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

DAY CLASS — Dr. S. Smith
DURATION OF EXAMINATION: 3 hours
MCMASTER UNIVERSITY MIDTERM EXAMINATION — March 4, 2021

NAME: [Enter your name here —SS]

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

This examination paper includes 17 pages and 4 questions. You are responsible for ensuring that your copy of the examination paper is complete. Bring any discrepancy to the attention of your instructor.

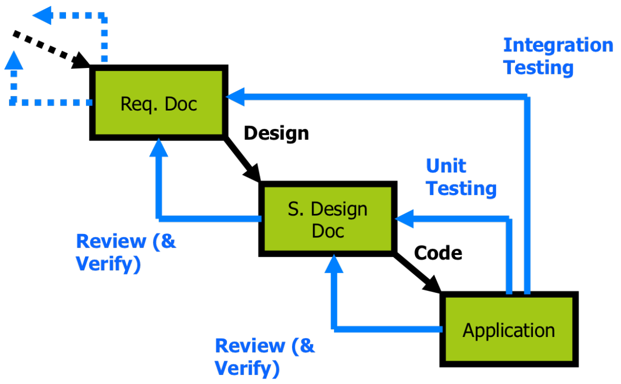
By submitting this work, I certify that the work represents solely my own independent efforts. I confirm that I am expected to exhibit honesty and use ethical behavior in all aspects of the learning process. I confirm that it is my responsibility to understand what constitutes academic dishonesty under 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 [6 marks] Parnas advocates faking a rational design process as depicted in the figure below. The faked documentation follows these steps: Requirements (SRS) → Design (MG and MIS) → Application Implementation (code) → Verification and Validation (Unit Testing, Integration Testing, Review). How are the principles of a) abstraction and b) separation of concerns applied in a rational design process? In your answer you can refer to any aspects of the process, documentation, and/or Parnas's principles.



[Fill in your answer below —SS]

a) Abstraction

The principle of abstraction is defined as the process of focusing on what is important while ignoring what is irrelevant in the project. This is also known as a special case of the principle of separation of concerns as will be explained later. This principle can be applied in a rational design process as you can use it to plan out what you are going to design/create before you actually start designing it, and allows you to take a look at the problem in a more abstract way. For example, when you are in the first stage of the faked rational design process which is Requirements, you can use this principle to look at these requirements in terms of what is most important so that you can start designing with those important details first, before worrying about the smaller things. This can allow designers to be able to make better decisions for the design, as they are more focused on what is crucial to the project, rather than getting lost in the small details that may not matter as much in the grand scheme of things. This principle can also be helpful in the design stage, as using an abstract mindset, you can make your design able to be used in many different ways or have many differing ideas brought about because you have taken the problem and instead of looking at it in a specific sense only, you can make it more broad and cause wilder ideas to come into play that may not have been thought of if not for this principle as it might have constrained the design process.

b) Separation of Concerns

The principle of separation of concerns is defined as the principle that different concerns should be isolated and considered separately from one another. This principle can be applied to the faked rational design process as you can allow designers to reduce the problem from a complex and hard to grasp problem, to many simpler and easy to work with tasks. Since different concerns are considered at different times, in the implementation step of this process, this can allow programmers to be able to split the problem into more parts that are easier to work on and that can allow for easier work between people to occur, allowing for efficient work flow in solving the tasks at hand. This principle can also be helpful when documenting requirements of a design, as this can allow you to approach the problem in a more systematic way, separating some parts from others to focus on what it is that you need to accomplish and in what order. This can make planning and scheduling easier as well, as if the project is broken up into different parts, and each part can be estimated on how long it will take and what needs to be done to complete each task, this can help the people working on the project to get things done more efficiently.

This is how the principles of abstraction and separation of concerns can be applied in a rational design process.

Consider the specification for two modules: SeqServices and SetOfInt.

Sequence Services Library

Module

SeqServicesLibrary

Uses

None

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
max_val	seq of $\mathbb{Z}$	$\mathbb{N}$	ValueError
count	$\mathbb{Z}$, seq of $\mathbb{Z}$	$\mathbb{N}$	ValueError
spices	seq of $\mathbb{Z}$	seq of string	ValueError
new_max_val	seq of $\mathbb{Z}$, $\mathbb{Z} \rightarrow \mathbb{B}$	$\mathbb{N}$	ValueError

Semantics

State Variables

None

State Invariant

None

Assumptions

- All access programs will have inputs provided that match the types given in the specification.

Access Routine Semantics

max_val(s)

- output: $out := |m| : \mathbb{N}$ such that $(m \in s) \wedge \forall(x : \mathbb{Z} | x \in s : |m| \geq |x|)$
- exception: $(|s| = 0 \Rightarrow \text{ValueError})$

count(t, s)

- output: $out := +(x : \mathbb{Z} | x \in s \wedge x = t : 1)$
- exception: $(|s| = 0 \Rightarrow \text{ValueError})$

spices(s)

- output: $out := \langle x : \mathbb{Z} | x \in s : (x \leq 0 \Rightarrow \text{“nutmeg”} | \text{True} \Rightarrow \text{“ginger”}) \rangle$
- exception: $(|s| = 0 \Rightarrow \text{ValueError})$

new_max_val(s, f)

- output: $out := \text{max\_val}(\langle x : \mathbb{Z} | x \in s \wedge f(x) : x \rangle)$
- exception: $(|s| = 0 \Rightarrow \text{ValueError})$

Set of Integers Abstract Data Type

Template Module

SetOfInt

Uses

None

Syntax

Exported Types

SetOfInt = ?

Exported Constants

None

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
new SetOfInt	seq of $\mathbb{Z}$	SetOfInt	
is_member	$\mathbb{Z}$	$\mathbb{B}$	
to_seq		seq of $\mathbb{Z}$	
union	SetOfInt	SetOfInt	
diff	SetOfInt	SetOfInt	
size		$\mathbb{N}$	
empty		$\mathbb{B}$	
equals	SetOfInt	$\mathbb{B}$	

Semantics

State Variables

s: set of $\mathbb{Z}$

State Invariant

None

Assumptions

- The SetOfInt 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 access programs will have inputs provided that match the types given in the specification.

Access Routine Semantics

new SetOfInt(x_s):

- transition: $s := \cup(x : \mathbb{Z} | x \in x_s : \{x\})$
- output: $out := self$
- exception: none

is_member(x):

- output: $x \in s$
- exception: none

to_seq():

- output: $out :=$ set_to_seq(s)
- exception: none

union(t):

- output: SetOfInt(set_to_seq(s)$||t$.to_seq())
  # in case it is clearer, an alternate version of output is:
  SetOfInt(set_to_seq($s \cup \{x : \mathbb{Z} | x \in t$.to_seq() $: x\}$))
- exception: none

diff(t):

- output: SetOfInt(set_to_seq($s \cap$ complement(t.to_seq())))
- exception: none

size():

- output: $|s|$
- exception: none

empty():

- output: $s = \varnothing$
- exception: none

equals(t):

- output: $\forall(x : \mathbb{Z} | x \in \mathbb{Z} : x \in t$.to_seq() $\leftrightarrow x \in s)$ *# this means:* t.to_seq() $= s$
- exception: none

Local Functions

set_to_seq : set of $\mathbb{Z} \rightarrow$ seq of $\mathbb{Z}$
set_to_seq$(s) \equiv \langle x : \mathbb{Z} | x \in s : x \rangle$ # *Return a seq of all of the elems in the set s, order does not matter*

complement : seq of $\mathbb{Z} \rightarrow$ set of $\mathbb{Z}$
complement$(A) \equiv \{x : \mathbb{Z} | x \notin A : x\}$

Question 2 [15 marks]
[Complete Python code to match the above specification. —SS] The files you need to complete are: `SeqServicesLibrary.py` and `SetOfInt.py`. Two testing files are also provided: `expt.py` and `test_driver.py`. The file `expt.py` is pre-populated with some simple experiments to help you see the interface in use, and do some initial test. You are free to add to this file to experiment with your work, but the file itself isn't graded. The `test_driver.py` 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.py` file. 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.
Any exceptions in the specification have names identical to the expected Python 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 where it would benefit from an explanation.
You do not need to worry about PEP8. Adherence to PEP8 will not be part of the grading.
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 by preceding the name with double underscores.

Code for SeqServicesLibrary.py

```
## @file SeqServicesLibrary.py
#  @author Cassidy Baldin
#  @brief Library module that provides functions for working with
   sequences
#  @details This library assumes that all functions will be provided
   with arguments of the expected types
#  @date 03/04/2021

def max_val(s):
    if len(s) == 0:
        raise ValueError("Sequence must have a length")

    m = 0
    for x in s:
        if abs(x) >= abs(m):
            m = x
    return m

def count(t, s):
    if len(s) == 0:
        raise ValueError("Sequence must have a length")

    count = 0
    for x in s:
        if x == t:
            count += 1
    return count

def spices(s):
    if len(s) == 0:
        raise ValueError("Sequence must have a length")

    list_of_spices = []
    for x in s:
        if x <= 0:
            list_of_spices.append("nutmeg")
        else:
            list_of_spices.append("ginger")
    return list_of_spices

def new_max_val(s, f):
    if len(s) == 0:
        raise ValueError("Sequence must have a length")
    list_f = []
    for x in s:
        list_f.append(f(x))
    print("F: ", list_f)

    return max_val(list_f)
```

Code for SetOfInt.py

```
## @file SetOfInt.py
#  @author Cassidy Baldin
#  @brief Set of integers
#  @date 03/04/2021

class SetOfInt:

    def __init__(self, xs):
        #self.s is the union of all values in the sequence xs made
           into a set of ints
        self.s = set(xs)

    def is_member(self, x):
        for i in self.s:
            if x == i:
                return True
        return False

    def to_seq(self):
        return self.__set_to_seq__(self.s)

    def union(self, t):
        #retuns a new object of the class that is a union of the two
           sets of ints
        new = self.__set_to_seq__(self.s)
        for i in range(len(t.to_seq())):
            new.append(t[i])

        #where new is the new sequence of ints including all values
           from s and t
        return SetOfInt(new)

    def size(self):
        return len(s)

    def empty(self):
        return self.s == []

    def equals(self, t):
        return False

    def __set_to_seq__(self, s):
        return list(s)
    def __complement__(self, a):
        new = {}
```

Code for expt.py

```
## @file expt.py
#  @author Spencer Smith
#  @brief This file is intended to help test that your interface
   matches the specified interface
#  @date 03/04/2021

from SeqServicesLibrary import *
from SetOfInt import *

#Exercising Sequence Services Library
print()
print("SeqServicesLibrary, max_val expt:", max_val([1, 2, -3]))
print("SeqServicesLibrary, count expt:", count(1, [1, 1, 1, 1]))
print("SeqServicesLibrary, spices expt:", spices([-5, 0, 23]))
print("SeqServicesLibrary, new_max_val expt:", new_max_val([-5, 0,
   23], lambda x: x >10))
print()

# Exercising Set of Integers
# xs = [-9, 6, 23, 21, -5]
# ys = list(xs)
# zs = {1, 21, 3, 4, 5, 6}
# # print(list(zs))
# zs.add(12)
# zs.add(6)
# print(list(zs))
# ys.append(99)
# S = SetOfInt(xs)
# print("SetOfInt, is_member expt:", S.is_member(21))
# print("SetOfInt, to_seq expt:", S.to_seq())
# S2 = SetOfInt(ys)
# S3 = S.union(S2)
# print("SetOfInt, union expt:", S3.to_seq())
# S4 = S2.diff(S)
# print("SetOfInt, diff expt:", S4.to_seq())
# print("SetOfInt, size expt:", S4.size())
# print("SetOfInt, size expt:", S4.empty())
# S5 = SetOfInt([-9, 6, 23, -5, 21])
# print("SetOfInt, equals expt:", S.equals(S5))
# print()
```

Code for test_driver.py

```
## @file test_driver.py
#  @author Your Name
#  @brief Tests implementation of SeqServicesLibrary and SetOfInt ADT
#  @date 03/04/2021

from SeqServicesLibrary import *
from SetOfInt import *

from pytest import *

## @brief Tests functions from SeqServicesLibrary.py
class TestSeqServices:

    # Sample test
    def test_sample_test1(self):
        assert True

## @brief Tests functions from SetOfInt.py
class TestSetOfInt:

    # Sample test
    def test_sample_test2(self):
        assert True
```

Question 3 [5 marks]
Critique the design of the interface for the SetOfInt module. Specifically, review the interface with respect to its consistency, essentiality, generality and minimality. Please be specific in your answer.
[Put your answer for each quality below. —SS]

- **consistency**:

  The design for SetOfInt in terms of its consistency was indeed consistent in regards to implementation and use of its methods. Consistency is defined as the requirement that use of the same project carried out by different people using the same method should produce a similar result, meaning that if one person was to create and use a method from this specification, someone else should also get a similar result when doing it as well. This specification was consistent, as it used a more formal way of defining what each method was supposed to do in terms of math and its notation. Each method also showed the naming conventions, ordering of parameters, and exceptions that it needed to have, so that anyone who would implement this specification would be able to reproduce the same design, and therefore get the same results as someone else. There is slight room for error as some implementations of this specification might be different, but if they follow the specification, then they should expect to get the same results as another implementation. Therefore, this specification is consistent.

- **essentiality**:

  The design for SetOfInt in terms of its essentiality was not essential, as it had methods that were redundant. A specification that is essential is defined as one that omits unnecessary features from a design so that it is the most useful it can be, without adding extra bulk to the design. In this specification, there are methods called `size` and `empty` where they return the size of the set and a boolean to tell if the set is empty respectively. The `empty` method is not essential, as the `size` method is already in place and can tell the user if the set is empty of not (if the size is zero it is empty). Therefore this interface as a whole is not essential, however the rest of the methods are unique making them essential methods.

- **generality**:

  The design of SetOfInt in terms of its generality is not very general, as it is only dealing with sets of integers, as opposed to a more general solution to this problem that could deal with a set of any data type. Generality is defined as a way to generalize a solution to a given specific problem, so that it can be used not just for that one specific problem, but can be used to solve others as well. This allows a program to be reused for other purposes once the main problem has been solved. In the case of this specification, since it is only dealing with sets of integers, it cannot be generalized for sets of other data types like floats or strings, it that is not in the specification (or in the name which is very specific). This means that it would not be able to be used for many other purposes as this principle suggests, meaning it is not a very general specification.

- **minimality**:

  The design of SetOfInt in terms of its minimality is indeed a minimal design as it contains unique access routines. A specification that is minimal avoids access routines with two or more potentially independent services (such as a getter that returns two state variables of the class). In the case of this specification, it is minimal as each method serves one purpose and only returns one value or object, meaning that this design is minimal.

THE END.

Question 4 [4 marks]
The module SetOfInt is for a set of integers. Please answer the following questions related to making that module generic.

a. How would you change the specification to make it generic? (Specifically what changes would you make to the given specification. You don't need to redo the spec, just summarize what changes you would need to make.)

b. What changes would you need to make to the Python implementation to make it generic for type T? (Again, you can describe and characterize the changes; you don't actually have to make them.)

c. What relational operator needs to be defined for type T to be a valid choice?

d. BONUS (1 mark) How would you specify (in the MIS) the relational operator constraint (from the previous question) on the generic type T?

[Put your answer below. —SS]

a. The changes I would make to make this specification more generic would be to change the types of values that the methods take into the class (and change the name too). For example the specification only takes in integer values in the set in the constructor. In order to make this more generic, instead of only making it take in integer values in the set, it could take in floats, strings etc, so that it could be used to perform operations of sets for all data types. This would make it more generic as it would allow this specification to be used for many other applications, rather than strictly for applications involving integers in a set.

b. The changes I would need to make to the Python implementation to make it generic for inputs of type T would be to change the `equals` method to account for

c. The relational operator that needs to be defined for type T to be a valid choice is the equal to operator, as the `equals` method is the only method that might be affected by this change from type integer to type T (as it is the only access routine that uses a relational operator). This needs to be defined as if this specification was to be used for sets of multiple types (ex. floats and strings), this would be a problem in the current implementation, as when it tries to compare the two sets, it may throw an error as the two sets are not of the same type. This could be handled

d. (BONUS)

In the MIS you could specify the relational operator constraint by adding an exception in the `equals` method to check for whether the two sets being compared are of the same type. This would not allow sets that are not of the same type to be compared. This could also be changed by a conditional statement in the MIS in the `equals` method where if the input set is of a different type from the other set, it will return False, as there is no way for the two sets to be equal if the both involve different data types. This could be done by checking the type first before doing the comparison.