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.

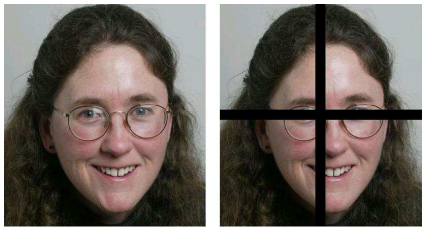
```
int c = 0;
for (int k=0; k < words.size(); k++)
{
    c += words.get(k).length();
}
```

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)
}
```