CS 61BL (Clancy/Lewis) **Exam 1**
Spring 2011

Read and fill in this page now.
Do NOT turn the page until you are told to do so.

Your name:_____ A random five-digit number: _____

Circle the last two letters of your login (cs61bl-xx)

1st letter: a b c d e f g h i j k l m n o p q r s t u v w x y z

2nd letter: a b c d e f g h i j k l m n o p q r s t u v w x y z

Circle your section:

Tu/Th 8-11	Tu/Th 11-2	Tu/Th 2-5	Tu/Th 2-5	Tu/Th 5-8	W/F 8-11	W/F 11-2	W/F 2-5
Matt	Mike	Courtney	Tarush	Courtney	Stephanie	Kaushik	Tarush

Name of the closest person on your right (possibly "aisle" or "wall") _____

Name of the closest person on your left (possibly "aisle" or "wall") _____

Problem 0	_____			Total:			/36
Problem 1	_____	_____	_____	Problem 4	_____	_____	
Problem 2	_____	_____		Problem 5	_____		
Problem 3	_____			Problem 6	_____		

This is an open-book test. You have approximately an hour and 50 minutes to complete it. You may consult any books, notes, or other paper-based inanimate objects available to you. You may not use any electronic devices.

To avoid confusion, read the problems carefully. If you find it hard to understand a problem, ask us to explain it. If you have a question during the test, please come to the front or the side of the room to ask it.

This exam comprises 12% of the points on which your final grade will be based. Partial credit may be given for wrong answers. Your exam should contain seven problems (numbered 0 through 6) on twelve pages. (The last two pages of the exam contain some background information.) Please write your answers in the spaces provided in the test; in particular, we will not grade anything on the back of an exam page unless we are clearly told on the front of the page to look there.

Some students are taking this exam late. Please do not talk to them, mail them information, or post anything about the exam to news groups or discussion forums until after Friday.

Relax—this exam is not worth having heart failure about.

Your five-digit number: _____

Problem 0 (2 points)

Put your secret number on each page. Also make sure you have provided the information requested on the first page.

Problem 1 (5 points)

In earlier lab activities, we encountered the need to override the equals method in the Object class. The simplified Point class at the end of this exam contains an example:

```
public boolean equals (Object obj) {
   Point p = (Point) obj;
   if (obj == null) {
      return false;
   } else {
      return myX == p.myX && myY == p.myY;
   }
}
```

Part a

The code above tests only that one of the references being compared is null. Why do we not have to check that *both* are null?

Part b

Rewrite equals as a static method whose arguments are Point references.

Part c

Suppose you don't know whether or not Object.equals has been overridden in the Point class and you don't have access to the source code. Write a main method to determine whether or not Object.equals in the Point class has been overridden as described above. Your experiment should result in one of the following being printed:

```
equals has been overridden
equals has not been overridden
```

```
public static void main (String [ ] args) {
    // Your code goes here.
```

Your five-digit number: _____

Problem 2 (5 points)

Part a

Provide the code for a subclass of Point named URQuadPoint ("upper right quadrant point") that represents only points whose coordinates are both nonnegative. (Code for the Point class, a simplified version of the real thing, appears at the end of this exam.) An attempt to construct a point with a negative x or y value should produce an IllegalArgumentException.

Your code for URQuadPoint should take advantage of the Point methods and variables as much as possible.

```
public class URQuadPoint extends Point {
```

Part b

For each of the assignments below, indicate whether it is legal, causes a compile error, or compiles without errors but throws an exception when run. If the latter, describe what kind of exception.

assignment	*compiles and runs without error?*	*compile error?*	*runtime exception? if so, what kind?*
`Point p1 = new URQuadPoint (-7, 3);`			
`URQuadPoint p3 = new Point (7, 3);`			

Problem 3 (5 points)

Consider the output below, produced by the main method in the Point class described at the end of this exam.

```
First step
(5,6)
(5,6)
(14,15)
Second step
(7,8)
(7,8)
(15,16)
```

Here is most of the code that produced it.

```
public static void main (String [ ] args) {
   // declaration and initialization of myPoints goes here

       ;

   myPoints [0] = ____________________________________________  ;

   myPoints [1] = ____________________________________________  ;

   myPoints [2] = ____________________________________________  ;

   System.out.println ("First step");

   for (int k=0; k<3; k++) {
      myPoints[k].myX++;
      myPoints[k].myY++;
   }

   for (int k=0; k<3; k++) {
      System.out.println (myPoints[k]);
   }

   System.out.println ("Second step");

   for (int k=0; k<3; k++) {
      myPoints[k].myX++;
      myPoints[k].myY++;
   }

   for (int k=0; k<3; k++) {
      System.out.println (myPoints[k]);
   }
}
```

Fill in the blanks so that the resulting program produces the output given above.

Your five-digit number: _____

Problem 4 (7 points)

Part a

Provide a method named removeAdjacentDuplicates for the IntSequence class that removes all sequence elements that duplicate their immediate predecessors in the sequence. Here's an example.

sequence prior to calling removeAdjacentDuplicates	*sequence after the call*
`2 3 55 3 3 16 16 16`	`2 3 55 3 16` (with `myCount == 5`)

```
public IntSequence removeAdjacentDuplicates ( ) {
```

Your five-digit number: _____

Part b

In the table below, list up to eight calls to the check method defined below that provide more information about the correctness of removeAdjacentDuplicates than does the given example (2, 3, 55, 3, 3, 16, 16, 16). Accompany each call with the extra information it would provide.

```
private void check (int [ ] testArray, int [ ] resultArray) {
   IntSequence testSeq = new IntSequence ( );
   for (int k=0; k<testArray.length; k++) {
      testSeq.add (testArray[k]);
   }
   IntSequence resultSeq = new IntSequence ( );
   for (int k=0; k<resultArray.length; k++) {
      resultSeq.add (resultArray[k]);
   }
   // Use IntSequence equals method.
   assertEquals (testSeq.removeAdjacentDuplicates ( ), resultSeq);
}
```

Contents of testArray argument to `check`	Contents of resultArray argument to `check`	Information provided by the test
`{2, 3, 55, 3, 3, 16, 16, 16}`	`{2, 3, 55, 3, 16}`	

Problem 5 (6 points)

Provide methods named initIterator, hasNext, and next that implement an iterator for the IntSequence class. (Headers for this class are listed at the end of this exam.) The iterator should enumerate all the elements of the sequence except those that are equal to their immediate predecessors in the sequence. More formally, if there is a k for which myValues[k] == myValues[k+1] = … = myValues[m], then elements k+1 through m should not appear in the enumeration. Here's an example.

contents of myValues	*sequence returned by successive calls to next*
2 3 55 3 3 16 16 16	2 3 55 3 16

Your initIterator method must run in constant time for full credit. Write your code below, and include the invariant your implementation maintains.

```
public class IntSequence {
      ... // Other IntSequence methods appear here; see the page at the end of the exam.
   // Declaration of extra instance variables go here.




   // Invariant maintained by hasNext and next goes here.
   //
   //
   //




   public void initIterator ( ) [




   }
   public boolean hasNext ( ) {




   }
```

```
    public int next ( ) {

    }
}
```

Your five-digit number: _____

Problem 6 (6 points)

Given below is a partial implementation of the adding machine program on an earlier homework. (Recall that the adding machine printed a subtotal when it encountered a 0, and a total when it encountered two consecutive zeroes.)

```
public static void main (String [ ] args) {
   int subtotal = 0;
   int total = 0;
   boolean justStarting = true;
   Scanner inputValues = new Scanner (System.in);
   while (true) {
      int k = inputValues.nextInt ( );
      if (k == 0 && !justStarting) {
         System.out.println ("total" + total);
         System.exit (0);
      }
         ...
```

Below, supply the rest of the code, using *only* the following statements plus any number of right braces. You may use a statement more than once. You need not use all the statements. Don't change the existing code.

`subtotal = 0;`	`justStarting = false;`
`subtotal = subtotal + k;`	`justStarting = true;`
`total = total + k;`	`k = inputValues.nextInt ( );`
`while (k != 0) {`	`while (k == 0) {`
`System.out.println ("subtotal " + subtotal);`	`System.out.println ("total " + total);`

Your five-digit number: _____

The Point class

```
public class Point {

   private int myX;
   private int myY;

   public Point (int x, int y) {
      myX = x;
      myY = y;
   }

   public int getX ( ) {
      return myX;
   }

   public int getY ( ) {
      return myY;
   }

   public String toString ( ) {
      return "(" + myX + "," + myY + ")";
   }

   public boolean equals (Object obj) {
      Point p = (Point) obj;
      if (obj == null) {
         return false;
      } else {
         return myX == p.myX && myY == p.myY;
      }
   }
}
```

Your five-digit number: _____

Headers for the IntSequence class

```
// Constructor; the argument is the actual size of the array,
// or equivalently, the (temporary) maximum number of elements it can hold.
public IntSequence (int capacity);

// Return true when this sequence is empty, and return false otherwise.
public boolean isEmpty ( )

// Return the number of values in this sequence.
public int size ( )

// Return the value at the given position in this sequence.
public int elementAt (int position);

// Include the argument among those stored in this sequence by placing it
// in the first unused spot in the array and incrementing the count.
// Assumption: this sequence isn't full.
public void add (int toBeAdded);

// Return the external representation of this sequence.
public String toString ( );

// Remove the sequence value at the given position.
// (Positions start at 0.)
// Assumption: the given position is less than the number of values
// in this sequence.
public void remove (int position);

// Insert the argument among those stored in this sequence by moving values
// at and after the given position to the right, then setting the value
// at the given position to the argument value.
// Assumptions: this sequence isn't full; the given position is less than
// the number of values in this sequence.
public void insert (int value, int position);

// Return true if the given value is among those in this sequence; return false otherwise.
public boolean contains (int value);

// Return true if this sequence and the argument represent the same sequence; return false otherwise.
public boolean equals (Object seq);
```