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*.

IGNORE PROBLEM 1

1. class vs.
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)***)

IGNORE PROBLEM 3

In this problem, you will write methods to determine 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
```

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 `BouncingBall`s in the following ways:

1. `ColorfulBall`s 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)
  {
```

PROBLEM 1 : (*Blasting out DNA (24 points)*)

In this problem, we use a single strand of DNA to generate fragments of DNA. A DNA strand is made up of the nucleotides A, C, G and T. For example, a DNA strand might be "CGTA". Each symbol has a complement. A and T are complements of each other, and C and G are complements of each other. The complement of "CGTA" is the complement of each symbol, resulting in "GCAT".

We want to generate DNA fragments from a DNA strand and a reactant (one of the four nucleotides). A DNA fragment is generated by generating a string of complements, stopping randomly at the chosen reactant.

For example, suppose the DNA strand is "CGATACTCTGT" and the reactant is "A". Shown below are four copies of this DNA strand on the first line with the four possible fragments below it. Each symbol in the fragment is the complement of the corresponding symbol directly above it and each fragment has to end in the reactant A.

```
CGATACTCTGT   CGATACTCTGT   CGATACTCTGT   CGATACTCTGT
GCTA          GCTATGA       GCTATGAGA     GCTATGAGACA
```

In this problem, given a DNA strand and a reactant we will randomly generate an array of possible fragments. We will write this in three parts.

First we give you the method complement which given a string of one letter (C,G,T or A) gives you the complement letter. You may want to use this method.

```
public String complement(String str)
{
   if (str.equals("A"))
      return "T";
   if (str.equals("T"))
      return "A";
   if (str.equals("C"))
      return "G";
   return "C";
}
```

PART A. (8) Complete the method *findPositionOfAllOccurences* which has two String parameters: dna is a string of nucleotides and nuc is one nucleotide. This method returns an `ArrayList` of the positions of nuc in dna. For example, if dna is AGCATA and nuc is A, then return the ArrayList of integers 0, 3, 5, which are the positions of the A in the dna strand.

```
public ArrayList<Integer> findPositionOfAllOccurences(String dna, String nuc)
{
```

PART B. (8) Complete the method *createOneFragment* that is given a dna String and a position in the String and returns one fragment that ends at that position. The fragment is created by generating a new String in which each character is the complement of the character in the dna String. The fragment may be shorter then the dna String as it must stop at the lastposition parameter.

For example, suppose the dna string is "CGTAG" and the last position is 2, then the fragment returned is "GCA", since GCA is the complement of CGT and the fragment ends at position 2.

```
public String createOneFragment (String dna, int lastPosition)
{
```

PART C. (8) Complete the method *generateFragments* that has three parameters: dna is a strand of dna, reactant is a single nucleotide and number is the number of fragments to generate. This method generates fragments of different random lengths, all ending in the same reactant.

For example, suppose the dna string is "CGTATGT", the reactant is "A" and number is 4, then an array of four random fragments are generated that all end in A, which is the complement of "T". These four fragments might be "GCA", "GCATACA", "GCA", and "GCATA".

```
public  String [] generateFragments(String dna, String reactant, int number)
{
```

PROBLEM 2 : (*Extra curricular activities (28 points)*)

You are given the data from Dook University of the clubs and members of each club in the following format. The data for each club is on two lines. The first line is the name of the club. The next line is the name of all the members in the club, with members separated by a colon.

The sample data below shows the members of three clubs.

```
Indoor and Outdoor Tennis
Jeff Forbes:Hillary Rodham Clinton:Mary Lou Retton
Gourmet Cooking
Susan Rodger:Oprah Winfrey:Cay Horstmann:Mary Lou Retton:Owen Astrachan
Photography
Owen Astrachan:Oprah Winfrey:Mary Lou Retton
```

We will create the Club class to store and manipulate the data for one club and the DookClubs class to store and manipulate data on all the clubs.

```
public class Club
{
    private String myName;
    private Set<String> myMembers;

    public String getName() { return myName; }
    public Set<String> getMembers() { return myMembers; }
    // constructor not shown

}

public class DookClubs
{
   private ArrayList<Club> myClubs;

   // constructor and member functions not shown

}
```

PART A. (6) Complete the Club class constructor which is passed in the name of the club and the members in the club, in the format described earlier with members separated by colons.

```
public  Club(String name, String members)
{




}
```

PART B. (8) Complete the DookClubs class constructor which is passed in a Scanner that is ready to read from a file. The constructor reads all the club data from the file and stores the information in `myClubs`.

```
public  DookClubs (Scanner input)
{


}
```

PART C. (8) Complete the DookClubs member function *allMembers* that returns an ArrayList of all the people from Dook University in a club.

Using the datafile given earlier, the `ArrayList` would contain: Jeff Forbes, Hillary Rodham Clinton, Mary Lou Retton, Susan Rodger, Oprah Winfrey, Cay Horstmann, and Owen Astrachan.

```
public ArrayList<String> allMembers ()
{




}
```

PART D. (6) Complete the DookClubs member function peopleNotInClubs that takes an `ArrayList` of names and returns an `Arraylist` of all the people from students who are *not* in a club.

For example, if the ArrayList students contained the names: Michael Jackson, Owen Astrachan, Oprah Winfrey, and Peter Lange, then `peopleNotInClubs(students)` would return the `ArrayList` containing Michael Jackson and Peter Lange, as they are not in any clubs.

```
public ArrayList<String> peopleNotInClubs (ArrayList<String> students)
{

}
```

PROBLEM 3 : (***Sorted Pictures (18 points)***)

IGNORE Problem 3

In this problem, you will create a Pixmap *Command* that sorts a Pixmap by sorting each column of Colors by *luminance*. Luminance is a measure of the intensity of light as perceived by the human eye. White would have the most luminance while black would have the least. By sorting each column by luminance, the darkest colors will be on the top and brightest colors will be on the bottom. You should not change the value of any individual color, just its position in its column.

You computed luminance for the `WeightedGrayScale` filter as part of the Pixmap classwork. If you have a color (R, G, B) for the red, green, and blue values, the luminance is defined as $0.3R + 0.59G + 0.11B$.

PART A. (8) Complete the *LumComparator* class below. You will need to write:

1. the `luminance` method that returns the luminance of a `Color`
2. the `compare` method that should use the `luminance` method to return a negative integer, zero, or a positive integer if the first color is less than, equal to, or greater than the second in terms of luminance.

```
public class LumComparator implements Comparator<Color>
{
    // Computes the luminance of c where luminance is defined as
    // 0.3R + 0.59G + 0.11B
    private int luminance(Color c)
    {
```

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

IGNORE Problem 1

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)) IGNORE PROBLEM 2

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;
   }
```

```
   public int getNumRooms()
   {
      return myNumRooms;
   }

   public int getNumFloors()
   {
      return myNumFloors;
   }
}


public class AthleticDorm extends Dorm {

   private String mySport;


   public AthleticDorm(String name, int numRooms,
         int numFloors, String sport)
   {
      super(name, numRooms, numFloors);
      mySport = sport;
   }

   public String getName()
   {
      return super.getName() + ": the " + getSport() + " dorm";
   }

   public String getSport()
   {
      return mySport;
   }
   public void setSport(String sport)
   {
      mySport = sport;
   }
}
```

A : Which of the two classes is the superclass?

B : Consider the following 4 sections of code. State if the two lines of code are valid or contain an error. If they contain an error, then state what the error is. If they do not contain an error, then list the corresponding output.

```
// 1.
```

```
   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.
```

PROBLEM 1 : (*Repeat Repeat Repeat:* (10 pts))

PART A *(5 pts)*:

Assume an ArrayList named *values* contains the following eight numbers:

```
4 5 22 7 15 31 40 24
```

```
int c = 0;
for (int k=0; k < values.size(); k++)
{
   if (values.get(k) % 5 == 0)
   {
      c = c + values.get(k);
   }
}
```

a. Given the ArrayList *values* above, what is the value of c when the loop ends?

b. Give a meaningful loop invariant for this code.

PART B *(5 pts)*:

Assume an ArrayList named *words* contains the following eight strings:

```
"computer" "science" "ipod" "music" "go" "fun" "dance" "no"
```

For this ArrayList, consider the following code.

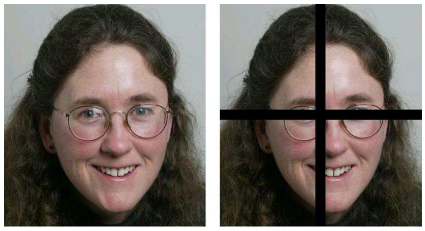
```
int c = 0;
for (int k=0; k < words.size(); k++)
{
    c += words.get(k).length();
}
```

a. What is the value of c after the loop ends?

b. Give a meaningful loop invariant for this code.

PROBLEM 2 : (*Put Prof. Rodger Behind a Window* (10 pts))

Write the `execute` method of `Window` whose header is given below. This method takes a color Pixmap image and modifies the picture to put thick black horizontal and vertical lines about 20 pixels wide in the middle of the picture to look like a window frame.



For example, in the figure below, a color picture is shown on the left and the same image is shown on the right after the Window method has executed. Note that the color Black is when the red, green and blue all have values of zero.

```
public class Window extends Command
{
    // code not shown that is not needed

    public void execute (Pixmap target)
    {
        Dimension bounds = target.getSize();   // size of pixmap
        // TODO: complete method below

    }
}
```

PROBLEM 3 : (*A Zoo full of Animals* (10 pts))

This problem refers to strings of animal types such as `"Bear"`.

Write the method *NumberUniqueTypes* that is given an ArrayList of TreeSets of animal types and returns the unique number of animal types in the ArrayList.

Consider the following ArrayList of sets called *zoos* that contains the following 4 sets.

```
Set 1: "Bear" "Crocodile" "Fox"
Set 2: "Rhino" "Elephant" "Zebra"  "Bear"
Set 3: "Fox" "Elephant" "Zebra"  "Ostrich"
Set 4: "Rhino" "Elephant" "Crocodile"  "Kangaroo"
```

For example, the call *NumberUniqueTypes(zoo)* would return 8 as there are 8 unique animals through the four sets.

```
// Given an ArrayList of Sets of animal types - return the number of
```

```
//     unique animals over all the sets.
public int NumberUniqueTypes (ArrayList <TreeSet <String> > zoolists)
{
```

PROBLEM 4 : (*Where do you live?*(50 pts))

Consider the following problem about housing groups on a college campus, where the information stored about each member of the house is their name and the type of car they have.

PART A: *(8 pts)*

Consider the *Member* class to store information about one member of the housing group, in particular their name and the type of car they drive.

Fill in the missing code for all the methods in the class.

```
public class Member {

 private String myName;      // name of member
 private String myCarType;   // type of car member has

 // constructor
 public Member (String name, String car)
 {




 }

 // return name of member
 public String getName()
 {

 }

 // return type of car member has
 public String getCarType()
 {

 }

 // set the car type to car
```

```
 public void setCarType(String car)
 {

 }
}
```

PART B:

Consider the following abstract HouseGroup class for housing groups on a college campus. Each house group keeps track of its name, its house members and candidates for membership.

```
public abstract class HouseGroup {

   private String myHouseName;        // House group name
   private ArrayList<Member> myMembers;  // members in house
   private ArrayList<Member> myCandidates;  // candidates for membership

   public HouseGroup(String houseName)     // Constructor
   { }  // code not shown

   // returns number of members in house group
   public int numberOfMembers()
   { }  // code not shown

   // add "number" members to group if there are enough candidates
   public abstract void AddMembers(int number);

   // read in current Members info
   public void setupCurrentMembers(Scanner input)
   { }  // code not shown

   // read in current candidates info
   public void setupCurrentCandidates(Scanner input2)
   { }  // code not shown

   // add one member to the house group
   public void addOneMember(Member someone)
   { } // code not shown

   // returns set of all car types for current Members
   public TreeSet<String> allCarTypes()
   {  } // code not shown

   // returns an ArrayList of all possible candidates
   public ArrayList<Member> getCandidates()
```

```
   {  } // code not shown

   // replace list of possible candidates with new list
   public void resetCandidates (ArrayList<Member> possibles)
   {  } // code not shown
}
```

PART B1 ***(16 pts)***:

1. Name the method(s) that must be created by a subclass.
2. Name the method(s) that are accessor methods
3. Name the state variables for the class

4. Fill in the code for the following methods from the HouseGroup class.

```
// Constructor
public HouseGroup(String houseName)
{









}

// returns number of members in house group
public int numberOfMembers()
{









}


// add one member to the house group
public void addOneMember(Member someone)
{









}
```

PART B2 ***(8 pts)***:

Write the HouseGroup class method *setupCurrentMembers* which has a Scanner parameter that is already bound to a file and ready for reading. This method reads in

the car type (one word) and name of a Member (the rest of the line) and adds the member to the ArrayList of members.

For example, assume the Scanner is bound to the following file:

```
Mercury Sarah Hayes
Porsche Mary Anne Johnston
Mustang Alexander Huang
Honda Jessica Li
Honda Daniel Best
Mercury Karna Whitfield
Suzuki Trent Yuen
```

Then these seven students and their car info would be added to the ArrayList.

```
// read in current Members info
public void setupCurrentMembers(Scanner input)
{

}
```

PART B3 ***(8 pts)***:

Write the method allCarTypes, that returns a set containing all the types of cars people own.

For example, if the previous set was the current list of members, then the call allCarTypes() would return the car types: Mercury, Porsche, Mustang, Honda, and Suzuki.

```
// returns set of all car types for current Members
public TreeSet<String> allCarTypes()
{

}
```

PART C ***(10 pts)***:

We would now like to create the class ZZZHouse for the ZZZ housing group. This class extends the class HouseGroup.

```
public class ZZZHouse extends HouseGroup{

// constructor
public ZZZHouse(String name)
{
```

```
    super(name);
}

// Adds "number" members based on its algorithm for picking
//   members, and removes them from the candidate list.
//   If number is bigger then the possible candidates, then
//   just add in all candidates.
public void AddMembers(int number)
{ } // code not shown
```

Write the method AddMembers which has one integer parameter called *number*. AddMembers picks "number" candidates and adds them to its list of housing members. It picks members by first randomly picking from those who have a car different from current members. If all remaining candidates have similar cars, then it randomly picks them. This method must also update both the ArrayList of candidates and the ArrayList of members.

For example, if the list of candidates is the following (with the type of car listed first):

```
Honda Margaret Zhang
Jaguar Henry Roberts
Toyota David Pell
BMW Matthew Tobin
Volvo Eric Campbell
Mercury Sherril Heysham
Volvo Tracey Kitange
```

Suppose number is 3. Then it would randomly pick three members from Kitange, Campbell, Tobin, Roberts and Pell, since they all have cars that none of the current members of the house have. It doesn't matter that Campbell and Kitange have the same type of car, they could both be picked.

Suppose number is 6. Then it would pick all five members Kitange, Campbell, Tobin, Roberts and Pell since they have cars that none of the current members have. Then it would randomly pick one more member choosing between Zhang and Heysham.

In both cases the list of candidates is reset to those who were not picked.

Complete this method on the next page. It has been started for you.

```
public void AddMembers(int number)
{
    // list of possible candidates
    ArrayList<Member> candidates = getCandidates();

    // set of car types of current members
    TreeSet<String> carPicked = allCarTypes();
```

```
    // Place your code here
```

```
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 Pixmap
{
  // create a new Pixmap
  public Pixmap(int width, int height)

  // returns true if (x,y) is in the bounds of the Pixmap
  public boolean isInBounds(int x, int y)

  // return the Dimension of the pixmap, Dimension has a width and height
  public Dimension getSize ()
```

```
  // returns the Color of the pixel at (x,y)
  public Color getColor (int x, int y)

  // sets the Color of the pixel at (x,y)
  public void setColor (int x, int y, Color value)
}

public class Color
{
   // create a Color with each of r, g, b in the range from 0 to 255
   public Color(int r, int g, int b)

   // returns the blue color (0 to 255 value), similar methods for red and green
   public int getBlue()
}

public class Dimension
{
    // returns height of Dimension
    public int getHeight()

    // returns width of Dimension
    public int getWidth()
}

public class ArrayList
{
    // Constructs an empty list
    public ArrayList ()

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

    // Returns element at index in this list.
    public Object get (int index)

    // Replaces the element at the specified position
    // in this list with the specified element.
    public Object set (int index, Object element)

    // Appends specified element to end of this list.
    public boolean add (Object o)
}


public class File
```

```
{
  // Open a new file from the given pathname
  public File (String pathname);
}

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 (or get rest of line if in middle of a line)
    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 ()
}

public class TreeSet
{
   // creates an empty TreeSet
   public TreeSet()

   // adds object e to the TreeSet
   boolean add(Object e)

   // removes all objects from the TreeSet
   void clear()

   // returns true if e is in the set, othewise returns false
   boolean contains(Object e)

  // returns true if set is empty, otherwise returns false
```

```
  boolean isempty()

  // returns an Iterator for the set
  Iterator<Object> iterator()

  // removes the object e from the set
  boolean remove(Object e)

  // returns the number of elements in the set
  int size()

}

public class Random
{
  // constructor
  public Random ()

  // returns random number from 0 (inclusive) to n (exclusive) (does not
  // include n)
  public int nextInt(n)
}
```