CSC 148H Midterm

Fall 2005

St. George Campus

Duration — 50 minutes

Student Number: |_|_|_|_|_|_|_|_|_|

Family Name: ________________________________

Given Name: ________________________________

No Aids Allowed.
*Do **not** turn this page until you have received the signal to start.*
Read this entire page or you'll miss the bonus question.

Bonus question: if you legibly write your student ID at the top of each odd numbered page you will get an extra mark.

# 1:	_____/10
# 2:	_____/10
# 3:	_____/10
BONUS:	_____/ 1
TOTAL:	_____/30

Good Luck! PLEASE HAND IN

Question 1. [10 MARKS]

```
public class ArrayQueue
        implements Queue {

  public int size;
  public Object[] contents;

  public ArrayQueue(int n) {
    contents = new Object[n];
  }
  /** Precondition:
   *    The queue cannot be full. */
  public void enqueue(Object o) {
    contents[size++] = o;
  }
  /** Precondition:
   *    The queue cannot be empty. */
  public Object dequeue() {
    Object head = contents[0];
    for (int i = 0; i < size - 1; ++i) {
      contents[i] = contents[i + 1];
    }
    --size;
    return head;
  }
  /** Precondition:
   *    The queue cannot be empty. */
  public Object head() {
    return contents[0];
  }
  public int size() {
    return size;
  }
  public int capacity() {
    return contents.length;
  }
} // End of ArrayQueue.
```

The `ArrayQueue` class definition to the left is same as the one we discussed in class, except the instance variables `size` and `contents` are `public`. (While it is is not a good idea in general to set these instance variables to have `public` accessibility, it is useful for the purposes of this exam.) Assume the interface Queue includes the methods headers (other than the constructor) of all the methods in `ArrayQueue`. Note that the first element in the queue (if any) is always stored at index 0 of the `contents` array, and the last element (if any) is at index `size-1`.

Write the Java code for a subclass of `ArrayQueue`, called `LeakyQ`, which has the following properties. It should have a **constructor which takes a single integer parameter** which specifies the (constant) capacity of the queue to be constructed. It should also have a **remove method which takes a single integer parameter** providing the index of a queue element to be removed. The `remove` method should return the Object that is removed from the queue. Any items which come after the removed item in the queue should be moved one space towards the front of the queue. If the parameter for `remove` is not between `0` and `size-1`, then the method should throw a `java.lang.IndexOutOfBoundsException`, which is a `RuntimeException`.

Your `LeakyQ` should not declare any instance variables. Moreover, it should only contain one constructor and one method, as described above.

An example of the use of this new class is as follows:

```
LeakyQ q = new LeakyQ(10);
q.enqueue("A");
q.enqueue("B");
q.enqueue("C");
q.remove(1);  // returns "B"
q.size();     // returns 2
q.head();     // returns "A"
q.remove(1);  // returns "C"
q.size();     // returns 1
q.head();     // returns "A"
```

Use the back of the last page for scratch work, and write your solution on the next page.

Question 1. (CONTINUED)

SOLUTION

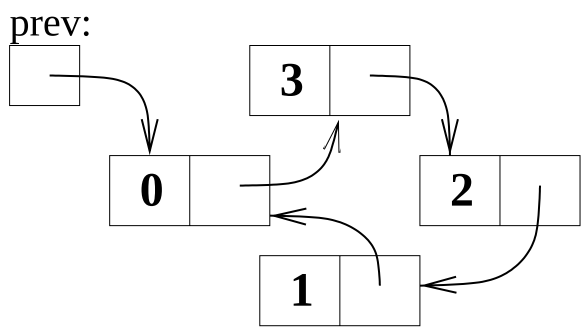
```
public class LeakyQ extends ArrayQueue {

  public LeakyQ(int cap) {
    super(cap);
  }

  public Object remove(int k) {
    if (k < 0 || k > size() -1)
      throw new IndexOutOfBoundsException("Index "+k+
                                           " out of bounds.");
    Object o = contents[k];
    --size;
    for (int i = k; i < size; i++)
      contents[i] = contents[i+1];
    return o;
  }
}
```

MARKING

```
+1 public class LeakyQ extends ArrayQueue.
+1 Constructor, with integer argument.  Calls super.
+1 public Object remove(int k)  (any parameter name)
+1 check index in range.
+2 throw new exception (does not need to have a message).
+1 size decremented.
+2 contents shifted by one, starting at removed item.
+1 returns the object that was removed.
```

Question 2. [10 MARKS]

The LinkedRing class provides a linked list of items where the last element in the list refers back to the first element. A sketch of a possible configuration for the LinkedRing is given below:



Write the insert method in the LinkedRing class according to the method comment below. For example, the state pictured to the left would arise from the following code:

```
LinkedRing lr = new LinkedRing();
lr.insert(new Integer(0));
lr.insert(new Integer(1));
lr.insert(new Integer(2));
lr.insert(new Integer(3));
```

Each item in the list is stored in a ListNode, which is defined as follows:

```
public class ListNode {
  public Object value;
  public ListNode link;
  public ListNode(Object o) {
    value = o;
  }
}
```

Finally, here is the beginning of the LinkedRing class definition:

```
public class LinkedRing {
  /** If prev == null then the list is empty,
   *  otherwise, prev.link refers to the first item
   *  on the list. */
  private ListNode prev;
  /** The number of items in the list. */
  private int size;

  public LinkedRing() {}

  /** Insert object o as the first element in the list, pushing all the other
   *  items in the list one step further away from being the first item.
   *  Postcondition: The link for last item in the list will refer to this
   *  newly inserted item. */
  public void insert(Object o) {
     // Complete this method (only).  There is more space on the next page.
```

Question 2. (continued)

SOLUTION

```
  public void insert(Object o) {

    ListNode tmp = new ListNode(o);

    if (size == 0) {
      // Create a list of length 1.
      prev = tmp;
      prev.link = prev;
      size = 1;
      return;
    }

    tmp.link = prev.link;
    prev.link = tmp;
    ++size;
  }
```

MARKING

```
+2 Create a new ListNode containing the Object o.
+4 Insertion when the list is empty.
    1 Check for empty list.
    1 Set prev.
    1 Set the new node's link to itself.
    1 Set size to be 1 (or increment it by one).
+4 Insertion when the list is not empty.
    1 Check list is not empty.
    1 Link the new node to the old first node.
    1 Link the last node (referred to by prev) to
      the new node.
      Make sure these updates are done in the correct order (or
      a temporary variable is used... no deduction for using an
      extra variable).
    1 Increment size.

DEDUCTIONS:
-2 If the method can ever throw a NullPointerException.
-1 If the insertion is done in the wrong place.
-2 If part of the list can be lost.
```

Question 3. [10 MARKS]

Draw the memory model for the situation where the 5th line of the main method is about to be executed. You do not need to draw `String` or `String[]` objects. There is more space on the last page.

```
   public class Driver {
     public static void main(String[] args) {
 1:    Point a = new Point(0, 0);
 2:    Origin b = new Origin(a);
 3:    Point c = b.get();
 4:    a.x = 100;
 5:    a.y = 50;
     }
   }
```

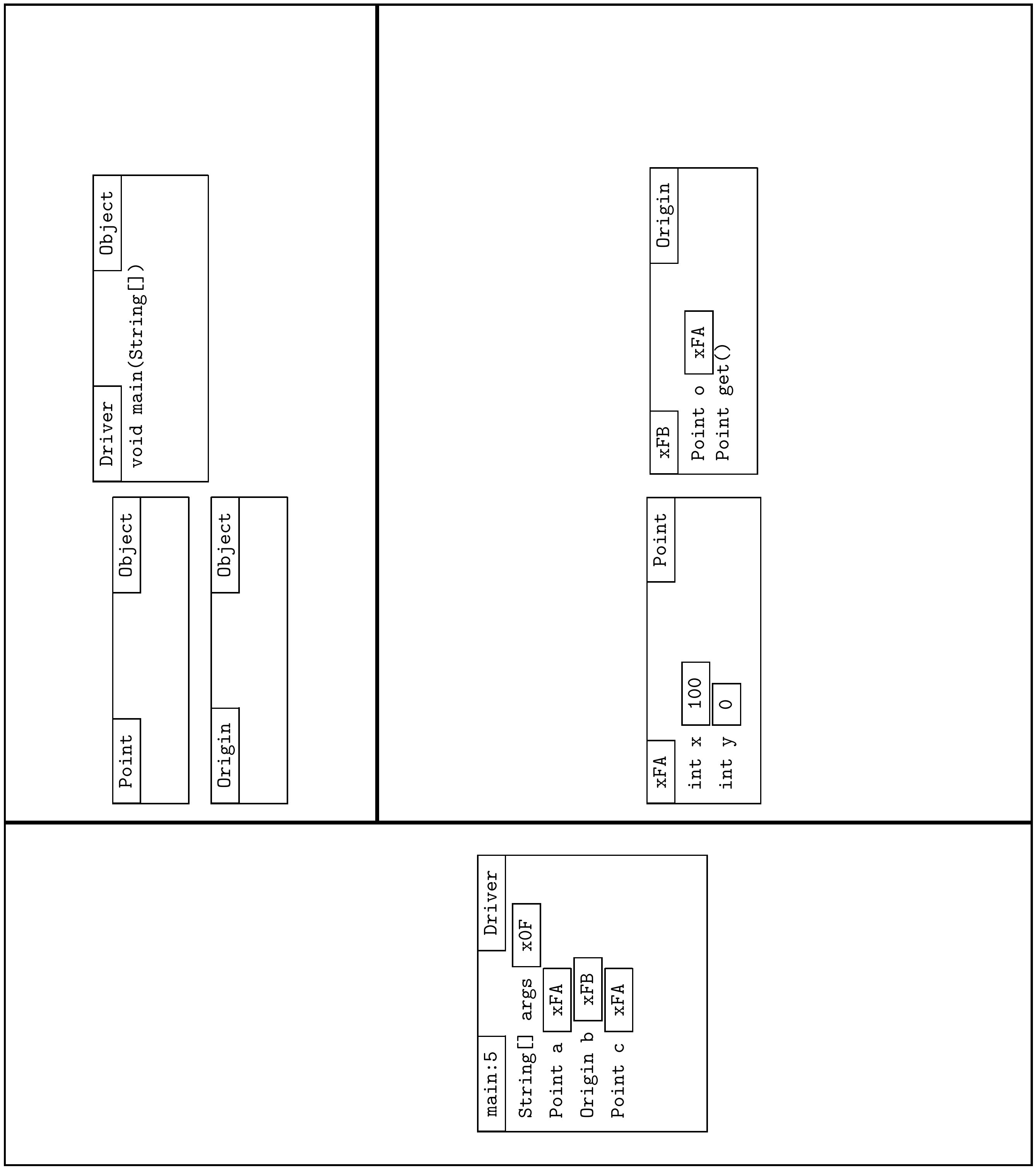
```
public class Origin {
  private Point o;
  public Origin(Point p) { o = p; }
  public Point get() { return o; }
}
```

```
public class Point {
  public int x;
  public int y;
  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
}
```

SOLUTION: See next page.
MARKING:

```
+4 Runtime Stack
    1 Box with label (main:5) and scope (Driver) sub-boxes
    1 Point a  address1
    1 Origin b address2
    1 Point c  SAME ADDRESS as address1
    Can leave types off of a,b,c.
    Can have a String[] args entry, but can also omit it.
+2 Static space.
    1 Must have box for Driver, correct label (Driver) and
      scope (Object).
    1 main method in Driver box.
    Can include boxes for other classes (but not necessary).
+4 Object space (Heap)
    2 For one Point box, and one Origin box.  These
      boxes should have the correct labels (addresses) and scopes
      (i.e. Point and Origin, respectively).
    1 Contents of Point box. x MUST be 100, and y should be 0
      (-1/2 if y is 50).
      Can leave types off x and y.
    1 Contents or Origin box.  o MUST have address of a Point
      box in it.   The get() method should be here (1/2 a point),
      but can leave off the return type of the get() method.
    Can also have String and/or String[] objects (but not necessary).

DEDUCTIONS:
-2 If there is more than 1 Point box in the Object Space.
-2 Extra boxes on runtime stack, static space, or heap (other
   than more Point boxes, which is addressed above, or String
   and String[] boxes, which are optional).
```

Question 3. (continued)


Point
Object
Origin
Object
Driver
Object
void main(String[])
xFA
Point
int x
100
int y
0
xFB
Origin
Point o
xFA
Point get()
main:5
Driver
String[] args
x0F
Point a
xFA
Origin b
xFB
Point c
xFA