Name: ____________________________

Pennkey: ____________________________

CIS 120 Midterm II

November 2011

1	/20
2	/20
3	/20
4	/20
5	/20
Total	/100

- Do not begin the exam until you are told to do so.
- You have 50 minutes to complete the exam.
- There are 100 total points.
- There are 9 pages in this exam.
- Make sure your name and Pennkey (a.k.a. username) is on the top of this page.

1. **Abstract Stack Machine (20 points)**

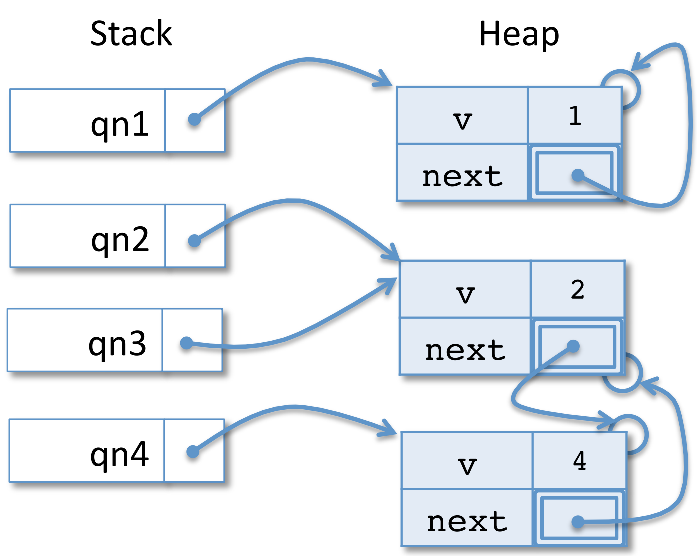
Consider the following OCaml program that uses the queue types seen in Lecture and HW05:

```
(* Mutable queues, as defined in class. *)
type 'a queuenode = { v: 'a;
                      mutable next: 'a queuenode option}

type 'a queue = { mutable head : 'a queuenode option;
                  mutable tail : 'a queuenode option }

let qn1 : int queuenode = {v = 1; next = None;}
let qn2 : int queuenode = {v = 2; next = Some qn1;}
let qn3 : int queuenode = qn2
let qn4 : int queuenode = {v = 4; next = Some qn3;}
;; qn1.next <- qn3.next
;; qn3.next <- Some qn4
(* HERE *)
```

Complete the diagram below of the state of the stack and heap parts of the ASM when the program reaches the point marked `(* HERE *)` in the program above. Note that you may need to add "`Some` bubbles" in the appropriate places, and that if you are simulating the execution of the program, you might have to *erase* pointers at times (or, if using ink, mark the erased pointers *clearly* with an X)—you should show only the final state! The Appendix of the exam contains an example of the stack and heap diagram for a similar OCaml program.



answer:

2. Mutable State, Queues, and Iteration (20 points total)

This problem uses the `'a queue` and `'a queuenode` datatypes from Problem 1. Recall that the *queue invariant* for an `'a queue` value `q` is:

- `q.head` and `q.tail` are either both `None`, or
- `q.head` and `q.tail` both point to `Some` nodes, and
  - `q.tail` is reachable by following `next` pointers from `q.head`
  - `q.tail`'s `next` pointer is `None`

a. (10 points) Complete the definition of the function `insert_after`, given below. Its inputs are `q`, an `'a queue`, `qn` an `'a queuenode` value *that is assumed to be a node in* `q`, and a value `x`. The function should modify `q` *in place* to insert a new node with value component `x` after the node `qn`. Note that this function is *not* recursive. Make sure that it preserves the queue invariant.

answer:

```
(* Modify q by inserting a node with value x after the node qn:

   q satisfies the queue invariant
   qn is a node assumed to be in q
   x is the value to insert after qn

   After the insertion, q must still satisfy the queue invariant.
*)
let insert_after (q:'a queue) (qn:'a queuenode) (x:'a) : unit =
  begin match qn.next with
    | None -> (* qn must be last by Queue invariant *)
        let new_node = {v = x; next = None} in
          qn.next <- Some new_node;
          q.tail <- Some new_node
    | Some n -> (* qn must not be last *)
        let new_node = {v = x; next = Some n} in
          qn.next <- Some new_node
  end
```

or

```
let insert_after (q:'a queue) (qn:'a queuenode) (x:'a) : unit =
  let new_node : 'a queuenode = {v = x; next = qn.next} in
  qn.next <- Some new_node;
  if (new_node.next = None) then
    q.tail <- Some new_node
```

b. (10 points) Complete the definition of the function `insert_after_first`, which, given a queue `q` and values `x` and `a`, inserts a node with value `x` after the *first occurrence* of a node with value `a` that is reachable from `q.head`. If `q` does not contain `a`, then `x` is not inserted.

For example: if `(to_list q)` $\Longrightarrow$ `[1;2;3;2;4]`, where `1` is the head and `4` is the tail, running

```
;; insert_after_first q 17 2
```

should yield: `(to_list q)` $\Longrightarrow$ `[1;2;17;3;2;4]`.

Your solution should use the function `insert_after` defined above. Write the `loop` function in *tail recursive* form.

answer:

```
let insert_after_first (q: 'a queue) (x:'a) (a:'a) : unit =
  let rec loop (qno : 'a queuenode option) : unit =
   begin match qno with
     | None -> ()
     | Some qn ->
        if qn.v = a then insert_after q qn x
        else loop qn.next
   end
  in
   loop q.head
```

3. Object Encodings, Transition to Java (20 points total)

Suppose that we wanted to port the OCaml GUI project to Java. Assume that we have already implemented `Gctx` and `Event` classes, the relevant parts of which are shown below:

```
public class Gctx {
  public Gctx translate(int dx, int dy) { /*...omitted...*/ }
  public void drawString(String s, int x, int y) { /*...omitted...*/ }
  public void textWidth(String s) { /*...omitted...*/ }
  public void textHeight(String s) { /*...omitted...*/ }
  ...
}

public class Event { /*...omitted...*/ }
```

Recall these OCaml type definitions from the `widget` library:

```
type t = {
  repaint: Gctx.t -> unit;
  handle: Gctx.t -> Gctx.event -> unit;
  size: Gctx.t -> int * int
}

type label_controller = {
  set_label: string -> unit
}
```

a. (8 points) A Java **interface** called `LabelController` is the translation of the OCaml type `label_controller`. It specifies the signature of the `set_label` method like this:

```
interface LabelController {
  public void setLabel(String s);
}
```

Do the same thing for the `t` type: Define a Java **interface** called `Widget` that is the translation of the `t` datatype given above. Because Java does not have tuples, replace the single `size` method with *two* methods, `width` and `height`:

answer:

```
interface Widget {
   public void repaint(Gctx g);
   public void handle(Gctx g, Event e);
   public int width(Gctx g);
   public int height(Gctx g);
}
```

b. (12 points) The OCaml code below (taken verbatim from lecture and the project) creates a `label` widget.

```
let label (s: string) : t * label_controller =
 let r : string ref = ref s in
   {
    repaint = (fun (g: Gctx.t) -> Gctx.draw_string g (0,0) !r);
    handle = (fun _ _ -> ());
    size = (fun (g: Gctx.t) -> Gctx.text_size g !r)
   },
   { set_label = fun (s: string) -> r := s }
```

Translate the above OCaml code into Java by completing the single **`class`** called `Label` that provides the functionality required by *both* the `Widget` and `LabelController` interfaces. Be sure to implement a constructor and all the required methods, which themselves might need to invoke `Gctx` methods. Properly encapsulate the `Label`'s local state.

answer:

```
public class Label implements Widget, LabelController {
   private String r;

   public Label(String s) {
      r = s;
   }

   public void repaint(Gctx g) {
    g.drawString(r,0,0);
   }

   public void handle(Gctx g, Event e) {
      return;
   }

   public int getWidth(Gctx g) {
      return g.textWidth(s);
   }

   public int getHeight(Gctx g) {
      return g.textHeight(s);
   }

   public void setLabel(String newString) {
      r = newString;
   }
}
```

4. Java Subtyping: Inheritance, Interfaces and Dynamic Classes (20 points)

Hint: Draw the subtype hierarchy. Consider these Java class and interface definitions:

```
interface I { public A method1(); }

interface J extends I { public B method2(); }

interface K { public B method3(B b); }

class A implements I, K {
   public A method1() { return new A(); }
   public B method3(B b) { return b; }
}

class B implements J {
   public A method1() { return new A(); }
   public B method2() { return new C(); }
   public B method3(B b) { return b; }
}

class C extends B implements K {
   public B method4(B b) { return new B(); }
}
```

For each code fragment below, fill in the blank with the name of the *dynamic class* of the value stored in the variable indicated on the line, or write "ill typed" if the code fragment contains a compile-time type error (i.e. Eclipse would put a red line under some part of the program).

```
(a)  K k1 = new C();                                       k1:_________C______________

(b)  I i1 = new C();                                       i1:_________C______________

(c)  I i2 = new A();                                       i2:_________A______________

(d)  K k3;
     if (true) {k3 = new A();} else {k3 = new B();}        k3:____ill typed_________

(e)  K k4;
     if (true) {k4 = new A();} else {k4 = new C();}        k4:_________A______________

(f)  J j1 = (new A()).method3(new C());                    j1:_________C______________

(g)  B b1 = new C();
     B b2 = b1.method4(new B());                           b2:___ill typed__________

(h)  J j2 = new B();
     J j3 = j2.method2();                                  j3:_________C______________

(i)  C c1 = new C();
     Object o1 = c1.method4(new C());                      o1:_________B______________

(j)  C c2 = new Object();                                  c2:___ill typed__________
```

5. Array Programming (20 points)

Implement the Java method `growByTwo` that takes in a 2D array of **`int`**s and returns a new array in which each single **`int`** value is replaced with a 2×2 square of copies of that value.

For example, if the input array is:

```
int[][] arr = {{ 0, 1, 2 },
               { 3, 4, 5 }};
```

The output of `growByTwo(arr)` should be:

```
{{ 0, 0, 1, 1, 2, 2 },
 { 0, 0, 1, 1, 2, 2 },
 { 3, 3, 4, 4, 5, 5 },
 { 3, 3, 4, 4, 5, 5 }}
```

You may assume that the input array is rectangular (i.e. every row is the same length) and that both dimensions are of size > 0.

answer:

```
public int[][] growByTwo(int[][] arr) {
   int height = arr.length * 2;
   int width = arr[0].length * 2;
   int[][] newArr = new int[height][width];

   for(int i=0; i<arr.length; i++ ){
      for(int j=0; j<arr[i].length; j++) {
         newArr[i*2][j*2] = arr[i][j];
         newArr[i*2+1][j*2] = arr[i][j];
         newArr[i*2][j*2+1] = arr[i][j];
         newArr[i*2+1][j*2+1] = arr[i][j];
      }
   }
   return newArr;
}
```

OR

```
public int[][] growByTwo2(int[][] arr) {
   int height = arr.length * 2;
   int width = arr[0].length * 2;
   int[][] newArr = new int[height][width];

   for(int i=0; i<newArr.length; i = i + 2 ){
      for(int j=0; j<newArr[i].length; j = j + 2) {
         newArr[i][j] = arr[i/2][j/2];
         newArr[i+1][j] = arr[i/2][j/2];
         newArr[i][j+1] = arr[i/2][j/2];
         newArr[i+1][j+1] = arr[i/2][j/2];
      }
   }
```

```
        return newArr;
    }
```

OR

```
    public int[][] growByTwo2(int[][] arr) {
        int height = arr.length * 2;
        int width = arr[0].length * 2;
        int[][] newArr = new int[height][width];

        for(int i=0; i<newArr.length; i++ ){
            for(int j=0; j<newArr[i].length; j++) {
                newArr[i][j] = arr[i/2][j/2];
            }
        }
        return newArr;
    }
```

Appendix

This appendix shows an example of the Stack and Heap components of the OCaml Abstract Stack Machine. Your diagram for Problem 1 should use similar "graphical notation" for `Some _` and `None`.

```
(* The types of mutable queues. *)
type 'a queuenode = { v : 'a;
                      mutable next : 'a queuenode option}

type 'a queue = { mutable head : 'a queuenode option;
                  mutable tail : 'a queuenode option }

let qn1 : int queuenode = {v = 1; next = None;}
let qn2 : int queuenode = {v = 2; next = Some qn1;}
let q : int queue = {head = Some qn2; tail = Some qn1;}
(* HERE *)
```

The OCaml program above yields the ASM Stack and Heap depicted below when the program execution reaches the point marked *(* HERE *)*.