Name _____Period_____

1. Consider the following instance variable and method. Method wordsWithCommas is intended to return a string containing all the words in listOfWords separated by commas and enclosed in braces. For example, if `listOfWords` contains `["one", "two", "three"]`, the string returned by the call `wordsWithCommas()` should be "`{one, two, three}`".

```
private List<String> listOfWords;

public String wordsWithCommas() {
     String result = "{";

     int sizeOfList = listOfWords.size();

     for (int k = 0; k <= sizeOfList; k++)   {
          /* Modify the code */
          if(k != sizeOfList-1)
               result = result + ", ";
     }
     result = result + "}";
     return result;
  }
```

(a) Identify the error in the code above with a circle. Write the corrected code below.

```
int k = 0; k < sizeOfList; k++
```

(b) Write one line of code that could replace `/* Modify the code */` to ensure the method works as intended.

Actually, two lines need to be modified. We also need to assume that listOfWords is initialized somewhere else in the code

```
result = result + listOfWords.get(k);//need to get the value
```

/4

2. Consider the following code segment.

```
/** @param wordList an ArrayList of String objects
 *  @param word the word to be removed
 *  Postcondition:  All occurrences of word have been removed from
 *  wordList.
 */
public static ArrayList<String> removeWord(ArrayList<String> wordList,
String word){
     for(int i = 0; i < wordList.size();i++){
          if(wordList.get(i).equals(word))
               wordList.remove(i);
     }
return wordList;
}
```

(a) Consider a `wordList` which contains the following values,

`{bat  dog  cat  sat  rat  rat  rat}`

What is returned when the following method is called?

`removeWord(wordList, "rat")`

bat
dog
cat
sat
rat

an instance of rat is still in the list. How could you modify the code above to fix this?

/1

3. A two-dimensional array of integers in which most elements are zero is called a sparse array. Because most elements have a value of zero, memory can be saved by storing only the non-zero values along with their row and column indexes. The following complete `SparseArrayEntry` class is used to represent non-zero elements in a sparse array. A `SparseArrayEntry` object cannot be modified after it has been constructed.

```
public class SparseArrayEntry {
/** The row index and column index for this entry in the sparse array */
     private int row;
     private int col;

/** The value of this entry in the sparse array */
     private int value;

/** Constructs a SparseArrayEntry object that represents a sparse
*   array element with row index r and column index c, containing
*   value v.
*/
public SparseArrayEntry(int r, int c, int v)   {
     row = r;
     col = c;
     value = v;   }

/** Returns the row index of this sparse array element. */
     public int getRow()   {
          return row;
     }

/** Returns the column index of this sparse array element. */
     public int getCol()   {
          return col;
     }

/** Returns the value of this sparse array element. */
     public int getValue()   {
          return value;
      }
}
```

The `SparseArray` class represents a sparse array. It contains a list of `SparseArrayEntry` objects, each of which represents one of the non-zero elements in the array. The entries representing the non-zero elements are stored in the list in no particular order. Each non-zero element is represented by exactly one entry in the list

```
public class SparseArray {
/** The number of rows and columns in the sparse array. */
     private int numRows;
     private int numCols;
/** The list of entries representing the non-zero elements of the
 *  sparse array. Entries are stored in the list in no particular order.
 *  Each non-zero element is represented by exactly one entry in
 *  the list.
*/
     private ArrayList<SparseArrayEntry> entries;
/** Constructs an empty SparseArray. */
     public SparseArray()   {
          entries = new ArrayList<SparseArrayEntry>();
     }

/** Returns the number of rows in the sparse array. */
     public int getNumRows()   {
          return numRows;
     }

/** Returns the number of columns in the sparse array. */
     public int getNumCols()   {
          return numCols;
     }

/** Returns the value of the element at row index row and column
 *  index col in the sparse array.
 *  Precondition:  0 ≤ row < getNumRows()
 *  0 ≤ col < getNumCols()
*/
     public int getValueAt(int row, int col)   {
          /* implementation not shown */
     }

/** Returns the col in which the first instance of an element is found
 *  Precondition:  0 ≤ col < getNumCols()
 *  0 ≤ col < getNumCols()
*/
     public int findCol(int value)   {
          /* to be implemented in part (a) */
     }
/** Removes the row from the sparse array.
 *  Precondition:  0 ≤ row < getNumRows()
 */
     public void removeRow(int row)   {
          /* to be implemented in part (b) */
     }
// There may be instance variables, constructors, and methods that are not
shown.
}
```

(a) The following table shows an example of a two-dimensional sparse array. Empty cells in the table indicate zero values

	0	1	2	3	4
0					
1		5			4
2	1				
3		-9			
4					
5					

The sample array can be represented by a `SparseArray` object, `sparse`, with the following instance variable values. The items in entries are in no particular order; one possible ordering is shown below.

```
numRows: 6

numCols: 5

entries: [ row: 1 col: 4 value: 4 ] [ row: 2 col: 0 value: 1 ] [ row: 3 col: 1 value: -9 ] [ row: 1 col: 1 value: 5 ]
```

Write the `SparseArray` method `findCol`. The method returns the col in which the first instance of the value is found. If there are no cols that contain the specified value, 0 is returned.

In the example above, the call `sparse.findCol(-9)` would return 1, and `sparse.findCol(3)` would return 0.

Complete method `findCol` below.

```
/** Returns the col in which the first instance of an element is found
 *  Precondition:  0 ≤ col < getNumCols()
 *                 0 ≤ row < getNumRows()
 */
public int findCol(int value)
```

```
    public int findCol(int value)   {
        for(int e = 0; e < entries.size();e++){
            if(entries.get(e).getValue() == value){
                return entries.get(e).getCol();
            }
        }
    return 0;
    }
```

/3

(b) Write the `SparseArray` method `removeRow`. After removing a specified row from a sparse array:

- All entries in the list entries with row indexes matching row are removed from the list.
- All entries in the list entries with row indexes greater than row are replaced by entries with row indexes that are decremented by one (moved one row to the up).
- The number of rows in the sparse array is adjusted to reflect the row removed.

The sample object sparse from the beginning of the question is repeated for your convenience.

	0	1	2	3	4
0					
1			5		4
2	1				
3			-9		
4					
5					

The outlined entries below correspond to the shaded column above.

entries	row: 2 col: 0 value: 1	row: 1 col: 2 value 5	row: 1 col: 4 value: 4	row: 3 col: 2 value: -9

When sparse has the state shown above, the call `sparse.removeRow(1)` could result in sparse having the following values in its instance variables (since entries is in no particular order, it would be equally valid to reverse the order of its two items). The shaded areas below show the changes.

```
numRows: 5
numCols: 5

entries:
```

row: 1 col: 0 value: 1	row: 2 col: 2 value: -9

```
numRows: 4
numCols: 5
```

Class information repeated from the beginning of the question

```
public class SparseArrayEntry

public SparseArrayEntry(int r, int c, int v)
public int getRow()
public int getCol()
public int getValue()
```

```
public class SparseArray

private int numRows
private int numCols
private List<SparseArrayEntry> entries
public int getNumRows()
public int getNumCols()
public int getValueAt(int row, int col)
public void removeRow(int col)
public int findCol(int value)

Complete method removeRow below.

/** Removes the row from the sparse array.
 *  Precondition:  0 ≤ row < getNumRow()
 */

public void removeRow(int row)

         public void removeRow(int row)   {
              //First lets remove the entries that correspond to the current
row
              for(int e = 0; e < entries.size(); e++){
                if(entries.get(e).getRow() == row){
                      entries.remove(e);
                      e--;
                }
              }

              //now we will update the entries with a row greater than the
current row
              for(int e = 0; e < entries.size(); e++){
                if(entries.get(e).getRow() > row){
                    entries.set(e, new
SparseArrayEntry(entries.get(e).getRow()-1, entries.get(e).getCol(),
entries.get(e).getValue()));
                }
              }


         }
```

/5

4. In this question, you will complete methods in classes that can be used to represent a multi-player game. You will be able to implement these methods without knowing the specific game or the players' strategies.

   The GameState interface describes the current state of the game. Different implementations of the interface can be used to play different games. For example, the state of a checkers game would include the positions of all the pieces on the board and which player should make the next move.

   The GameState interface specifies these methods. The Player class will be described in part (a).

```
public interface GameState
{
  /** @return true if the game is in an ending state;
   *          false otherwise
   */
  boolean isGameOver();

  /** Precondition: isGameOver() returns true
   *  @return the player that won the game or null if there was no winner
   */
  Player getWinner();

  /** Precondition: isGameOver() returns false
   *  @return the player who is to make the next move
   */
  Player getCurrentPlayer();

  /** @return a list of valid moves for the current player;
   *          the size of the returned list is 0 if there are no valid moves.
   */
  ArrayList<String> getCurrentMoves();

  /** Updates game state to reflect the effect of the specified move.
   *  @param move a description of the move to be made
   */
  void makeMove(String move);

  /** @return a string representing the current GameState
   */
  String toString();

}
```

The makeMove method makes the move specified, updating the state of the game being played. Its parameter is a String that describes the move. The format of the string depends on the game. In tic-tac-toe, for example, the move might be something like "X-1-1", indicating an X is put in the position (1, 1).

The Player class provides a method for selecting the next move. By extending this class, different playing strategies can be modeled.

```
public class Player
{
   private String name;    // name of this player

   public Player(String aName)
   {  name = aName;  }

   public String getName()
   {  return name;  }

   /** This implementation chooses the first valid move.
    *  Override this method in subclasses to define players with other strategies.
    *  @param state the current state of the game; its current player is this player.
    *  @return a string representing the move chosen;
    *          "no move" if no valid moves for the current player.
    */
   public String getNextMove(GameState state)
   {  /* implementation not shown */  }
}
```

(a) The method getNextMove returns the next move to be made as a string, using the same format as that used by makeMove in GameState. Depending on how the getNextMove method is implemented, a player can exhibit different game-playing strategies.

Write the complete class declaration for a RandomPlayer class that is a subclass of Player. The class should have a constructor whose String parameter is the player's name. It should override the getNextMove method to randomly select one of the valid moves in the given game state. If there are no valid moves available for the player, the string "no move" should be returned.

```
public class RandomPlayer extends Player{

     public RandomPlayer(String n){
         Super(n);
     }

     public String getNextMove(GameState state){
            ArrayList<String> temp = state.getCurrentMoves();
            if(temp.size() == 0){
                   return “no move”;
            }
            return temp.get((int)(Math.random()* temp.size()))
     }
}
```

(b) The GameDriver class is used to manage the state of the game during game play. The GameDriver class can be written without knowing details about the game being played

```
public class GameDriver
{
  private GameState state;   // the current state of the game

  public GameDriver(GameState initial)
  {  state = initial;  }


  /** Plays an entire game, as described in the problem description
   */
  public void play()
  {  /* to be implemented in part (b) */ }

  // There may be fields, constructors, and methods that are not shown.
}
```

Write the GameDriver method play. This method should first print the initial state of the game. It should then repeatedly determine the current player and that player's next move, print both the player's name and the chosen move, and make the move. When the game is over, it should stop making moves and print either the name of the winner and the word "wins" or the message "Game ends in a draw" if there is no winner. You may assume that the GameState makeMove method has been implemented so that it will properly handle any move description returned by the Player getNextMove method, including the string "no move".

Complete method play below

```
/** Plays an entire game, as described in the problem description
 */
public void play()
```

```
public void play(){
     System.out.println("initial state: " + state);

      while(!state.isGameOver()){
            Player currPlayer = state.getCurrentPlayer();
            String currMove = currPlayer.getNextMove(state);
            System.out.println(currPlayer.getName + ": " + currMove);
            state.makeMove(currMove);
       }

       Player winner = state.getWinner();
       if(winner != null){
             System.out.println(winner.getName() + " wins");
       }else{
             System.out.println("Game ends in a draw");
       }
 }
```

5. Refer to the code below,

```
//Assume nextLine() and nextInt() are static methods in
//a class named Scanner that reads a String and an integer
//from the keyboard.

Scanner rdr = new Scanner(System.in);
String str = rdr.nextLine();
int j = rdr.nextInt();

try  {
      System.out.print( str.charAt(j) );

} catch(StringIndexOutOfBoundsException e) {

      System.out.print("Error: " + j);
}
```

(c) What is the output of the code above, given the input below?

```
big mama
2
g
```

/1

(d) What is the output of the code above, given the input below?

```
big mama
22
Error: 22
```

/1

6. For each of the following unchecked errors, write a try-catch block to catch the error.

```
int[] myArray = {1, 2, 3}
System.out.println(myArray[3]);
```

```
try{
     int[] myArray = {1, 2, 3}
     System.out.println(myArray[3]);
}catch(ArrayIndexOutOfBoundsException e){
     System.out.println(e);
}
```

```
System.out.println(10/0);
```

```
try{
     System.out.println(10/0);
}catch(Arithmetic e){
     System.out.println(e);
}
```

```
String pointer = null;


try{
     if(pointer.equals("this");
              //do something
}catch(NullPointerException e){
     System.out.println(e);
}
```

```
Object x = new Integer(0);
System.out.println((String)x);


try{
     System.out.println((String)x);
}catch(ClassCastException e){
     System.out.println(e);
}
```

```
String s = "Hello";
System.out.println(s.charAt(5));


try{
     System.out.println(s.charAt(5));
}catch(IndexOutOfBoundsException e){
     System.out.println(e);
}
```

/10