University of Illinois at Urbana-Champaign
Department of Computer Science

Second Examination Fall 2011

CS 125 Introduction to Computer Science
90 minutes permitted

First name: ___________________ Last name: ___________________

NetID: _ _ _ _ _ _ _ _ _ @ illinois.edu
(please write legibly)

Discussion Section: AY___
(We will return your exam manuscript to you in this section)

```
AYA Tues 09:00am-10:50
AYB Tues 11:00am-12:50
AYC Tues 01:00pm-02:50
AYD Tues 03:00pm-04:50
AYE Tues 05:00pm-06:50
AYF Tues 07:00pm-08:50
AYG Wed 09:00am-10:50
AYH Wed 11:00am-12:50
AYK Wed 01:00pm-02:50
AYI Wed 03:00pm-04:50
AYJ Wed 05:00pm-06:50
```

• This is a closed book and closed notes exam. No electronic aids are allowed. You must turn this exam booklet in before leaving.

• You have 9 sheets total including one scratch sheet. You may use both sides of the paper for your response. The last sheet is scratch paper; you may detach it while taking the exam, but must turn it in with the exam when you leave.

• Carefully read the specification given for each class. You may not write, or assume the existence of, any unspecified additional methods or constructors.

• Unless we say otherwise in the specific problem, you can assume all values entered by the user will be acceptable input for that program.

• When you write code, you may use a shorthand for System.out and TextIO input and output methods provided it is obvious to the graders which method you are using. For example it is acceptable to use Sopln in place of System.out.println and to use Sopt in place of System.out.print Likewise, you can use T.rlnI(), T.rlnC(), and T.rlnD() in place of TextIO.readlnInt(), TextIO.readlnChar(), and TextIO.readlnDouble().

• For full marks correct syntax is required: Ensure all statements include a semicolon and the correct use of upper/lower case, single quotes and double quotes. However a syntactically correct but flawed program will earn a low score i.e. semantics and correctness is more important than minor typos and syntax errors.

Problem	Points	Score	Grader
1	15		
2	16		
3	15		
4	21		
5	18		
6	15		
Total	100		

1. Short Answers – 15 points (3 points each)

(a) What is printed by the last two lines of the following Java code? Assume that the constructor initializes the private instance variables and that no compile or runtime errors occur. You do not need to write an explanation.

```
Account  a1 = new Account("James", 1234);
Account  a2 = new Account ("Jill", 4444);
Account  a3 = new Account ("John", 9870);
a1 = a2;
a2 = a3;
a3 = a1;
System.out.print(a2 == a3);
System.out.print(a1 == a2);
```

Write exactly the program output here and nothing else: ____________________

(b) The following code compiles without error. What can you conclude about the classes A and B?

```
A x = new B();
```

____

(c) "The class provides public methods and hides it's implementation" is best described as which one of the following?

A) Primitive types
B) Operator overloading
C) Encapsulation
D) Compile-time type checking
E) Polymorphism

Your answer: __________

(d) Which answer best describes immutable?

A) The class must define at least one constructor.
B) The class must not define a copy constructor.
C) The class must not have any class methods.
D) In Java, only primitive values can be immutable.
E) None of the above.

Your answer: __________

(e) Consider the following code example, then answer two questions: How many String objects are created and which line will cause an Exception to be thrown?

```
1    String[] data = new String[8];
2    for(int i=0;i<4;i++)
3            data[i] = data[7-i];
4    String result = data[1];
5    System.out.println( result.toUpperCase() );
```

Number of String objects created: ______

Exception thrown at line: ______

2. Lists, Stacks and Queues – 16 points

Complete the First-In-Last-Out *Stack* class by implementing `peekTop`, `getShortStack`, `anyEmpty` public instance methods.

```
public class Stack {
  private String[] arr = new String[0];

  public int getCount() { return arr.length;}

  public void push(String s) {
```
// Code not shown. Appends the string: Copies *arr* references to a new array. Appends string s to the end of this array and updates *arr* to point to the new array.
```
  }
  public String pop(){/* reduces the stack by one string (code not shown) */}
```

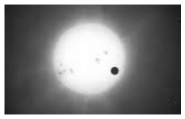
/* A. The public instance method `peekTop` takes no parameters and returns a String. Returns the String most recently added but does not modify the stack. Returns a `null` reference if the stack is empty. */

/* B. The public instance method *getShortStack* takes no parameters and returns a new Stack object. Returns a stack of the strings from the original stack that have a length less than 32. The ordering of strings is unspecified. The original stack is unchanged. Use String's *length()* method and Stack's *push* method to construct the result. */

/* C. The public instance method *anyEmpty* takes no parameters and returns a boolean. Returns true if any of the strings are zero length, false otherwise. Perform this search efficiently: Do not check any more strings as soon as an empty string is found. */

```
} // end class
```

3. I took CS125 & all I got was a new planet named after my dog – 15 Points



You are searching for exoplanets by looking for periodic variations in star brightness. You downloaded two *grayscale* pictures of exactly the same patch of stars taken at different times and want to calculate the change in brightness at every point.

A picture is represented as a 2D int array. Valid integer values are in the range 0(black) to 255 (white) i.e. only brightness is represented and there are no red/green/blue components. You can assume that both pictures are rectangular and are identical in size (width of A = width of B).

Complete the class method below to create a new picture of the same size. If either parameter is `null`, immediately return `null`. Each output pixel brightness value represents the **twice** (2x) the **absolute** difference of the corresponding pixel in image A and image B. If the output value is greater than 255, set the output value to 255. The source arrays are never modified.

```
public static int[][] diff(int[][] A, int[][] B) {
```

4. X marks the spot – 21 Points

The entire Geo class is shown below.

```
public class Geo {
  private double lat = 180* Math.random() - 90; // a random latitude
  private double longi = 360* Math.random();    // a random longitude
  public double getLat() {return lat;}
  public double getLong() {return longi;}
}
```

Write a new class named "Secret". Your class:

1. Extends the Geo class.
2. Has one additional private variable named 'showAfter' of type long that represents time in milliseconds since Midnight Jan 1 1970 UTC.
3. Has a constructor that takes no parameters. The constructor sets the *showAfter* variable to the current time (use the class method `currentTimeMillis`() in the `System` class) plus a random delay of up to 60000 milliseconds.
4. Has an instance method named 'toString' that takes no parameters and returns a String. If the current time is at least *showAfter*, then return the string "*latitude*,*longitude*" (where *latitude* and *longitude* are replaced by the instance values), otherwise return an empty string. Hint: Notice the Geo class does not define a *toString* method and the instance variables are private.

Complete the following program. Your program calls a class method, named "create", defined in a different class named "Factory" that takes no parameters and returns an array of 100 Secret objects. Waits for the user to press return then prints the location of the *Secret* objects that are now known. Use their *toString* method to print the location. Do not print any empty strings. Print one location per line. Continue overleaf if necessary.

```
public class Finder { // Continue overleaf if more space is required
   public static void main(String[] args) {
```

5. Objects I – 18 Points

0	1	0	1	0

Do not create any other additional constructors, instance or class methods or instance or class variables – only create the constructors, methods and instance variables specified below.

A sparse binary list is a very large list of data where most of the entries are zero and only a few entries are one. To efficiently store a massive list, the list internally holds the *positions* of the non-zero entries. Your programming buddy has already started the BigList class shown below:

```
public class BigList {
     private int[] pos; // should this be static?
     public void reset(int[] newPositions) {
        this.pos = newPositions;
     }
```

A. Create a public **instance** method named '**getV**' that takes an integer parameter named 'index' and returns an integer. If the *pos* array contains the parameter value return 1, otherwise return 0.

B. Create a public **constructor** that takes one integer named 'count'. Initialize *pos* to a new integer array of length count. Initialize the contents of the array to random integers, including the first entry. Each integer entry should be greater than the previous entry by a small random amount (e.g. +1...+10). All array values should be positive.

C. Create a public copy constructor that takes a reference to another BigList object (which will not be changed). Perform a deep copy i.e. the new object will use its own position array.

6. Objects (BigList Continued) – 15 points

D. Complete a public instance method **anyOverlap** that takes a reference to another BigList and returns a boolean. Return true if the two biglists have any ones in the same position, false otherwise i.e. there's a integer value that exists in both arrays. You may assume the position values are in increasing order.
BigList continues...

E. Your buddy claims their BigList equals method below returns *true* only when two big lists are identical (ones in the same position).

BigList continues...

```
      public boolean equals(BigList o) { // works!??
        if(o == null) return false;
        for(int i =0;i < pos.length; i++)
           if(this.pos[i] != o.pos[i] )
              return false;
        return true;
      }
```

Complete the test code below to show that the equals method is broken: Add data values to complete the example so that list1.equals(list2) returns true when in fact the lists are different.

```
      BigList list1 = new BigList(0);
      BigList list2 = new BigList(0);

      list1.reset({3,7,13}); // The reset method assigns pos to the given int array.

>>    list2.reset({______________});
      boolean b = list1.equals(list2); // equals method returns true! (should be false)
```

F. Your buddy claims it's OK to add the 'static' modifier to BigList's *pos* variable. They even created a short example to show you that it works! Modify their example below: Insert a very small amount of code to demonstrate to your buddy that adding static to *pos* is incorrect. Assume *getV* works as specified in Q5.

```
public class BigListExample {
      public void static main(String[] args) {
      // Demonstrates that I can add static to pos ! :-) ??
            BigList t = new BigList(0);
            t.reset({4,7});
            System.out.println(t.getV(7)); // prints one as expected
            System.out.println(t.getV(6)); // prints zero as expected
      }
}
```

Empty Page.

Scratch paper