Name: ______________________________

Pennkey: ______________________________

CIS 120 Final Exam
December 21, 2011

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

Signature: ______________________________ Date: ______________________

1	/20
2	/14
3	/18
4	/38
5	/30
Total	/120

- Do not begin the exam until you are told to do so.
- You have 120 minutes to complete the exam.
- There are 120 total points.
- There are 17 pages in this exam.
- Make sure your name and Pennkey (a.k.a. username) are on the top of this page.

1. True or False (20 points)

Indicate whether each of these statements is true or false. Some of the questions refer to the Java IO library's `FileReader` class—some of its documentation is reproduced in the appendix.

a. T F In Java, a **`static`** method gets passed an implicit **`this`** parameter.

b. T F According to the `FileReader` documentation, the following variable declaration is well typed:

```
Readable r = new FileReader("file.txt");
```

c. T F According to the following signature, this `FileReader` constructor may never raise a `NullPointerException`.

```
public FileReader(String filename) throws FileNotFoundException
```

d. T F In Java Swing programming, when the program's visual state changes it must call the `repaint()` method to update the display.

(This question was dropped from the exam for being ambiguous, all students received credit for any answer.)

e. T F In both the Java Swing libraries and our OCaml GUI library, the job of a `listener` is help route an event to the appropriate component (or widget) for subsequent handling.

f. T F In Java Swing programming, using an anonymous inner class is a good way to implement an `ActionListener` because the `actionPerformed` method often needs access to the outer class's local state.

g. T F It is not possible to write your own `Iterator` class in Java.

h. T F The Java collection type `Map<Section,List<Student>>` provides a simple means of determining which lab `Section` a given CIS 120 `Student` is in.

i. T F In Java, if `B` is a subtype of `A`, then `List<B>` is a subtype of `List<A>`.

j. T F For a type like `Set<E>` (in Java) or `'a set` (in OCaml) to be truly *abstract*, any client code that uses this type must *not* be able to determine its concrete representation (via lists, trees, arrays, etc.).

2. **Inheritance and Overriding (14 points)**

   Consider the following Java class declarations:

```
class A {
   public void print1() {
      System.out.println("A’s print1");
   }

   public void callPrint() {
      print1();
   }
}

class B extends A {
   public void print2() {
      System.out.println("B’s print2");
   }

   @Override
   public void callPrint() {
      print2();
   }
}

class C extends A {
   @Override
   public void print1() {
      System.out.println("C’s print1");
   }
}
```

   Answer the multiple choice questions on the following page.

For each code snippet below, indicate what string will get printed to the console, or mark "Ill typed" if the snippet has a type error.

a.
```
B x = new B();
x.print1();
```

`A's print1`   `B's print2`   `C's print1`   Ill typed

b.
```
A y = new B();
y.print1();
```

`A's print1`   `B's print2`   `C's print1`   Ill typed

c.
```
A q = new B();
q.print2();
```

`A's print1`   `B's print2`   `C's print1`   Ill typed

d.
```
C z = new C();
z.print1();
```

`A's print1`   `B's print2`   `C's print1`   Ill typed

e.
```
A v = new B();
v.callPrint();
```

`A's print1`   `B's print2`   `C's print1`   Ill typed

f.
```
C w = new C();
w.callPrint();
```

`A's print1`   `B's print2`   `C's print1`   Ill typed

g.
```
A u = new C();
u.callPrint();
```

`A's print1`   `B's print2`   `C's print1`   Ill typed

3. OCaml: Trees and First-class Functions (18 points)

Recall our OCaml definition of binary trees that store data in their internal nodes.

```
type 'a tree =
| Empty
| Node of 'a tree * 'a * 'a tree
```

An example of such a tree is given by:

```
let t : int tree =
  Node(Node(Node(Empty, 1, Empty), 2, Node(Empty, 0, Empty)),
       3,
       Node(Empty, 4, Empty))
```

We draw it like this (eliding the `Empty` values):

```
     3
    / \
   2   4
  / \
 1   0
```

Consider the following OCaml function:

```
let rec tree_fold (combine : 'b -> 'a -> 'b -> 'b)
                  (base : 'b)
                  (t : 'a tree) : 'b =
  begin match t with
    | Empty -> base
    | Node(lt, x, rt) ->
       combine (tree_fold combine base lt) x (tree_fold combine base rt)
  end
```

For each OCaml program below, circle the value computed for `ans`.

a.
```
let c = fun (ls:int) (x:int) (rs:int) -> ls + x + rs
let b = 0
let ans = tree_fold c b t

5            10             11             16
```

b.
```
let c = fun (ls:int) (x:int) (rs:int) -> ls + x + rs
let b = 1
let ans = tree_fold c b t

5            10             11             16
```

The tree `t` is reproduced here for your reference:

```
t =     3
       / \
      2   4
     / \
    1   0
```

---

c.

```
let c = fun (ls:int list) (x:int) (rs:int list) -> [x] @ ls @ rs
let b = []
let ans = tree_fold c b t

[3; 1; 0; 2; 4]     [3; 2; 4; 1; 0]     [3; 2; 1; 0; 4]     [3; 4; 2; 0; 1]
```

d.

```
let c = fun (ls:int tree) (x:int) (rs:int tree) -> Node(ls, x + x, rs)
let b = Empty
let ans = tree_fold c b t


    6           6           6          6
   / \         / \         / \        / \
  4   6       4   6       4   8      8   4
 / \         / \         / \            / \
3   2       3   2       2   0          0   2
```

e.

```
let rec l (t:int tree) : int =
  begin match t with
    | Empty -> 0
    | Node(Empty, x, Empty) -> x
    | Node(lt, _, rt) -> (l lt) + (l rt)
  end
let ans = l t

5             10            11            16
```

f.

```
let rec r (t:'a tree) : 'a tree =
  begin match t with
    | Empty -> Empty
    | Node(lt, x, rt) -> Node(r rt, x, r lt)
  end

let ans = r t


    3           3         3          3
   / \         / \       / \        / \
  2   4       4   2     4   2      4   2
 / \         / \           / \        / \
1   0       0   1         1   0      0   1
```

4. Program Design and Encapsulation (38 points)

In this problem you will use the test-driven design methodology to implement a Java class that represents a (simplified version of) finite multisets. For the purposes of this problem, a *finite multiset* is a collection of natural numbers (i.e. non-negative integer values). Order does not matter, but there may be more than one occurrence of each element. Each multiset has a *size*, which is the total number of occurrences of all elements. Here are some examples, written using a mathematical notation:

$\{\}$	the empty multiset, its size is 0
$\{0\}$	the multiset containing just one occurrence of 0, its size is 1
$\{1, 2, 1\}$	a multiset with two occurrences of 1 and one occurrence of 2, its size is 3

Such a structure can be defined in Java using an interface similar to Java's `Set` interface:

```
interface FiniteMultiSet {
   void add(int x);       // add one occurrence of x to the multiset
   void remove(int x);    // remove one occurrence of x, if present, from the multiset
   int size();            // return the number of elements in the multiset
   int numberOf(int x);  // return the number of occurrences of x
}
```

The following class implements the `FiniteMultiSet` interface for *bounded sets*, whose elements are in the range `0` to `max-1`.

```
class FMSet implements FiniteMultiSet {
   private int[] elts;
   private int size;

   public FMSet(int max) {
      this.elts = new int[max];
      this.size = 0;
   }

   public int size() {
      return size;
   }

   public void add(int x) {
      if (x < 0 || x >= elts.length) {
            throw new IllegalArgumentException("exceeded capacity");
      }
      elts[x] = elts[x] + 1;
      size = size + 1;
   }

   public int numberOf(int x) {
      if (x < 0 || x >= elts.length) {
            throw new IllegalArgumentException("exceeded capacity");
      }
      return elts[x];
   }
   public void remove(int x) { ... }
}
```

a. (5 points) Fill in the blanks of the following unit tests so that, according to the implementation of `FMSet` above, all of the tests succeed.

```
public class FMSetTest {
   @Test
   public void testEmpty() {
      FMSet s = new FMSet(0);

      assertEquals(________, s.size());
   }

   @Test
   public void testEmpty2() {
      FMSet s = new FMSet(3);

      assertEquals(________, s.size());
   }

   @Test
   public void testAdd0() {
      FMSet s = new FMSet(3);
      s.add(0);

      assertEquals(________, s.numberOf(0));

      assertEquals(________, s.size());
   }

   @Test
   public void testAdd00() {
      FMSet s = new FMSet(3);
      s.add(0);
      s.add(0);

      assertEquals(________, s.numberOf(0));

      assertEquals(________, s.size());
   }

   @Test
   public void testAdd01() {
      FMSet s = new FMSet(3);
      s.add(0);
      s.add(1);

      assertEquals(________, s.numberOf(0));

      assertEquals(________, s.numberOf(1));

      assertEquals(________, s.size());
   }
}
```

b. (9 points) Complete three *distinct* test cases for the `remove` method (whose implementation was omitted). Each test case should test a different aspect of `remove`'s functionality. You may insert additional uses of `assertEquals`.

```
@Test
public void testRemoveA() {
   FMSet s = new FMSet(3);









   assertEquals(________, s.size());
}

@Test
public void testRemoveB() {
   FMSet s = new FMSet(3);









   assertEquals(________, s.size());
}

@Test
public void testRemoveC() {
   FMSet s = new FMSet(3);









   assertEquals(________, s.size());
}
```

c. (6 points) The implementor of the `FMSet` class intends the code to enforce the following invariants:

(1) `elts != null`

(2) `elts[i] >= 0` for all `i` in `0...(elts.length-1)`

(3) `size = elts[0] + elts[1] + ... + elts[elts.length-1]`

Complete the implementation of the `remove` method. It should never throw an exception, and it should preserve the `FMSet` invariants. NOTE: you *do not* need to use **`try...catch`**.

```
public void remove(int x) {









}
```

d. (10 points) The `FMSet` implementor wanted to make a more convenient way of constructing a multiset out of an integer array, so he added the following constructor. It is intended to create an `FMSet` from an array in such a way that if the array is does not satisfy the invariants, an `IllegalArgumentException` is thrown.

```
public FMSet(int[] arr) {
   // maintain invariant (1)
   if (arr == null) {
      throw new IllegalArgumentException();
   }
   // initialize the array
   elts = arr;

   // calculate the size
   size = 0;
   for(int i=0; i<arr.length; i++) {
      size = size + elts[i];
   }
}
```

(Question continues on the next page.)

Although the constructor above is guaranteed to preserve invariant (1), invariants (2) and (3) can both be broken by code that uses this constructor. Demonstrate that this is the case by completing the code for the following two tests.

```
@Test
public void testInv2() {
    // complete this test that breaks invariant (2)


    int[] a = __________________ ;
    // use try ... catch to create a test that succeeds if
    // new FMSet(a) appropriately raises IllegalArgumentException
    // the method fail () can be used to cause the test to fail











}

@Test
public void testInv3() {
    // complete this test that shows how to break invariant (3)


    int[] a = __________________ ;



    FMSet s = new FMSet(a);







    assertEquals(1, s.numberOf(0));
    assertEquals(1, s.size());
}
```

e. (8 points) Fix the implementation of the broken `FMSet` constructor so that all interactions with the `FMSet` are guaranteed to preserve invariants (2) and (3). Your implementation should raise an `IllegalArgumentException` if `a` does not represent a valid multiset.

```
public FMSet(int[] arr) {
    // maintain invariant (1)
    if (arr == null) {
       throw new IllegalArgumentException();
    }
    // initialize the array

    elts =











 }
```

Hint: If your solution does not easily fit into the space above, it is too complicated.

5. Java ASM and Linked Datastructures (30 points)

This problem concerns the following Java implementation of a singly-linked `Queue` datatype.

```java
import java.util.NoSuchElementException;

class QueueNode<E> {
   public E v;
   public QueueNode<E> next;
   QueueNode(E elt, QueueNode<E> n) {
      v = elt;
      next = n;
   }
}

class Queue<E> {
   private QueueNode<E> head;
   private QueueNode<E> tail;

   public Queue() {
      head = null;
      tail = null;
   }

   public boolean isEmpty() {
      return head == null;
   }

   public void enq(E elt) {
      QueueNode<E> newNode = new QueueNode<E>(elt, null);
      if (tail == null) {
         head = newNode;
         tail = newNode;
      } else {
         tail.next = newNode;
         tail = newNode;
      }
   }

   public E deq() {
      if (head == null) {
         throw new NoSuchElementException();
      }
      E x = head.v;
      head = head.next;
      if (head == null) {
         tail = null;
      }
      return x;
   }
}
```

a. (15 points) Draw the state of the Java Abstract Stack Machine that will be reached at the point marked "HERE" after running the code below. Do *not* show the class table, but do show the stack and heap. Be sure to label the *dynamic class* of each object in the heap. The appendix contains an example Java ASM configuration for your reference.

```
Queue<String> q1 = new Queue<String>();
Queue<Queue<String>> q2 = new Queue<Queue<String>>();
q2.enq(q1);
q2.enq(q1);
// HERE
```

Stack   Heap

b. (15 points) Complete the method `twin`, which is added to the `Queue` class and creates a twin for each element in the queue. That is, if the `Queue q` contains the elements `1, 2, 3` in order from head to tail, then `q.twin()` will update `q` to contain the elements `1, 1, 2, 2, 3, 3`. Your method should re-use the `QueueNode` objects from the original queue; note that this means you cannot use the `deq` and `enq` operations—manipulate the pointers directly. Be sure to preserve the queue invariants.

```
public void twin() {











}
```

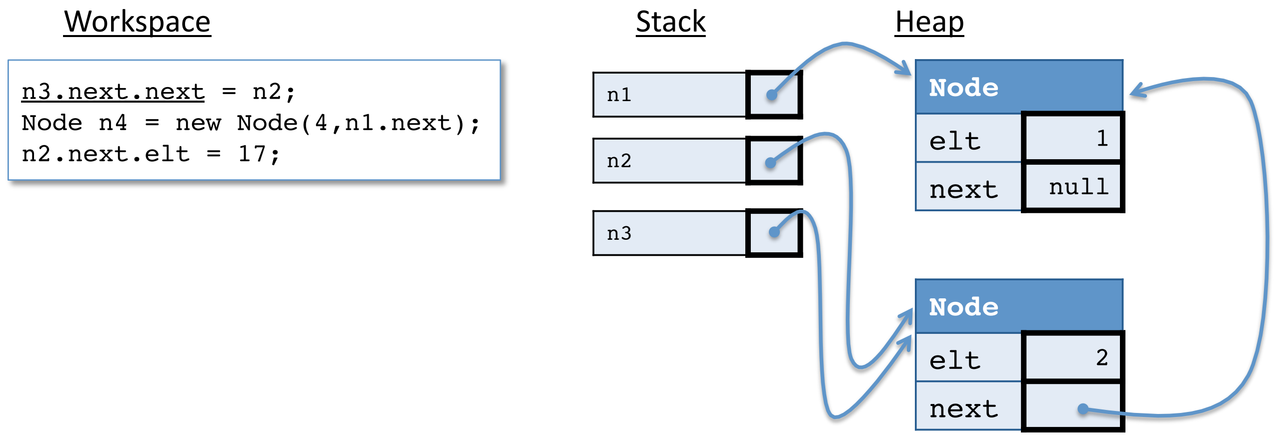
Hint: If your solution does not easily fit into the space above, it is too complicated.

Appendix

Example Java Abstract Stack Machine configuration, *excluding the class table*, for the following program when run to the point marked “HERE”:

```
public class Node {
  public int elt;
  public Node next;
  public Node(int e0, Node n0) {
    elt = e0;
    next = n0;
  }
}

public static void main(String[] args) {
  Node n1 = new Node(1, null);
  Node n2 = new Node(2, n1);
  Node n3 = n2;
  // HERE
  n3.next.next = n2;
  Node n4 = new Node(4, n1.next);
  n2.next.elt = 17;
}
```



java.io

Class FileReader

```
java.lang.Object
  └java.io.Reader
      └java.io.InputStreamReader
          └java.io.FileReader
```

All Implemented Interfaces:

Closeable, Readable

```
public class FileReader
extends InputStreamReader
```

Convenience class for reading character files. The constructors of this class assume that the default character encoding and the default byte-buffer size are appropriate. To specify these values yourself, construct an InputStreamReader on a FileInputStream.

`FileReader` is meant for reading streams of characters. For reading streams of raw bytes, consider using a `FileInputStream`.

FileReader

```
public FileReader(String fileName)
           throws FileNotFoundException
```

Creates a new `FileReader`, given the name of the file to read from.

Parameters:

`fileName` - the name of the file to read from

Throws:

`FileNotFoundException` - if the named file does not exist, is a directory rather than a regular file, or for some other reason cannot be opened for reading.