PRACTICE TESTS CPS006 FALL2009

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
```

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 `Strings`, 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)
{
```

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 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 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 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 ()
}
```

PROBLEM 1 : (*Mystery Repeat Repeat Repeat:* (22 pts))

PART A *(12 pts)*:

Consider the following *Mystery* method.

```
public int Mystery (String phrase)
{
   int pos = phrase.indexOf("e");
   int pos2 = phrase.indexOf("e",pos+1);
   System.out.println(phrase.substring(pos,pos+3));
   return phrase.substring(pos2).length();
}
```

A. What type is the return value for the method *Mystery*?

B. How many parameters are there?

C. For the call `Mystery(''GoeDukeiea'')`, list first what is printed as output and list second the return value.

D. For the call `Mystery(''eeeee'')`, list first what is printed as output and list second the return value.

E. Describe in words what the method *Mystery* does.

F. Give an example value for phrase that will cause the function *Mystery* to crash. Explain why it crashes.

PART B *(10 pts)*: Consider the following *Mystery2* method.

```
public int Mystery2(ArrayList<Integer> numbers)
{
   int x = 0;
   for (Integer num: numbers)
   {
      if (num < 6)
      {
         x += num;
      }
   }
   return x;

}
```

A. List the names of the local variables in *Mystery2*.

B. What is the return type of *Mystery2*?

C. Suppose *values* is an `ArrayList<Integer>` and has the values 8, 4, 9, 2 and 3 stored in this order from position 0 to position 4. What is the return value of the call `Mystery2(values)`?

D. Describe in words what the method *Mystery2* does.

E. Explain why the the ArrayList is of type Integer instead of type int.

PROBLEM 2 : (***Don't forget the middle:*** **(8 pts))**

Complete the method *InsertMiddle* that is given two string parameters. The first string is a name consisting of a first name and a last name separated by one blank. The second string is a middle name. This method returns the name with the middle name inserted between the first and last name.

For example, *InsertMiddle("Sarah Forth", "Go")* would return the string "Sarah Go Forth".

You can assume that there is exactly one blank in *name*, between the first name and last name.

```
public String InsertMiddle(String name, String middle)
{

}
```

PROBLEM 3 : (***Living on Campus:*** **(10 pts))**

Consider the following two classes.

```
public class Dorm {

   private String myName;
   private int myNumRooms;
   private int myNumFloors;

   public Dorm(String name, int numRooms, int numFloors)
   {
      myName = name;
      myNumRooms = numRooms;
      myNumFloors = numFloors;
   }

   public String getName()
   {
      return myName;
   }
```

```
   Dorm dorm1 = new Dorm(''Keohane'', 80, 4);
   System.out.println(dorm1.getName());

// 2.
   Dorm dorm2 = new Dorm(''Blackwell'', 60, 2);
   System.out.println(dorm2.myName);

// 3.
    Dorm dorm3 = new AthleticDorm(''Nelson'', 60, 3,
       ''basketball'');
    System.out.println(dorm3.getName());

// 4.
    AthleticDorm dorm4 = new
        AthleticDorm(''Blumenhurst'', 20, 2, ''volleyball'');
    dorm4.setSport(''golf'');
    System.out.println(dorm4.getName());
```

PROBLEM 4 : (*Checkmate*(42 pts))

Consider the class *Player* shown below to represent a chess player. Chess is a board game played by two people. A *Player* stores information about a chess player including their name, their rank (a number that is larger means the player is higher ranked, and 0 means they are unranked.), and their grade in school, 1-13, with 13 meaning they are out of high school. There is no grade higher than 13.

```
public class Player {

   private String myName;     // name of player
   private int myRank;        // rank of player
   private int myGrade;       // grade player is in, from 1-13

   public Player (String name, int rank, int grade)
   {  // code not shown    }

   // return name of player
   public String getName()
   {   // code not shown   }

   // return rank of player
   public int getRank()
   {   // code not shown    }

   // return grade of player
   public int getGrade()
```

```
   {  // code not shown    }

   // increment grade of player by 1
   // except if grade of player is 13, do not change grade.
   public void incrementGrade()
   {   // code not shown    }

   // change rank of player to "rank"
   public void setRank(int rank)
   {  // code not shown    }
}
```

PART A *(18 pts)*:

PART A1 *(4 pts)*: Complete the constructor for the class Player.

```
   public Player (String name, int rank, int grade)
   {



   }
```

PART A2 *(14 pts)*:

Complete the code for the following methods.

```
   // return name of player
   public String getName()
   {

   }

   // return rank of player
   public int getRank()
   {

   }

   // return grade of player
   public int getGrade()
   {

   }

   // increment grade of player by 1
   // except if grade of player is 13, do not change grade.
   public void incrementGrade()
```

```
{



}

// change rank of player to "rank"
public void setRank(int rank)
{


}
```

PART B *(24 pts)*:

Consider the ChessTournament class that is listed below with an example.

```
public class ChessTournament {

   private ArrayList<Player> myPlayers; // list of all chess players

   public ChessTournament (Scanner input)
   {  // code not shown  }

   public static void main(String[] args) throws FileNotFoundException
   {
      String inputFileName = "chessdata.txt";
      FileReader reader = new FileReader(inputFileName);
      Scanner in = new Scanner(reader);
      ChessTournament Feb8 = new ChessTournament(in);

      System.out.println("Grade 5 highest player is: " + Feb8.highestPlayerInGrade(5));
      System.out.println("Grade 9 highest player is: " + Feb8.highestPlayerInGrade(9));
      System.out.println("Grade 4 highest player is: " + Feb8.highestPlayerInGrade(4));
      System.out.println("Number players with rank between 400 and 700 is: "
            + Feb8.NumberPlayersBetween(400, 700));

      ArrayList<String> highRankPlayers = Feb8.PlayersWithRankGreater(600);
      System.out.println("Name of players rank greater than 600: ");
      for (String name: highRankPlayers)
      {
         System.out.print(name + " ");
      }
   }

   // returns the number of players whose rank is between minRank and maxRank inclusive
   public int NumberPlayersBetween (int minRank, int maxRank)
```

```
   {      // code not shown    }

   // returns the name of the highest ranking player in the given grade
   public String highestPlayerInGrade(int grade)
   {      // code not shown    }

   // returns an ArrayList of names of players whose rank is greater than rank
   public ArrayList<String> PlayersWithRankGreater(int rank)
   {   // code not shown  }
}
```

Here is a sample data file called chessdata.txt.

```
Narten 680 5
Lapidus 956 5
Ward 550 5
Smith 430 3
Yee 800 4
Parker 0 8
Kumar 0 4
Guilak 758 5
```

Here is the corresponding output when the program is run with this data file.

```
Grade 5 highest player is: Lapidus
Grade 9 highest player is: No Player in this grade.
Grade 4 highest player is: Yee
Number players with rank between 400 and 700 is: 3
Name of players rank greater than 600:
Narten Lapidus Yee Guilak
```

PART B1 ***(6 pts)***:

Note that the Scanner input is already bound to a file in main. The file is in the following format. Each line in the file has the name of a player (containing no blanks), the rank of the player as an integer and the grade of a player as an integer.

Complete the constructor for the ChessTournament class below. (hint: what do you need to construct with *new*?)

```
   public ChessTournament (Scanner input)
   {

   }
```

PART B2 *(6 pts)*:

Complete the method *NumberPlayersBetween* that has two parameters *minRank* and *maxRank* and returns the number of players whose rank is between *minRank* and *maxRank* inclusive.

For the example shown earlier in which Feb8 is a ChessTournament variable, the call `Feb8.NumberPlayersBetween(400, 700))` returned 3.

```
// returns the number of players whose rank is between minRank and maxRank inclusive
public int NumberPlayersBetween (int minRank, int maxRank)
{

   }
```

PART B3 *(6 pts)*:

Complete the method *highestPlayerInGrade* that has a grade as a parameter and returns the name of the highest ranking player in that grade, or returns the string "No Player in this grade" if there are no players in that grade.

See the three examples earlier for grade 5 (Lapidus), grade 9 (No player in this grade) and grade 3 (Yee).

```
// returns the name of the highest ranking player in the given grade
public String highestPlayerInGrade(int grade)
   {

   }
```

PART B4 *(6 pts)*:

Complete the method *PlayersWithRankGreater* that has one parameter, a rank, and returns an ArrayList of names of players with that rank.

See the previous example in which Feb8 is a ChessTournament object tied to the given data file and the call *Feb8.PlayersWithRankGreater(600)* returned an ArrayList with the names of the 4 players whose rank is greater than 600 (Narten Lapidus Yee Guilak).

```
// returns an ArrayList of names of players whose rank is greater than rank
public ArrayList<String> PlayersWithRankGreater(int rank)
{

   }
```

```
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, returns -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 ArrayList
{
    // Constructs an empty list
    public ArrayList ()

    // Returns the number of elements in this list.
    public int size ()

    // Returns element at position index in this list.
    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 int nextDouble ()
}
```