Name:

PennKey (penn login id):

I certify that I have complied with the University of Pennsylvania's Code of Academic Integrity in completing this examination.

- There are 100 total points. The exam length is intended to be 50 minutes. The exam is asynchronous and you have 24 hours to complete the exam. You can take as much or as little time as you need as long as you submit by the deadline. We encourage you to block out a one-hour period of time without distractions in which to complete the exam.
- This exam is open notes, open computer, and open internet.
- Do not collaborate with anyone else when completing this exam.
- Submit your exam via Gradescope (see link on Piazza) before the deadline.
- This is an editable PDF document. Please fill out the entries as indicated. (Use an "X" for the checkbox answers.)
- Different PDF viewers provide different editing capabilities. On OSX, Preview is a good choice. On Windows, Adobe Reader or Adobe Acrobat should work. We recommend *not* using a web-browser embedded PDF viewer (e.g., in Chrome or Safari) since refreshing the browser tab will lose your work. In all cases, save your work periodically!
- For coding problems: aim for accurate syntax, but we will not grade your code style for indentation, spacing, etc.
- The code in the exam is available on Codio, and you're welcome to use that rather than trying to type in this code yourself. *However, using Codio for the exam is completely optional and you can do well in the exam even if you decide not to use it.*
- If you have a question or need clarification, please put a private (optionally anonymous) post on Piazza.
- There are 9 pages in the exam.
- Do not spend too much time on any one question. Be sure to recheck all of your answers.
- There is a separate Appendix for your reference. Please do not submit the Appendix.
- Good luck!

1. Mutable Queues (invariants, mutable state, ASM) (30 points total)

Appendix A contains the implementation of the singly-linked queue data structure from Homework 4, including the (correct) implementations of the create, enq, and deq operations. It also summarizes the (singly-linked) queue invariants.

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
- (c) (7 points) Starting from an empty Stack and Heap, write code that will produce the ASM shown above. You can create additional stack bindings not displayed in the ASM diagram above, but you cannot create additional bindings in the heap.

For this part, you *must only* create or modify records directly and *cannot* use any of the provided queue operations (create, enq, and deq).

If it's not possible to create the ASM with these constraints, explain why.

(d) (7 points) Starting from an empty Stack and Heap, write code that will produce the ASM shown above. You can create additional stack bindings not displayed in the ASM diagram above, but you cannot create additional bindings in the heap.

For this part, you *cannot* create or modify records directly and *must only* use the provided queue operations (create, enq, and deq).

If it's not possible to create the ASM with these constraints, explain why.

(e) (12 points) Given a valid queue, write a delete_alternate function that deletes every *alternate* element from that queue after the head. (That is, it would delete the 2nd, 4th, 6th, etc. elements from the queue.). E.g., if the queue contains the elements 1 -> 2 -> 3 -> 4 where the head is at 1 and the tail is at 4, the resulting queue will be 1 -> 3 where the head is still at 1 and the tail is now at 3.

The queue should be updated in-place and satisfy the queue invariants at the end. The function should be tail-recursive.

let delete_alternate (q: 'a queue) : unit =
 (* Hint: Do you need a recursive helper function? *)

2. OCaml Programming: Encapsulation and Objects (23 points total)

Recall the definition of OCaml's generic reference type:

```
type 'a ref = { mutable contents : 'a }
```

For this question, you'll implement two versions of the counter type defined below:

```
type counter = {
  get : unit -> int;
  incr : unit -> unit;
}
```

(a) (9 points) Version 1 with NO encapsulation

The first implementation will have no encapsulation for the internal state. Implement this version such that the code that follows will print out values as indicated.

```
let make_counter_1 () : counter =
```

```
let ctr1 = make_counter_1 ()
(* this line will print 0 *)
;; print_endline (string_of_int (ctr1.get()))
;; ctr1.incr()
;; ctr1.incr()
(* this line will print 2 *)
;; print_endline (string_of_int (ctr1.get()))
```

Write a line of code that modifies the internal state, *without calling* get or incr, such that the following line *must* print a value other than 2.

```
(* some code unrelated to type counter (i.e., not get or incr) *)
(* this line must print a value other than 2 *)
```

```
;; print_endline (string_of_int (ctr1.get()))
```

(b) (14 points) Version 2 with encapsulated coupled counters

The second implementation will have encapsulation for the internal state as follows:

- It will return 2 *counters*, say ctr1 and ctr2, and a function.
- The two counters will maintain their independent internal counts.
- The two counters are "coupled", which means that only one counter is unlocked at any given time. If a counter is unlocked, when incr is called on it, it will increment the value. If a counter is locked, when incr is called on it, it will do nothing.
- The function returned (i.e., the third item in the tuple) has type unit -> unit. When it is called, it will toggle which counter is locked and which is unlocked. E.g., if ctr1 is locked (and therefore, ctr2 is unlocked), calling toggle will result in ctr1 being unlocked and ctr2 being locked.
- The default initial values for the counts for both counters is **0**.
- The default locking state is that ctr1 is locked and ctr2 is unlocked.

```
(* Version 2 with encapsulated coupled counters *)
let make_counter_2 () : counter * counter * (unit -> unit) =
```

```
let ctr1, ctr2, toggle = make_counter_2 ()
```

```
(* this line will print 0 since that's the initial value for ctr1 *)
;; print_endline (string_of_int (ctr1.get()))
(* this line will print 0 since that's the initial value for ctr2 *)
;; print_endline (string_of_int (ctr2.get()))
```

```
;; ctr1.incr()
;; ctr2.incr()
(* this line will still print 0 since ctr1 is locked by default *)
;; print_endline (string_of_int (ctr1.get()))
(* this line will print 1 since ctr2 is unlocked by default *)
;; print_endline (string_of_int (ctr2.get()))
(* ctr1 is now unlocked; ctr2 is now locked *)
;; toggle()
;; ctr1.incr()
;; ctr2.incr()
(* this line will now print 1 since ctr1 is unlocked *)
;; print_endline (string_of_int (ctr1.get()))
(* this line will still print 1 since ctr2 is locked *)
;; print_endline (string_of_int (ctr2.get()))
```

3. Inheritance and Subtyping in Java (22 points total)

The following questions refer to the Java interfaces and class definitions shown in Appendix B. Consider the class below, whose main method has some expressions omitted. (Assume all the code from the appendix is in the same source directory as this program, so there is no need to use import anywhere.)

```
public class ExampleCode {
 1
2
       public static void useFonJ(F obj, J tgt) {
3
            obj.invokeMethod(tgt);
4
       }
5
6
       public static void main(String[] args) {
8
            F f = /* omitted code X */
9
            J j = /* omitted code Y */
10
            useFonJ(f, j);
11
       }
12
```

(a) (10 points) Suppose that when you run this ExampleCode program it raises a NullPointerException. Where in the program might the exception originate dynamically? (That is, on what line in which class will the exception be thrown when the program is run?) There are *at most five* possible sources of such an error. For each possibility, fill in the class and line number that throws the NullPointerException. Describe what the omitted code x or Y of ExampleCode would have to do to lead to that exception being raised. (Leave excess templates blank.)

```
Class
                     , line
                                    throws NullPointerException when
omitted code
                  has the behavior of
Class
                     , line
                                    throws NullPointerException when
omitted code
                  has the behavior of
Class
                     . line
                                    throws NullPointerException when
omitted code
                  has the behavior of
Class
                     , line
                                    throws NullPointerException when
omitted code
                  has the behavior of
Class
                     , line
                                    throws NullPointerException when
omitted code
                  has the behavior of
```

(b) (3 points) Assuming the code as given in the appendix, which types could be put in each blank below so that the variable declaration is statically well-typed?

I y = new	();				
I	J	С	D	E	F
J x = new	();				
I	J	С	D	E	F
D x = new	();				
I	J	С	D	E	F

- (c) (9 points) Suppose you wanted to add a new class G to the code from the appendix such that:
 - the statement (new G()).method1(); successfully compiles and prints the string "Class C's method1!" when run
 - the statement (new F()).invokeMethod(new G()); successfully compiles and prints the string "Class G's method2!" when run

There are many ways to achieve this goal! List three different ways that this can be accomplished using *simple inheritance*. In each case, indicate which superclass G **extends** (use only Object plus the classes available in the appendix); indicate which interface(s) G *must* explicitly implement (leave it blank if none); and for each method method1 and method2 indicate whether, for your choices above, G *must* explicitly provide the method.

class	G extends		implements
G	must	may not	provide method1()
G	must	may not	provide method2()
class	G extends		implements
G	must	may not	provide method1()
G	must	may not	provide method2()
class	G extends		implements
G	must	may not	provide method1()
G	must	may not	provide method2()

4. Java Array Programming (25 points)

}

Write a function hPair, that takes in two *rectangular* arrays of type int[][] (i.e. every inner array has the same length) and returns a new array that places the arrays horizontally adjacent to each other. Any "open" space left by the difference in array sizes should be filled with 0's. Pictorially, if a and b are as shown below, then the results of using hPair in the two orders are as shown.

	a b		hI	hPair(a , b)				hPair(b,a)						
1	1	1	3	3	1	1	1	3	3	3	3	1	1	1
2	2	2	4	4	2	2	2	4	4	4	4	2	2	2
			5	5	0	0	0	5	5	5	5	0	0	0

You may assume that the input arrays a and b are not null, that they are rectangular, and that they contains no null sub-arrays. Note that a[i] refers to the row i in a. If a has no rows, then hPair should return a clone of b (and vice versa). You may find it useful to use the static methods Math.max and Math.min, or the array clone method.

```
public int[][] hPair(int[][] a, int[][] b) {
```