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;   2
a2 = a3;  3
a3 = a1;   2
System.out.print(a2 == a3);
System.out.print(a1 == a2);
```

false false

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

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

B extends A. / A is a superclass of B / B is a subclass of A

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

C) Encapsulation

(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**

(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];  << Makes an array of String pointers.
2   for(int i=0;i<4;i++)
3       data[i] = data[7-i];
4   String result = data[1];  << Just a pointer to a String.   Number of String objects created:0
5   System.out.println( result.toUpperCase() );   Exception thrown at line: 5
```

2. Lists, Stacks and Queues – 16 points

/* 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. */

```
public String peekTop() {
  if(arr.length==0) return null;
  return arr[ arr.length -1];
}
```

/* 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. */

```
public Stack getShortStack() {
    Stack result =new Stack();
    for(int i =0; i < arr.length;i++)
        if(arr[i].length() <32) result.push( arr[i]);
    return 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. */

```
public boolean anyEmpty() {
    for(int i =0;i<arr.length;i++)
        if(arr[i].length() ==0) return true;
    return false;
}
```

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) {
```

```
        if(A==null || B == null) return null;
        int w = A.length , h = A[0].length;
        int[][] out = new int[ w ] [ h ];
        for(int i=0;i<iw;i++)
                for(int j = 0;j <h;j++) {
                        int diff = 2*Math.abs( A[i][j] - B[i][j] );
                        if(diff>255) diff = 255;
                        out[i][j] = diff;
                }
        return out;
}
```

CONFLICT. A picture is represented as a rectangular 2D int array. The hexadecimal integer, *rrggbb,* at array[x][y] encodes the red-green-blue components (8 bits each) at pixel position (x,y). Valid integer values are in the range 0(black) to 0xffffff (white). The blue component is stored in the lowest 8 bits (0-7), green in the next 8 bits (8-15) and red in the next 8 bits (16-23). These expressions may be useful: (x & 0xff), (x>>8) & 0xff , x>>16 .Complete the class method below to create a new 2D array of the same size.

- If the image parameter `A` is `null`, immediately return `null`.
- For each pixel, each output value at [x][y] is true if the blue component value at (x,y) is more than the given threshold value.
- If the output value at [x][y] is true also set the correspond pixel color at (x,y) in the original image to black.

```
public static boolean[][] moreBlueThanAverage(int[][] A,int threshold) {
        if(A==null) return null;
        int w = A.length , h = A[0].length; // (could be other way around; but that's OK)
        boolean[][] result = new boolean[ w][h];
        for(int i =0;i<w;i++)
                for(int j=0;j<h;j++)
                        result[i][j]=A[i][j] > threshold;
                        if(result[i][j])
                                A[i][j] = 0;
                }
        return result;
}
```

4. X marks the spot – 21 Points

```
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;}
}
```

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 60 000 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.

```
public class Secret extends Geo
  private long showAfter;
 [public] Secret() {
   showAfter = System.currentTimeMillis() + (int) (Math.random() * 60000);
}
 [public] String toString() {
  if(System.currentTimeMillis() > showAfter)
    return getX() + "," + getY();
  return "";
}
```

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) {
            Secret[] data = Factory.create();
            TextIO.getln();
            for(int i =0;i < data.length;i++) {
                    String pos = data[i].toString();
```

```
            if(pos.length() >0) TextIO.putln(pos);
        }
```

5. Objects I – 18 Points

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.

```
public int getV(int index) {
        for(int i =0; i < pos;i++)
                if(pos[i] == index) return 1;
        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.

```
public BigList(int count) {
        pos = new int[count];
        int x = 0;
        for(int i =0; i < count; i++) {
                x += 1+(int)(Math.random() * 10 );
                pos[i] = x;
        }
}
```

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.

```
public BigList(BigList src) {
        pos = new int[src.data.length];
        for(int i =0;i < pos.length;i++)
         pos[i] = src.data[i];
}
```

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.

```
public boolean anyOverlap(BigList other) {
        for(int i =0; i < pos.length;i++)
                if( other.getV( pos[i] ) == 1) return true;
        return false;
}
```

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

```
        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({3,7,13,15}); // must start with 3,7,13 and be longer
        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});
                new BigList(0);
// pos now points to an empty array say getV(7) returns zero
                System.out.println(t.getV(7)); // prints one as expected
                System.out.println(t.getV(6)); // prints zero as expected
        }
}
```

CONFLICT You're creating match-making site to help people find others with similar interests. Your programming buddy has already started the class to represent each persons' interest in a list of topics:

```
public class List {
      private boolean[] data; // Hey... Should this be static? - (Bud!)
      public void reset(boolean[] newValues) {
          this.data = newValues;
      }
```

A. Create a public **instance** method named '**find**' that takes two integer parameters named 'min' , 'max' and returns an integer. Return the index of the first true entry of the *data* array between indices *min* and *max* inclusive. Return -1, if no such entry exists. You can assume 0<=min<=max<data.length.

```
int find(int min, int max) {
      for(int i=min;i<=max;i++) if(data[i]) return i;
      return -1;
}
```

B. Create a public **constructor** that takes one integer named 'size'. Initialize *data* to a new boolean array of length *size*. Some of the entries will be set to true: Randomly set *size*/10 data entries to true. Assume *size* is divisible by 10 and be careful that you do not doubly count the setting the same data entry more than once.

```
public List(int count) {
      data = new boolean[count];
      int c = count / 10;
      while(c>0) {
            int i = (int)(Math.random() * count);
            if(!data[i]) {data[i] = true; c--;}
      }
}
```

C. Create a public copy constructor that takes a reference to another List object (which will not be changed). The new object will use its own array.

```
      public List(List src) {
            this.data = new boolean[src.data.length];
            for(int i=0;i<data.length;i++)
                  data[i] = src.data[i];
      }
```

6D. The public instance method **countOverlap** that takes a reference to another List and returns an integer. Return the number of interests (data[i] is true) shared by this list and the given list.

```
public List int countOverlap(List o) {
      int c = 0;
  int max = Math.min(o.data.length,this.data.length;
      for(int i = 0;i<max;i++)
            if(o.data[i] && data[i]) c++;
}
```

E. Your buddy claims their List equals method below returns *true* only when two people's lists are identical (true in the same positions).

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

```
      List list1 = new List(0);
      List list2 = new List(0);
      list1.reset({false,true });  // The reset method assigns data to the given boolean array.
>>    list2.reset({false,true, true, });  // must start with false,true
      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 List's *data* 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 *data* is incorrect. The method *find(min,max)* is specified in Q5.

```
public class ListExample {
      public void static main(String[] args) {
      // Demonstrates that I can add static to data ! :-) ??
            List t = new List(0);
            t.reset({false,true});
List l = new List(0); l.reset({false,false}); // data reset for all Lists
            System.out.println(t.find(0,1)); // prints 1 as expected
            System.out.println(t.find(0,0)); // prints -1 as expected
      }
}
```