PROBLEM 1 : (*Loop de loop:* (10 pts))

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

```
5  8  1  4  3  10  2
```

PART A *(5 pts)*:
Assume *item* has been declared an integer and assigned a value.

```
int count = 0;
for (int k=0; k<values.size(); k++)
{
   if (values.get(k) < item)
   {
      count++;
   }
}
```

a. If *item* is 7, what is the value of count when the loop ends?

b. Give a meaningful loop invariant for this code.

PART B *(5 pts)*:
For the same ArrayList above, consider the following code.

```
for (int k=1; k < values.size(); k++)
{
    values.set(k, values.get(k-1) + values.get(k));
}
```

a. List the elements in the ArrayList *values* after the code executes.

b. Give a meaningful loop invariant for this code.

PROBLEM 2 : (*Rotate Prof. Rodger* (10 pts))

Write the `execute` method of `Rotate` whose header is given below. This method takes a color Pixmap image and a number (typed in a dialog box) and rotates the picture that many pixels to the right (wrapping around). NOTE: The variable *x_change* is the rotate amount.

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 Rotate method has been pressed and 150 has been entered.


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

    public void execute (Pixmap target)
    {
        // code not shown that is not needed
        int x_change = scale.width;            // amount to rotate
        Dimension bounds = target.getSize();   // size of pixmap
        Pixmap copy = new Pixmap(target);      // Keep a copy to Rotate

        // TODO: complete method below

    }
}
```

PROBLEM 3 : (*Take Me Out to the Ballgame* (36 pts))

Consider the following abstract Team class for a sports team.

```
public abstract class Team {

   private String myCoach;  // name of coach
   private int myNumGames;  // number of games to play for season
   private int myNumWins;   // number of wins
   private ArrayList<String> myStarters;  // list of players to start

   public Team(String coach, int numGames)  // constructor
   {
      myCoach = coach;
      myNumGames = numGames;
      myNumWins = 0;
      myStarters = new ArrayList<String>();
   }

   // Plays a game and returns true if win, returns false if lose
```

```
   public abstract boolean PlayGame();

   // selects starting player names and puts them in myStarters
   public abstract void CreateStartingLineup();

   // returns name of coach
   public String getCoach()    {  }          // code not shown

   // sets name of coach
   public void setCoach(String coach) { }    // code not shown

   // returns total number of games to play
   public int getNumGames()  { }             // code not shown

   // sets number of games to play
   public void setNumGames(int number) { }   // code not shown

   // adds one more win to win total
   public void AddWin()
   {
      myNumWins++;
   }

   // returns number of wins
   public int getNumWins()   {  }  // code not shown

   // add a player to the starting lineup
   public void AddPlayerStartingLineup(String name)  { } \\ code not shown

   // clear out the starting lineup
    public void ClearStartingLineup()
    {
       myStarters.clear();               // emptys ArrayList
    }
}
```

PART A ***(20 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 Team class.

```
// returns name of coach
public String getCoach()
{


}

// sets name of coach
public void setCoach(String coach)
{


}

// returns total number of games to play
public int getNumGames()
{


}

// sets number of games to play
public void setNumGames(int number)
{


}

// returns number of wins
public int getNumWins()
{


}

// add a player to the starting lineup
public void AddPlayerStartingLineup(String name)
{


}
```

PART B ***(30 pts)***:

We would now like to create the class BaseballTeam, which uses the class BaseballPlayer.

```
public class BaseballPlayer {

    private String myName;        // name of player
    private String myPosition;    // position to play
    private double myBatAverage;  // batting average

    public BaseballPlayer(String name, String position, double average)
    {
       myName = name;
       myPosition = position;
       myBatAverage = average;
    }

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

    // returns position for player to play
    public String getPosition()  { }     // code not shown

    // returns batting average for player
    public double getBattingAverage()  { }  // code not shown
}

public class BaseballTeam extends Team {
   private ArrayList<BaseballPlayer> myPlayers;    // players on the team

   // constructor
   public BaseballTeam(Scanner input) { }  // code not shown

   // play one game and return true if this team wins, false otherwise
   public boolean PlayGame()  { }  // code not shown

   // create a starting lineup
   public void CreateStartingLineup()  { }  // code not shown

   // returns an ArrayList of all the players who play this position
   public ArrayList<String> possiblePlayers(String position) { } // code not shown

   // returns a set of all the possible positions on this team
   public TreeSet<String> uniquePositions()  { }  // code not shown
}
```

PART B.1 ***(8 pts)***:

Complete the constructor for the BaseballTeam class. You should read from a file that has the coaches name (one word), the number of games to play (an integer) and a list of players. Each player has a name (one word), a position to play (one word) and a batting average (double). You do not know how many players are on the team.

A sample file might be the following. Note in this example, Rodger is the coach, 16 is the number of games to play and there are six players on the team.

```
Rodger 16
Todisco pitcher 0.250
Stecher firstbase .450
Stallworth pitcher .185
Hauptman catcher .380
Nicholson firstbase .410
Pavlova firstbase .317
```

Note the parameter *input* is of type Scanner and has already been connected to a file and is ready for reading. You do not need to create a File or Scanner.

```
   public BaseballTeam(Scanner input)
   {
      super("", 0);  // must call constructor of super class first
                     // will need to reset these values after reading from input


}
```

PART B.2 ***(8 pts)***:

Complete the BaseballTeam method `uniquePositions`. This method should return a set of all the unique positions the players on the team can play.

For example, using the previous data file, uniquePositions would return a set with three positions: pitcher, firstbase, and catcher.

Complete `uniquePositions` below.

```
   public TreeSet<String> uniquePositions()
   {

   }
```

PART B.3 ***(8 pts)***:

Complete the BaseballTeam method `possiblePlayers`. This method is given a type of position to play and should return an ArrayList of all the players who can play this position.

For example, using the previous data file, possiblePlayers("pitcher") returns an ArrayList of two names, Todisco and Stallworth.

Complete `possiblePlayers` below.

```
// returns an ArrayList of all the players who play this position
public ArrayList<String> possiblePlayers(String position)
{

}
```

PART B.4 ***(6 pts)***:

Complete the BaseballTeam method `CreateStartingLineup`. This method sets the starting lineup by randomly selecting one player for each type of position. You must figure out where to put the starting lineup, a hint is to look at the Team class again.

For example, using the previous data file, CreateStartingLineup() would set the starting lineup randomly to be one of the two pitchers (maybe Stallworth), Hauptman (he is the only catcher) and one of the three for firstbase (maybe Nicholson).

In writing `CreateStartingLineup`, you should call both `uniquePositions` and `possiblePlayers` that you wrote previously. Assume they are correct, regardless of what you wrote.

Complete `CreateStartingLineup` below.

```
// create a starting lineup
public void CreateStartingLineup()
{

}
```

PROBLEM 4 : (*Who do you know?* (10 pts))

Write the method `friendsInClub` whose header is given below. This method has two parameters, an ArrayList of type TreeSet (each set will be a TreeSet of type String) called *clubs* and a String called *name*. This method returns the unique names of all the people who are in a club with *name*.

For example, assume the sets in the ArrayList *clubs* are the following:

```
Set 1: Coon, Weiss, Pena
Set 2: Pena, Qian, Lawner
Set 3: Weiss, Pena, Lawner
Set 4: Belle, Garavito, Pena, Coon
Set 5: Bierbower, Cladek, Bencan, Pena
```

The call, `friendsInClub(clubs, ‘‘Bencan’’)` returns the set with Cladek, Pena and Bierbower (Note that Bencan is not returned), `friendsInClub(clubs, ‘‘Weiss’’)` returns the set with Coon, Lawner, and Pena, and `friendsInClub(clubs, "Pena")` returns a set with all the names above (since he is in every club) except Pena.

Complete the method `friendsInClub` below.

```
TreeSet<String> friendsInClub(ArrayList<TreeSet<String> > clubs, String name)
{
  TreeSet<String> names = new TreeSet<String>();  // create set to return
  // fill in code here


  return names;
}

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
    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)
}
```