CIS 120 Final Exam December 17, 2021

Name (printed): ______________________________

Pennkey (letters, not numbers): ______________________________

My signature below certifies that I have complied with the University of Pennsylvania's Code of Academic Integrity in completing this examination.

Signature: ______________________________ Date: ____________________

- Please wait to begin the exam until you are told it is time for everyone to start.
- There are 120 total points. The exam period is 120 minutes long.
- Please skim the entire exam first—some of the questions will take significantly longer than others.
- There are 14 pages in this exam.
- Please write your name and Pennkey (e.g., `stevez`) on the bottom of *every other* page where indicated.
- There is a separate appendix for reference. Answers written in the appendix will not be graded.
- Good luck!

1. **OCaml and Java Concepts** (20 points) (2 points each) Indicate whether the following statements are true or false.

a. True ☐ False ☐
In OCaml, the intended behavior of an abstract type is defined by its interface, its properties, and its implementation.

b. True ☐ False ☐
In OCaml, it is possible to use the sequencing operator `;` to execute multiple expressions and have multiple side-effects.

c. True ☐ False ☐
In the OCaml ASM, stack bindings are immutable by default whereas in the Java ASM, they are mutable by default.

d. True ☐ False ☐
In the OCaml ASM, a closure stores the required stack bindings on the heap with the function body.

e. True ☐ False ☐
In our OCaml GUI libraries, it is not possible for container widgets (like `hpair`) to handle events.

f. True ☐ False ☐
In the Java ASM, **`static`** variables are stored on the stack.

g. True ☐ False ☐
In Java, dynamic dispatch of a method invocation is guaranteed to find an appropriate method body in the class table, if the code successfully compiled.

h. True ☐ False ☐
In Java, it is possible to have multiple **`catch`** blocks, but it is possible that the order in which the blocks are written causes a compile-time error.

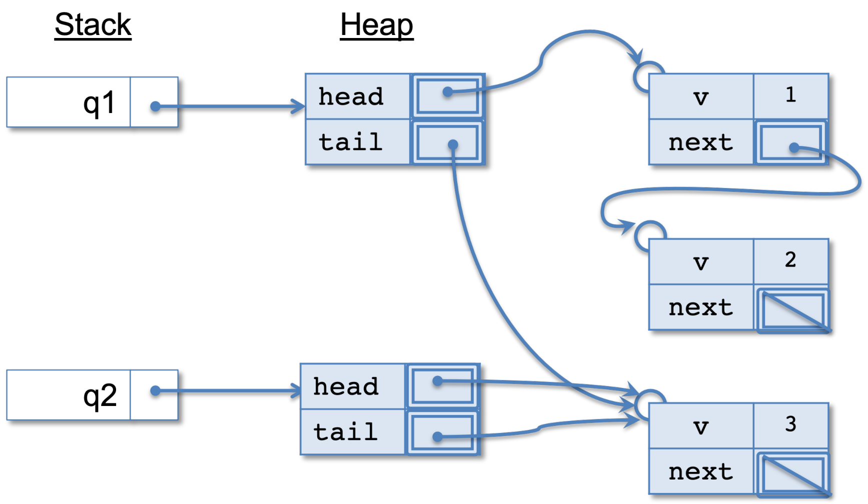
i. True ☐ False ☐
In Java, the `length` of an array is immutable.

j. True ☐ False ☐
In Java, the static type must be a subtype of the dynamic class of an object.

2. **OCaml Higher Order Functions, Queues, and ASM** (28 points total)

Recall the definition of singly-linked queues and their invariants from Homework 4. These are available in Appendix A.

Consider the Stack and Heap for the ASM shown below.



(a) (2 points) Does the queue `q1` satisfy the queue invariants?

☐ Yes ☐ No

(b) (2 points) Does the queue `q2` satisfy the queue invariants?

☐ Yes ☐ No

Next, we'll create a modified version of the higher order function `transform` (from Homework 3 and 4) that works on queues. The code is shown below:

```
let rec transform_queue (f: 'a qnode -> unit) (q: 'a queue) : unit =
  let rec loop (no: 'a qnode option) : unit =
    begin match no with
    | None -> ()
    | Some n -> let next = n.next in
                f n;
                loop next
    end
  in loop q.head
```

(c) (2 points) Is the `transform_queue` function tail recursive?

☐ Yes ☐ No

(d) (4 points) Assuming that `f` can only access its input argument (and nothing else on the stack or heap), does the `transform_queue` function always preserve the queue invariants?

☐ Yes ☐ No

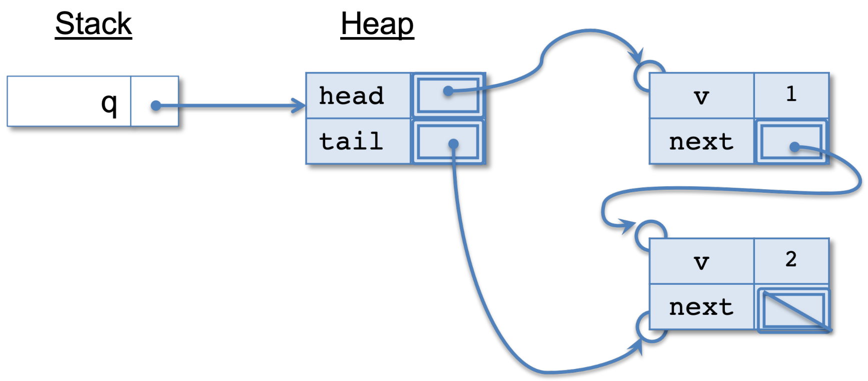
Explain why:

____

____

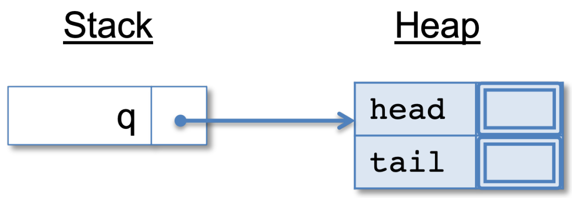
(e) (8 points) Consider the `mystery` function and the queue `q` as shown below.

```
let mystery (no: 'a qnode) : unit =
  if no.next <> None then
    let new_node = {v = no.v; next = no.next} in
    no.next <- Some new_node
```



Next, we perform the function call `transform_queue mystery q`.

Draw the ASM stack and heap after the function call is completed. For the purposes of this question, you can ignore everything on the Stack and Heap, other than what is part of the queue `q`.
(Note that `<>` is structural inequality in OCaml.)



Finally, we'll create a modified version of the higher order function `fold` (from Homework 3 and 4) that works on queues. The code is shown below:

```
let rec fold_queue (combine: 'a -> 'b -> 'b) (base: 'b) (q: 'a queue) : 'b =
  let rec loop (no: 'a qnode option) : 'b =
    begin match no with
    | None -> base
    | Some n -> combine n.v (loop n.next)
    end
  in loop q.head
```

(f) (2 points) Is the `fold_queue` function tail recursive?

☐ Yes ☐ No

(g) (4 points) Assuming that `combine` can only access its input arguments (and nothing else on the stack or heap), does the `fold_queue` function always preserve the queue invariants?

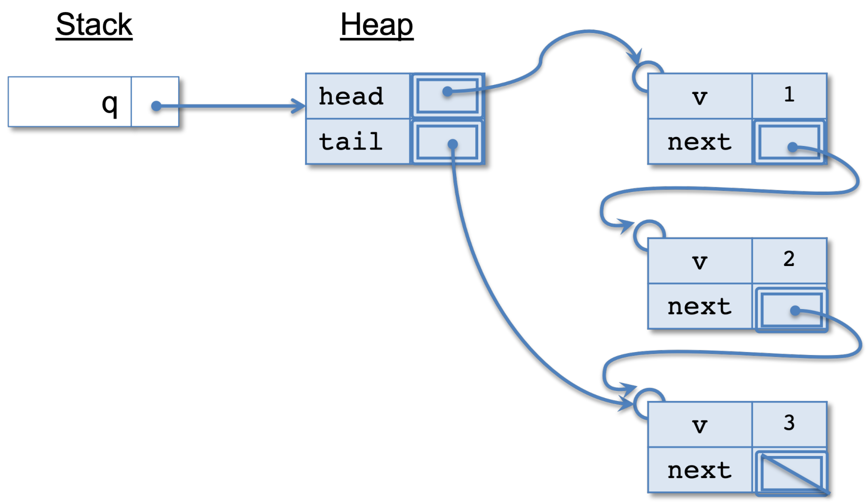
☐ Yes ☐ No

Explain why:

____

____

(h) (4 points) Consider the function call `fold_queue (fun hd acc -> hd * acc) 1 q` where the queue `q` is shown below.



What will be the result of the above function call? (1–2 sentences will be sufficient here.)

____

____

3. **Java SubTyping, Inheritance, and Exceptions** (29 points)

This problem refers to two interfaces and several classes that might be part of a program for working with the Library book series by Genevieve Cogman. You can find them in Appendix B.

(a) (2.5 points) Which of the following classes are an example of simple inheritance in Java (either explicitly or implicitly)? (Mark all that apply.)

☐ `Librarian` ☐ `Fae` ☐ `Dragon` ☐ `LiteraryDetective` ☐ `SubTyping`

(b) (2.5 points) Which lines of code are example uses of subtype polymorphism in Java? (Mark all that apply.)

☐ Line 65 ☐ Line 66 ☐ Line 68 ☐ Line 69 ☐ Line 71

(c) (2.5 points) Which lines of code are example uses of parametric polymorphism (i.e., generics) in Java? (Mark all that apply.)

☐ Line 65 ☐ Line 66 ☐ Line 68 ☐ Line 69 ☐ Line 71

(d) (3.5 points)

```
_______ vale = new LiteraryDetective();
```

Which types (there may be one or more) can be correctly used for the declaration of `vale` above?

☐ `Human` ☐ `TravelsBetweenWorlds` ☐ `Librarian` ☐ `Fae`
☐ `Dragon` ☐ `LiteraryDetective` ☐ `Object`

Which of the following lines is legal Java code that will not cause any compile-time (i.e. type checking) or run-time errors?

If it is legal code, check the "Legal Code" box and answer the questions that follow it. If it is not legal, check one of the "Not Legal" options and explain why.
You can assume each option below is independent and written after line 71 in the `main` method (as shown in the Appendix).

(e) (3 points)

```
TravelsBetweenWorlds lordSilver = new Fae();
```

☐ Legal Code
  A. The static type of `lordSilver` is ______________________________ .
  B. The dynamic class of `lordSilver` is ______________________________ .
☐ Not Legal — Will compile, but will throw an `Exception` when run
☐ Not Legal — Will not compile

Reason for not legal (in either of the two illegal cases above):

____

(f) (3 points)

```
Fae vale = new LiteraryDetective();
printName(vale);
```

☐ Legal Code

The code above will print (Choose all that apply.)

☐ "Vale (aka Sherlock Holmes)"

☐ "This method is abstract and not implemented yet."

☐ Not Legal — Will compile, but will throw an `Exception` when run

☐ Not Legal — Will not compile

Reason for not legal (in either of the two illegal cases above):

____

(g) (3 points)

```
Fae vale = new LiteraryDetective();
System.out.println(vale.travel());
```

☐ Legal Code

The code above will print (Choose all that apply.)

☐ "I need to create a portal using a book"

☐ "Only powerful Fae can travel between worlds"

☐ "I can carry a Fae or a Librarian with me"

☐ "I am Vale (aka Sherlock Holmes), but I need a Librarian to help"

☐ Not Legal — Will compile, but will throw an `Exception` when run

☐ Not Legal — Will not compile

Reason for not legal (in either of the two illegal cases above):

____

(h) (3 points)

```
travellers.add(princeKai);
System.out.println(travellers.contains(princeKai));
```

☐ Legal Code

The code above will print (Choose all that apply.)

☐ **`true`**

☐ **`false`**

☐ Not Legal — Will compile, but will throw an `Exception` when run

☐ Not Legal — Will not compile

Reason for not legal (in either of the two illegal cases above):

____

For the following two parts, we have created two new `Exception` classes:
`FaeTriedToEnterLibraryException` that is a subtype of `Exception`, but not `RuntimeException`.
`LiteraryDetectiveFoundAFaeNemesisException` that is a subtype of `RuntimeException`.
(Note that `a instanceof b` checks whether `a`'s dynamic class is a subtype of `b`.)

(i) (3 points)

```
public void travelToLibrary(TravelsBetweenWorlds t) {
    if (t instanceof Fae) {
        throw new FaeTriedToEnterLibraryException();
    } else {
      System.out.println(t.travel());
    }
}
// somewhere else
travelToLibrary(new Dragon());
```

☐ Legal Code
  The code above will print (Choose all that apply.)
  ☐ "Only powerful Fae can travel between worlds"
  ☐ "I can carry a Fae or a Librarian with me"
  ☐ "I am Vale (aka Sherlock Holmes), but I need a Librarian to help"
  ☐ It will print something other than the options shown above.
☐ Not Legal — Will compile, but will throw an `Exception` when run
☐ Not Legal — Will not compile

Reason for not legal (in either of the two illegal cases above):

___

(j) (3 points)

```
public void nemesis(LiteraryDetective detective, TravelsBetweenWorlds t) {
    if (t instanceof Fae) {
        throw new LiteraryDetectiveFoundAFaeNemesisException();
    } else {
      System.out.println(detective.getName() + " found a nemesis");
    }
}
// somewhere else
nemesis(new LiteraryDetective(), new Fae());
```

☐ Legal Code
  The code above will print (Choose all that apply.)
  ☐ "Vale (aka Sherlock Holmes) found a nemesis"
  ☐ "Irene found a nemesis"
  ☐ It will print something other than the options shown above.
☐ Not Legal — Will compile, but will throw an `Exception` when run
☐ Not Legal — Will not compile

Reason for not legal (in either of the two illegal cases above):

___

4. **Java Programming** (33 points total)

When working with sets of data, one frequently useful operation is the ability to "group" subsets of the data that are related in some way. For example, if we have a set of CIS 120 students, we might want to group them by which recitation sections they are in. Or we might want to take a set of `String`s and group them by their lengths. In this problem we will add a method `groupBy` to Java's `Set` collection functionality by implementing a class called `GroupableSet`. Appendix E contains the relevant part of the Java Docs for the interfaces we will need.

Step 1: Understand the problem: (6 points) As an example, suppose we have the set of strings shown below (written informally, not using Java notation):

```
GroupableSet<String> set1 = {"ddd", "a", "bb", "cc" , "eee"}
```

We want `set1.groupBy(classifier)` to produce the following `Map`, whose integer keys are the string lengths found in `set1`. Each key's value is a `Set` containing the subset of `set1` of strings of that length:

```
set1.groupBy(classifier) =
  key   value
  1 ->  {"a"}            // group of length 1 strings
  2 ->  {"bb", "cc"}     // group of length 2 strings
  3 ->  {"ddd", "eee"}   // group of length 3 strings
```

For simplicity, we will assume that each "group" is identified by an `Integer` and that `groupBy` takes an input, called `classifier`, which determines to which group each element of the set belongs. This `classifier` is an object that provides an `apply` method taking an element returning an `Integer` for that element's group. For the example above, `classifier.apply(s) = s.length()`, where `s` is a `String`.

(a) Suppose that we add the empty `String` `""` to the `GroupableSet set1` mentioned above. What would be the key for `""` in the resulting map after grouping by length? (Choose one.)

☐ **`null`**   ☐ `0`   ☐ `1`   ☐ `2`   ☐ there is no such key

(b) Suppose we create a different `classifier2` object such that `classifier2.apply(s) = 2`. Which of the following are true statements about the map resulting from calling `set.groupBy(classifier2)`, *i.e.*, we use this new classifier on the same `set1` from above? (Mark *all* that apply.)

☐ The map will contain the key `1`
☐ The map will contain the key `2`
☐ The map will contain the key `3`
☐ All of the elements will be in one group.
☐ There will be one element in each group.

(c) Suppose we create a `GroupableSet<Integer> set3` with elements `{0, 1, 2, 3, 4, 5}` and we create a `classifier3` such that `classifier3.apply(x) = x`. Which of the following are true statements about the map resulting from calling `set3.groupBy(classifier3)`? (Mark *all* that apply.)

☐ The map will contain the key `1`
☐ The map will contain the key `2`
☐ The map will contain the key `3`
☐ All of the elements will be in one group.
☐ There will be one element in each group.

Step 2: Design the interface (11 points)

We will implement a class called `GroupableSet<E>` that implements the `Set<E>` interface and, additionally, provides the `groupBy` method. Note (from the Java Docs in Appendix E) that the `Function<T,R>` interface indicates an object that has an `R apply(T t)` method. Based on the desired behavior above, the type of the `groupBy` method we use is as follows:

```
Map<Integer,Set<E>> groupBy(Function<E,Integer> classifier)
```

(d) The type of the `groupBy` method is an example of parametric polymorphism (a.k.a. a generic type).

☐ True ☐ False

Now consider the example uses of `groupBy` given by the classes `MainA`, `MainB`, `MainC`, and `MainD` as shown in Appendix C. Answer each question below. If the answer is "false" give a brief explanation.

(e) The code in `MainA` is well typed. (Has no compile-time errors.)

☐ True

☐ False because ______________________________

____

(f) The code in `MainB` is well typed. (Has no compile-time errors.)

☐ True

☐ False because ______________________________

____

(g) The code in `MainC` is well typed. (Has no compile-time errors.)

☐ True

☐ False because ______________________________

____

(h) The code in `MainD` is well typed. (Has no compile-time errors.)

☐ True

☐ False because ______________________________

____

(i) Which classes exhibit the use of *anonymous inner classes*? (Mark all that apply.)

☐ `MainA` ☐ `MainB` ☐ `MainC` ☐ `MainD`

Step 3: Write test cases (6 points)

Recall that, when testing a method like `groupBy`, it is often helpful to think about the *properties* that we expect to hold, especially in relation to other operations. Such properties can usually be turned into test cases.

Assume the following:

- `set` is an object of type `GroupableSet<String>`
- `classifier` is an object of type `Function<String,Integer>`
- `map` is an object of type `Map<Integer,Set<String>>` returned by `set.groupBy(classifier)`
- `k`, `k1`, and `k2` are **`int`** values (which can be used implicitly as `Integer` objects)
- None of the objects are **`null`** and none of the methods raise exceptions. (We would write other kinds of test cases for those situations.)

Each of the following properties relates `groupBy` to the `Set<E>`, `Map<Integer,Set<E>>`, and `Function<E,Integer>` interface operations. Choose one option for each blank to make the property a correct description of the intended behavior of `groupBy`.

(j) For every `o`, if _____________ then there is some `k` such that `k == classifier.apply(o)` and `map.get(k).contains(o)`.

☐ `!map.containsKey(k)`

☐ `map.containsKey(k)`

☐ `!set.contains(o)`

☐ `set.contains(o)`

(k) For every `o` and `k`, if _____________ then `set.contains(o)`.

☐ `!map.get(k).contains(o)`

☐ `map.get(k).contains(o)`

☐ `!map.containsKey(k)`

☐ `map.containsKey(k)`

(l) For every `o`, `k1`, and `k2`, if `map.get(k1).contains(o)` and `map.get(k2).contains(o)` then _____________.

☐ `k1 == k2` and `k1 == classifier.apply(o)`

☐ `k1 != k2` and `k1 == classifier.apply(o)`

☐ `k1 == k2` and `k1 != classifier.apply(o)`

☐ `k1 != k2` and `k1 != classifier.apply(o)`

Step 4: Implement the code (10 points)

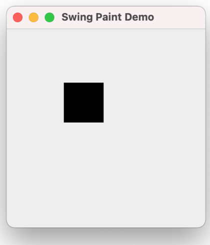
We want every `GroupableSet<E>` object to be an instance of `Set<E>`, but we don't want to have to re-implement all of the `Set` interface operations to achieve that. Recall that an *adapter* class provides default implementations of a given interface. Appendix D gives an appropriate `SetAdapter<E>` implementation of the `Set<E>` interface that we will use below as the basis for `GroupableSet`. Note that it contains the *private* field `set`.

Now complete the implementation of the `groupBy` method. You will need to create new `Set` and `Map` objects—we have imported the `TreeSet` and `TreeMap` classes for you.

```
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;
import java.util.TreeSet;
import java.util.function.Function;

public class GroupableSet<E> extends SetAdapter<E> implements Set<E> {

    public GroupableSet(Set<E> set) {
        super(set);
    }

    /* TODO: Complete this method */
    public Map<Integer,Set<E>> groupBy(Function<E,Integer> classifier) {









    }
}
```

5. **Java Swing Programming** (10 points)

The code in Appendix F implements a simple Java GUI program in which a 50x50 black box follows the mouse cursor around the window. It looks like this (the mouse cursor is not shown):



The following true/false questions concern this application and Java Swing programming in general.

a. True ☐ False ☐
The type `MyPanel` is a subtype of `Object`.

b. True ☐ False ☐
Lines 5–9 create a new object whose static type and dynamic class are, both, `Runnable`.

c. True ☐ False ☐
If we changed line 27 from `addMouseMotionListener` to `addMouseListener` then the black box would not follow the mouse cursor.

d. True ☐ False ☐
If we changed line 27 from `new MouseAdapter` to `new MouseMotionListener`, the code would still compile.

e. True ☐ False ☐
The instance variables `x` and `y`, declared on lines 23 and 24, can be declared as `final`.

f. True ☐ False ☐
There are two `@Override` annotations in the code. If we removed them, the code would still compile.

g. True ☐ False ☐
The `mouseMoved` method on line 28 is called by the Swing event loop in reaction to the user moving the mouse in the main window of the application.

h. True ☐ False ☐

The `paintComponent` method on line 42 is only invoked once, at the start of the application.

i. True ☐ False ☐

The anonymous inner class defined on line 27 implements or inherits all members of the `MouseMotionListener` interface.

j. True ☐ False ☐

The `GUI` class and the `createAndShowGUI()` method share the same stack and heap in the Java ASM.