1. (2 points each, 30 points total) Short answer questions. Place your answers on the attached answer sheet. For code samples state the output. If the code would cause a syntax error then answer "syntax error". If it would cause a runtime exception then answer "exception". If it would result in an infinite loop answer "infinite loop".

A. What is output by the following code when the client code is executed?

```java
public int a(int y){
    y--;  
    int x = y * 2 - 1;
    return x;
}

// client code
int x = 4;
int y = a(x);
System.out.print(x + " " + y);
```

B. What is output by the following code?

```java
ArrayList<Integer> vals = new ArrayList<Integer>();
ArrayList<Integer> other = vals;
other.add(4);
vals.add(2);
vals.add(5);
System.out.println(other);
```
C. What is output by the following code when the client code is executed?

```java
public void c(int[] list){
    list[list.length - 1] += 2;
    list[0]--;
}

// client code
int[] data = {3, 5, 2};
c(data);
System.out.println(Arrays.toString(data));
```

D. What is output by the following code?

```java
ArrayList<String> names = new ArrayList<String>();
System.out.println(names.size());
```

E. Briefly explain why the following code will not compile.

```java
Object obj = new ArrayList<String>();
obj.add("Kelly");
System.out.println( obj.toString() );
System.out.println( obj.equals(obj) );
```

F. Does the following code result in syntax error, a runtime error (exception), or neither?

```java
int k = 10;
String[] titles = new String[k + 2];
System.out.println( titles[ titles.length / 2 ].length() );
```
For questions G - N consider the following classes.

```java
public abstract class IPod {
    private int memory;
    
    public IPod() { memory = 40; }
    
    // pre: m > 0
    public IPod(int m) { memory = m; }
    
    public abstract String memoryType();
    
    public String toString() { return memoryType() + " " + memory; }
}

public class Touch extends IPod {
    public Touch() {}
    public Touch(int s) {
        super(s);
    }
    
    public String memoryType() { return "flash"; }
    
    public boolean hasTouchScreen() { return true; }
}

public class Classic extends IPod {
    public Classic(int s) { super(s); }
    
    public String memoryType() { return "hard drive"; }
    
    public int numColors() { return 2; }
}

public class HP extends Classic {
    public HP(int s) { super(s); }
    
    public int numColors() { return 1; }
}

G. Briefly explain why the following code will not compile:

IPod p = new IPod(-20);
H.  What is output by the following code?

    Touch t = new Touch(30);
    System.out.println( t.toString() );

I.  What is output by the following code?

    IPod[] ps = new IPod[3];
    ps[0] = new Touch();
    ps[1] = new Classic(20);
    ps[2] = new HP(30);
    for(IPod p : ps)
        System.out.println(p);

J.  State if the following declarations are valid or invalid (meaning they cause a syntax error).
    1 point each

    Classic c2 = new Touch(20);
    Classic c3 = new HP(20);

K.  State if the following declarations are valid or invalid (meaning they cause a syntax error).
    1 point each

    Object obj = new HP(20);
    HP h1 = new Classic(30);

L.  What is output by the following code?

    Object obj2 = new Touch(30);
    System.out.println(obj2);

M.  What is output by the following code?

    Classic c2 = new HP(30);
    System.out.println( c2.memoryType() + " " + c2.numColors() );

N.  What is output by the following code?

    IPod i2 = new Classic(30);
    System.out.println( i2.memoryType() + " " +
        ((Classic) i2).numColors() );

O.  How many interfaces can a class in Java implement?
2. The `GenericList` class. (25 points) In lecture, to demonstrate encapsulation and the syntax for building a class in Java, we developed a `GenericList` class to represent a list of objects. As discussed in class the internal storage container may have extra capacity.

Complete a method for the `GenericList` class named `interleave`. This is an instance method in the `GenericList` class that creates and returns a new `GenericList` object that has the elements of the calling `GenericList` and another `GenericList` sent as an explicit parameter interleaved.

The method header is:

```java
/*
pre: other != null
post: return a new GenericList that has the elements of this GenericList and other interleaved.
The size of the returned list equals 2 times the minimum of the size of this list and other.
The elements of the returned list alternate between the values of this list, then other, then this list, then other, and so forth.
Neither this nor other is altered as a result of this method call.
*/
public GenericList interleave(GenericList other) {

Examples of calls to `interleave`: (Assume values shown are Strings.)

```
[A, B, C].interleave([Q, M, P]) -> [A, Q, B, M, C, P]
[].interleave([Q, M, P]) -> []
[A, B, C].interleave([]) -> []
[A, B, C, D, E, F].interleave([Q, M, P]) -> [A, Q, B, M, C, P]
[].interleave([]) -> []
[A].interleave([]) -> []
[A].interleave([Q]) -> [A, Q]
```

You may not use any other methods in the `GenericList` class except the default constructor unless you define and implement them yourself. Recall that the `interleave` method is in the `GenericList` class so you have access to all `GenericList` objects' private instance variables.
You may not use objects or methods from other Java classes other than native arrays and the Math class. If you want to use any other `GenericList` methods, other than the default constructor, you must define and implement them yourself.

Recall the `GenericList` class:

```java
public class GenericList{
    private Object[] container;
    private int size;

    public GenericList() {
        container = new Object[10];
        size = 0;
    }
}
```

Complete the following instance method for the `GenericList` class.

```java
/*
pre: other != null
post: return a new GenericList that has the elements of this GenericList and other interleaved. The size of the returned list equals 2 times the minimum of the size of this list and other. The elements of the returned list alternate between the values of this list, then other, then this list, then other, and so forth. Neither this nor other is altered as a result of this method call.
*/
public GenericList interleave(GenericList other) {
    Complete on next page
    Complete on next page
    Complete on next page
    Complete on next page
    Complete on next page
    Complete on next page
```
public GenericList interleave(GenericList other) {
    assert other != null;
}
3. (20 points total) 2D Arrays. Write a method to determine the number of Color objects in a neighborhood of a 2D array of Color objects that have a greater intensity of green and blue than a Color object at a given location at the center of the neighborhood.

```java
public int numGreaterBlueGreen(Color[][] grid, int row, int col, int size) {

// Color objects store the intensity of the red, green, and blue components of a color. The intensity varies
// between 0 and 255. The methods for the Color class are:

public class Color {
    // All three of these methods will
    // return a value between 0 and 255.
    // The larger the value, the greater the intensity.

    public int getRed() // return red intensity of this Color
    public int getGreen() // return green intensity of this Color
    public int getBlue() // return blue intensity of this Color
}
```

The size parameter specifies how large an area to search around the cell at location row, col. For example if the array is 6 by 7, row = 2, col = 3, and size = 2, the neighborhood to search is shown below. The asterisk marks the center cell.
Note, based on row, column, and size there may be cells in the neighborhood that do not exist. Your methods must ignore these cells.

**You may not use any other Java classes except the Color and Math classes and native arrays.**

```
/*
pre: grid != null, no elements in grid are null,
     grid is a rectangular matrix, 0 <= row < grid.length,
     0 <= col < grid[0].length, size > 0
post: return the number of Color objects in grid that have a greater
      intensity of green and blue than the Color at location
      grid[row][col] that are in the neighborhood of the given size
      centered around the cell at location row, col.
*/
public int numGreaterBlueGreen(Color[][] grid, int row, int col, int size) {
    // DO NOT CHECK THE PRECONDITIONS. ASSUME THEY ARE MET.
```
A. (15 points) Complete an instance method in the NameRecord class that returns true if the ranks for two NameRecord objects (the calling object and the explicit parameter) differ by no more than some max value for every decade.

// pre: other != null, this.numDecades() == other.numDecades(),
//       maxDiff >= 0
// post: return true if the ranks for this NameRecord and other differ by
//       no more than maxDiff in their rank for every decade,
//       false otherwise.
//       A rank of 0 (unranked) is treated as a rank of 1001.
private boolean inSync(NameRecord other, int maxDiff) {

Recall the first integer is the rank for the decade 1900 - 1909, the second integer is the rank for the decade from 1910 - 1919, and so forth. Recall a lower number (except for 0's) means a name was more popular. A value of 0 indicates the name had a rank greater than 1000 for the decade. For this question you will assume a 0 indicates a rank of 1001.

Consider the following example:

The ranks for the name Margaret are: 3 4 5 8 14 23 54 101 95 107 112
The ranks for the name David are: 29 30 22 11 6 5 2 4 5 11 13

The absolute values of the difference between the ranks for each decade are:
26 26 17 3 8 18 52 97 90 96 99

If the max allowed difference were 100 the method would return true for those two NameRecord objects.
If the max allowed difference were 50 the method would return false for those two NameRecord objects.

If a name is unranked for a decade the getRank method from the NameRecord class returns a 0. For this problem you will treat all 0's as a rank of 1001. So for example, if one NameRecord has a rank of 950 and the other is unranked. The difference is calculated to be 51. (1001 - 950 = 51)

The NameRecord class you will use on this question is as follows:

public class NameRecord {

    private String name;
    private ArrayList<Integer> ranks;
    // ranks.size() always equals numDecades()

    // returns this NameRecords name
    public String getName()

    // Return the number of decades of data for this NameRecord.
    // Includes decades the name was not in the top 1000.
    public int numDecades()
// NameRecord class continued

// get the rank of this NameRecord for the given decade
// first decade of data is 0, next is 1, and so forth
// pre: 0 <= decade < numDecades()
public int getRank(int decade){

From the ArrayList class:

public int size() - returns the number of elements in this ArrayList

public E get(int pos) - returns the element at the specified position in this list. pre: 0 <= pos < size()

You are not allowed to use any other methods from the NameRecord or ArrayList classes other than those shown above and the Math class. You may not use any other Java classes. (You may use native arrays if you would like.)

Complete the inSync method from the NameRecord class.

// pre: other != null, this.numDecades() == other.numDecades(),
//      maxDiff >= 0
// post: return true if the ranks for this NameRecord and other differ by
//       no more than maxDiff in their rank for every decade,
//       false otherwise.
//       A rank of 0 (unranked) is treated as a rank of 1001.
private boolean inSync(NameRecord other, int maxDiff) {
    assert other != null && this.numDecades() == other.numDecades() 
    && maxDiff > 0 : "failed precondition, method inSync";

// More room for inSync if necessary on next page
4B (10 points) Complete an instance method in the Names class that returns an ArrayList<String> of all the names that are in sync with a given name based on a maximum allowed difference.

// pre: name != null, maxDiff >= 0
// post: return an ArrayList<String> of names that are in sync with // the name. If name is not in this Names database return an empty // ArrayList.
// The returned ArrayList will NOT contain name.
public ArrayList<String> getNamesInSync(String name, int maxDiff) {

For example given a maximum allowed difference of 100 per decade, the name Margaret is in sync with these names (Note, the list does NOT contain Margaret, the original name.):

Anna, Catherine, Charles, David, Edward, Edwin, Frank, George, Henry, Jack, James, John, Katherine, Laura, Martin, Mary, Paul, Peter, Raymond, Richard, Robert, Samuel, Thomas

The Names class you will use on this question is as follows:

public class Names {

    // the NameRecods in this Names object.
    private ArrayList<NameRecord> nameList;

    // return the NameRecord associated with name, ignoring case
    // if there is no NameRecord based on name, return null
    public NameRecord getName(String name)

}

From the ArrayList class:

public ArrayList() - construct an empty ArrayList

public int size() - returns the number of elements in this ArrayList

public E get(int pos) - returns the element at the specified position in this list. pre: 0 <= pos < size()

public boolean add(E obj) - add obj to the end of this ArrayList. Always returns true.

You are not allowed to use any other methods from the NameRecord, Names, String, or ArrayList classes other than those shown above and on part A of the question. You may not use any other Java classes. (You may use native arrays if you would like.)

Complete the method on the next page.
Complete the following instance method for the `Name` class.

// pre: name != null, maxDiff >= 0
// post: return an ArrayList<String> of names that are in sync with
// the name. If name is not in this Names database return an empty
// ArrayList.
// The returned ArrayList will NOT contain name.
public ArrayList<String> getNamesInSync(String name, int maxDiff) {
    assert name != null && maxDiff >= 0 : "Failed precondition, " + "method getNamesInSync.";
Question 1 answer Sheet

Name______________________________

A. ________________________________  I. ________________________________  
   1. 

B. ________________________________  2. 
   J. ________________________________  
   1. 

C. ________________________________  2. 
   K. ________________________________

D. ________________________________  L. ________________________________

E. ________________________________  M. ________________________________

F. ________________________________  N. ________________________________

G. ________________________________  O. ________________________________

H. ________________________________