need to explicitly inherit Object in Java, and you don't need to implement the Comparable interface. 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.

You do not need to worry about doxygen comments. However, you should include regular comments in the code in cases where the person marking would benefit from an explanation.

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.

GRADING

Automatically graded by unit tests

Code for TieHandler.java

```
package src;

import java.util.ArrayList;

public interface TieHandler {
    public double rCalc(ArrayList<Integer> L);
}
```

Code for MinCalc.java

```
package src;

import java.util.Collections;
import java.util.ArrayList;

public class MinCalc implements TieHandler {

    public double rCalc(ArrayList<Integer> L) {
        return Collections.min(L);
    }
}
```

Code for MaxCalc.java

```
package src;

import java.util.Collections;
import java.util.ArrayList;

public class MaxCalc implements TieHandler {

    public double rCalc(ArrayList<Integer> L) {
        return Collections.max(L);
    }
}
```

Code for Seq1D.java

```
package src;

import java.util.Collections;
import java.util.ArrayList;
import java.util.Comparator;

public class Seq1D<T extends Comparable<T>> {
   protected ArrayList<T> s;
   protected TieHandler tieHandler;

   public Seq1D(ArrayList<T> S, TieHandler t) {
      s = S;
      tieHandler = t;
   }

   public void setTieHandler(TieHandler t) {
      tieHandler = t;
   }

   public double rank(T p) {
      if (!(s.contains(p))) {
         throw new IllegalArgumentException("Argument is not contained
            in the sequence");
      }
      Collections.sort(s);
      return tieHandler.rCalc(indexSet(p, s));
   }

   private ArrayList<Integer> indexSet(T t, ArrayList<T> B) {
      ArrayList<Integer> tempLst = new ArrayList<Integer>();
      for (int k = 0; k < B.size(); k++) {
         if (t.equals(B.get(k))) {
            tempLst.add(k);
         }
      }
      return tempLst;
   }
}
```

Question 6 [5 marks]
Fill in the table below with 8 tests cases for the rank function, with T = Integer. You are only given 8 cases, so you need to select cases that are most likely to uncover errors. Each test case is summarized by the following: **s** for the state of the sequence, **tie** to identify which strategy is being used (either min or max), **p** the integer input for the rank function, **Expected** for the expected output (either an integer or exception), and **Explanation** to explain why you selected this particular test case. The explanation should identify the specific test case selection strategy employed. The strategy can be either white box (like statement coverage, edge coverage etc), or black box, or stress testing, etc. Following the table, explain why other test cases were excluded from your short list. What other tests could you have done, and what is your rationale for not emphasizing them. Your test cases should only focus on correctness, not performance.
Although it is not technically graded, you are highly recommended to complete the file `TestSeq1D.java` for testing `Seq1D.java` using the test cases in your table. More test cases in the `TestSeq1D.java` file are fine. The stub for `TestSeq1D.java`, and associated Makefile, are already available in your repo.

[Fill in this table. —SS]

#	s	tie	p	Expected	Explanation
1	[2, 1, 3, 4]	min	2	1	The tests should include a "normal" case where s is unsorted and there are no ties.
2	[ ]	min	1	Exception	White box testing path coverage suggests an empty loop for finding the index set.
3	[2, 1, 3, 4]	min	9	Exception	White box and black box testing suggest this case for statement (spec) coverage. Also for path coverage to have all False conditional branches within the loop.
4	[1, 1, 1, 1]	min	1	0	For path coverage to have a loop where the condition is True for all iterations. Could also classify as stress test.
5	[2, 1, 3, 1]	min	1	0	Repeats at beginning of list to test this boundary.
6	[2, 1, 3, 3]	min	3	2	Repeats at the end of the list to test this boundary.
7	[4, 3, 2, 1]	min	3	2	Extreme (worst) case for sorting algorithm.
8	[1, 1, 1, 1]	max	1	3	Test with max for statement/spec coverage.

[Explain below why other tests were left out. —SS]

If there were room for additional tests, then more test cases could be devoted to exercising the sorting algorithm and max. However, these tests are a low priority, since sort and max are part of the Java language and have already been extensively tests. The emphasis of the selected test cases was on the code and specification for indexSet.

GRADING

- Identify test case [] (1 mark)

- Identify test case equivalent to [1, 1, 1, 1] (1 mark)
- All selected test cases have a good rational, citing specific testing strategies (2 marks)
- Good explanation for why other tests were excluded (1 mark)

Question 7 [5 marks]

a. Another strategy for handling ties in the rank function is to average the natural numbers in the index set, rather than finding the minimum or maximum. What changes to `Seq1D.java` do you need to make to use the average strategy? Would you have to define any additional classes? If yes, describe what would be involved in writing this new code; you do not need to provide the actual code.

b. In the test cases above, you have used type T = Integer, but since the specification is written generically other types could be used. Assume that we want to add a type for students, say `StudentT`, what changes are necessary to the existing Java code you have written to use this new type? Are there any methods that `StudentT` needs to provide? If so, what are they?

c. The design for Seq1D uses an object oriented approach. How would the interface for Seq1D change if you were to adopt a functional programming approach instead? To answer this question provide below your revised details for the Exported Access Programs and State Variables.

Fill in the itemized list below

a. [Your answer here. —SS]

b. [Your answer here. —SS]

c. [Revise the following portions of the syntax section of the functional programming version of Seq1D. The spec given is for the OO version. You can make any changes you feel are necessary. —SS]

Exported Access Programs

Routine name	In	Out	Exceptions
new Seq1D	seq of T, TieHandler	Seq1D	
setTieHandler	TieHandler		
rank	T	$\mathbb{R}$	IllegalArgumentException

State Variables

s: seq of T
tieHandler: TieHandler

SAMPLE ANSWER

a. No changes are necessary to `Seq1D.java`. Average strategy written following the examples of min and max strategies. (1 mark)

b. No changes are necessary to the given Java code. StudentT needs to provide equals, compare_to and hashCode. (2 marks)

c. Replace TieHandler type with function type, or equivalent. (2 marks)

d. **Exported Access Programs**

Routine name	**In**	**Out**	**Exceptions**
new Seq1D	seq of T, seq of $\mathbb{N} \to \mathbb{R}$	Seq1D	
setTieHandler	seq of $\mathbb{N} \to \mathbb{R}$		
rank	T	$\mathbb{R}$	IllegalArgumentException

State Variables

s: seq of T
tieHandler: seq of $\mathbb{N} \to \mathbb{R}$

THE END.