PROBLEM 1 : (***Short and To the Point (14 points)***)

A. For each of the following object-oriented programming terms, summarize the distinction between the two terms. Your answer should be *brief.*

1. class vs. object
2. the constructor vs. any other class method

B. Write a method `majority` that takes three boolean inputs and returns the most common value. For example, `majority(false, false, true)` should return `false`, while `majority(false, true, true)` returns true.

C. The following method, `floorRoot`, was designed to compute the largest integer whose square is no greater than N, where N is assumed to be a positive number. (If N is 5, then the procedure should report the value 2.) Find and correct the error.

```
/* returns the largest integer whose square is no greater than n */
public int floorRoot(int n)
{
  int x = 0;
  while (x * x <= n)
  {
    x = x + 1;
  }
  return x;
}
```

D. What is the value of name after the following code executes?

```
String name = "Richard H. Brodhead"
name = name.substring(name.indexOf(" ")+1) + ", " +
          name.substring(0,name.indexOf(" "));
```

PROBLEM 2 : (***Loop-de-loop-de-loop (20 points)***)

Consider solving the problem of finding all the maximum values in an array and moving them to the front of the array, while keeping all the other elements in the array in the same order.

For example if the array was 7 8 9 3 2 9 5

Then after moving the maximum values (in this case two 9's) to the front of the array, the array values would be 9 9 7 8 3 2 5, note the non-max items are still in the same order.

We will solve this problem in two parts.

PART A. First, given an array `numbers` (assume it has been created and initialized with values), compute `max`, the maximum value in the array, and `maxCount`, the number of occurrences of `max` in the array. Do not modify the array for this part.

```
int [] numbers;    // code to initialize not shown
int max;           // maximum value in array numbers
int maxCount = 0;  // number of occurences of max in numbers

// TODO: compute max and maxCount for the array numbers
```

PART B. Consider the following code that assumes you have computed max and maxCount correctly in part A. This code puts all the `max` values in the front of the array, with all other elements still in the same order.

```
// move all max's to the front of the array numbers,
// keeping non-max elements in the same order

// LOOP 1
int index = numbers.length - 1;
for (int k=numbers.length-1; k>= 0; k--)
{
   if (numbers[k]!= max)
   {
      numbers[index] = numbers[k];
      index = index- 1;
   }
}
```

```
// LOOP 2
for (int k=0; k<maxCount; k++)
{
    numbers[k] = max;
}
```

For the following questions, assume numbers is initialized as follows before Loop 1.

```
numbers = {3, 8, 8, 2, 4, 8, 1, 7};
```

Q1. Give a meaningful loop invariant for the loop(s) you wrote in Part A.

Q2. What are the contents of `numbers` after four iterations of Loop 1?

Q3. What are the contents of `numbers` after Loop 1 completes?

Q4. Give a meaningful loop invariant for Loop 1 in Part B.

Q5. What are the contents of the array after Loop 1 and Loop2 complete?

PROBLEM 3 : (***Birthdays to Weight Classes (24 points)***)

In this problem, you will write methods to compute the proper weight class for an item of a given weight. Items need to go in the smallest weight class larger than their weight. Items bigger than all weight classes are classified as "heavyweight."

A. The weights will be easier to compare if represented as integers rather than text. In this step, you will write methods to convert from a weight description to the number of ounces.

Weight is given in pounds and ounces in the following form `"1 lb 6 oz"`. There are *16 ounces in a pound*, and both the number of pounds and ounces are ≥ 0. Below are some examples of calls to `weightToOz` and the appropriate return values.

```
weightToOz("0 lb 14 oz") → 14
weightToOz("4 lb 4 oz") → 68
weightToOz("1 lb 20 oz") → 36
```

Note:

- The `Integer.parseInt` method converts a String to an `int`. For example, `Integer.parseInt("408")` → 408.

Complete `weightToOz` below.

```
/**
 * Returns specified weight in ounces
 * @param weight nonnegative weight in the form "n lb m oz"
 * For example "0 lb 14 oz"
 */
public int weightToOz(String weight)
{
```

B. The external interface to your code requires a text representation of the weights. Write a method, *ozToWeight* to convert from a nonnegative number of ounces to weight of the form `"n lb m oz"`.

Examples:

```
    ozToWeight(0) → "0 lb 0 oz"
    weightToOz(40) → "2 lb 8 oz"
```

```
/**
 * Returns weight as "N lb M oz". For example,
 * ozToWeight(33) should return "2 lb 1 oz"
 * @param n a value that is greater than or equal to 0
 */
public String ozToWeight(int n)
{
```

C. Write the method `readWClasses` below that reads data from a file and returns an array of `String`s, one for each weight class in the file. The first line of the file contains the number of weight classes. Each additional line of the file has the name of the weight class followed by its weight limit. Every word is separated by exactly one space.

A sample data file is shown below with four weight classes.

```
4
Lightweight 1 lb 8oz
Featherweight 0 lb 4 oz
Cruiserweight 17 lb 3oz
Middleweight 5 lb 2 oz
```

Given a `Scanner` initialized to the file above, `readWClasses` should return

```
{"1 lb 8 oz", "0 lb 4 oz", "17 lb 3 oz", "5 lb 2 oz"}
```

Complete `readWClasses` below.

```
/**
 * Reads weight class information from the file represented by
 * the parameter Scanner and returns it in an array
 */
public String[] readWClasses(Scanner input) {
```

D. Given a `String weight`, and `String[] classes`, a list of weight classes, return a String, the value of the smallest weight class greater than `weight`. For example, if

```
String[] wc = {"1 lb 8 oz", "0 lb 4 oz", "17 lb 3 oz", "5 lb 2 oz"};
```

then

```
    nextWClass("4 lb 13 oz", wc) → "5 lb 2 oz"
    nextWClass("16 lb 25 oz", wc) → "Heavyweight"
    nextWClass("0 lb 0 oz", wc) → "0 lb 4 oz"
```

Notes:

- You can and should use `weightToOz` and `ozToWeight` in your solution to convert to and from a integer to string representations of weights.
- Use the `Arrays.sort` or `Collections.sort` to rearrange an array or `ArrayList`, respectively, of integers in increasing order.
- If `weight` is larger than all given classes, return `"Heavyweight"`.
- The reasoning for this problem is similar to that of the Birthday APT.

Complete `nextWClass` below.

```
/**
 * Returns the next weight class that is appropriate for an item
 * with a given weight. That is, return the smallest weight class
 * greater than or equal to weight.
 * @param weight nonnegative weight in the form "n lb m oz"
 * @param classes unsorted weights in the form "n lb m oz"
 */
public String nextWClass(String weight, String[] classes)
{
```

PROBLEM 4 : (***Bouncers Revisited (12 points)***)

Given the definition of `BouncingBall.java` from class (also included at the end of this test), create a new class `ColorfulBall` that is a subclass of `BouncingBall`.

ColorfulBall should have different behavior than `BouncingBalls` in the following ways:

1. `ColorfulBalls` should start with a *random* color.

2. `ColorfulBall`s should shift colors when they hit a wall. For example, if the ball's original color in terms of (*red*, *green*, *blue*) was (255, 127, 0), then the color of the ball should be (0, 255, 127) after hitting a wall once, (127, 0 255) after the second bounce, and back to the original color after the third bounce. The RGB values shift to the right and wrap around.

Complete `ColorfulBall` below. You should complete the constructor and only add those methods and variables that are necessary.

```
public class ColorfulBall extends BouncingBall {
  /**
   * Create a bouncer
   *
   * @param start initial position
   * @param velocity amount to move in x- and y-direction
   */
  public ColorfulBall(java.awt.Point center, java.awt.Point velocity)
  {
```

Throughout this test, assume that the following classes and methods are available. These classes are taken directly from the material used in class.

```
public class String {
    // Returns the length of this string.
    public int length ()
    // Returns a substring of this string that
    // begins at the specified beginIndex and
    // extends to the character at index
    // endIndex - 1.
    public String substring (int beginIndex,
                             int endIndex)
    // Returns a substring of this string that
    // begins at the specified beginIndex and
    // extends to the end of the string.
    public String substring (int beginIndex)
    // Returns position of the first
    // occurrence  of str, -1 if not found
    public int indexOf (String str)
    // Returns the position of the first
    // occurrence of str after index start
    // returns -1 if str is not found
    public int indexOf (String str, int start)
    // returns character at position index
    public char charAt(int index)
    // returns true if str has the exact
    // same characters in the same order
    public boolean equals(String str)
    // returns the string as an array
    // of characters
    public char [] toCharArray()
}

public class Arrays {
    // Sorts the specified array into
    // ascending numerical order
    public static void sort(int[] a)
}

public class Integer {
    // Returns the argument as a signed integer.
    public int parseInt(String s)
}

public class Random {
    // Create a new random number generator
    public Random()
    // Returns a pseudorandom, uniformly
    // distributed value in [0,n)
    public int nextInt(int n)
}
```

```
public class Color {
    // Creates a color with the specified red,
    // green, and blue values in the range
    // (0 - 255)
    public Color(int r,int g,int b)
    // Returns the red component
    public int getRed()
    // Returns the green component
    public int getGreen()
    // Returns the blue component
    public int getBlue()
}

public class ArrayList {
    // Constructs an empty list
    public ArrayList ()
    // Returns the number of elements
    public int size ()
    // Returns element at position index
    public Object get (int index)
    // Replaces the item at position index
    // with element.
    public Object set (int index, Object element)
    // Appends specified element to end of
    // this list.
    public boolean add (Object o)
}

public class Scanner
{
    // Create Scanner that reads data from a file.
    public Scanner (File file)
    // Create Scanner that reads data from a string.
    public Scanner (String str)
    // Change delimiters used to separate items
    public void useDelimiter (String characters)
    // Check if more items are available
    public boolean hasNext ()
    // Get next delimited item as a string
    public String next ()
    // Get next line as a string
    public String nextLine ()
    // Get next delimited item as an integer value
    public int nextInt ()
    // Get next delimited item as a Double value
    public double nextDouble ()
}
```