1.Consider the following incomplete `StringUtil` class declaration. You will write implementations for the two methods listed in this class. Information about the `Person` class used in the `replaceNameNickname` method will be presented in part (b).

```
public class StringUtil
{   private String str;
  /**
   *  @param oldstr a String
   *  @param newstr a String
   *  @return a new String in which all occurrences of the substring
   *              oldstr in str are replaced by the substring newstr
   */
  public  String apcsReplaceAll(String oldStr,String newStr)

  {  /* to be implemented in part (a) */      }

  /**
   *  @param    people array of references to Person objects
   *  @return     a copy of str modified so that each occurrence of a first
   *             name in people is replaced by the corresponding nickname
   */
  public String replaceNameNickname(Person[]people)

  {  /* to be implemented in part (b) */      }

  // There may be methods that are not shown.
}
```

(a) Write the `StringUtil` method `apcsReplaceAll`, which examines a given `String` and replaces all occurrences of a designated substring with another specified substring. In writing your solution, you may NOT use the `replace`, `replaceAll`, or `replaceFirst` methods in the Java `String` class.

The following table shows several examples of the result of calling `StringUtil.apcsReplaceAll( oldstr,newstr)`.

str	oldstr	newstr	String returned	Comment
`"to be or not to be"`	`"to"`	`"2"`	`"2 be or not 2 be"`	Each occurrence of `"to"` in the original string has been replaced by "2"
`"advanced calculus"`	`"math"`	`"science"`	`"advanced calculus"`	No change, because the string `"math"` was not in the original string
`"gogogo"`	`"go"`	`"gone"`	`"gonegonegone"`	Each occurrence of `"go"` in the original string has been replaced by `"gone"`
`"aaaaa"`	`"aaa"`	`"b"`	`"baa"`	The first occurrence of `"aaa"` in the original string has been replaced by `"b"`

Complete method `apcsReplaceAll` below.

```
/**
 *  @param oldstr a String
 *  @param newstr a String
 *  @ return a new String in which all occurrences of the substring
 *            oldstr in str are replaced by the substring newstr
 */
public  String      apcsReplaceAll(
                                String oldStr,
                                String newStr)
```

(b) The following `Person` class contains information that includes a first (given) name and a nickname for the person.

```
public class Person
{
  /** @return the first name of this Person */
  public String getFirstName()
  { /* implementation not shown */ }

  /** @return the nickname of this Person */
  public String getNickname()
  { /* implementation not shown */ }

  // There may be instance variables, constructors, and methods not shown.
}
```

Write the `StringUtil` method `replaceNameNickname`, which a array of `Person` objects that contain first names and a corresponding nicknames. The method is to replace all names by their nicknames in the given string. The array of `Person` objects is processed in order from lowest index to highest index. In writing your solution, you may NOT use the `replace`, `replaceAll`, or `replaceFirst` methods in the Java `String` class.
After the replacement, make sure the `String str` doesn't change

For example, assume the following table represents the data contained in the list `people`.

	`getFirstName()`	`getNickname()`
0	`"Henry"`	`"Hank"`
1	`"Elizabeth"`	`"Liz"`
2	`"John"`	`"Jack"`
3	`"Margaret"`	`"Peggy"`

Assume also that `String str` represents the following string.

```
"After Henry drove Elizabeth to dinner in Johnson City, Henry
paid for an appetizer and Elizabeth paid for dessert."
```

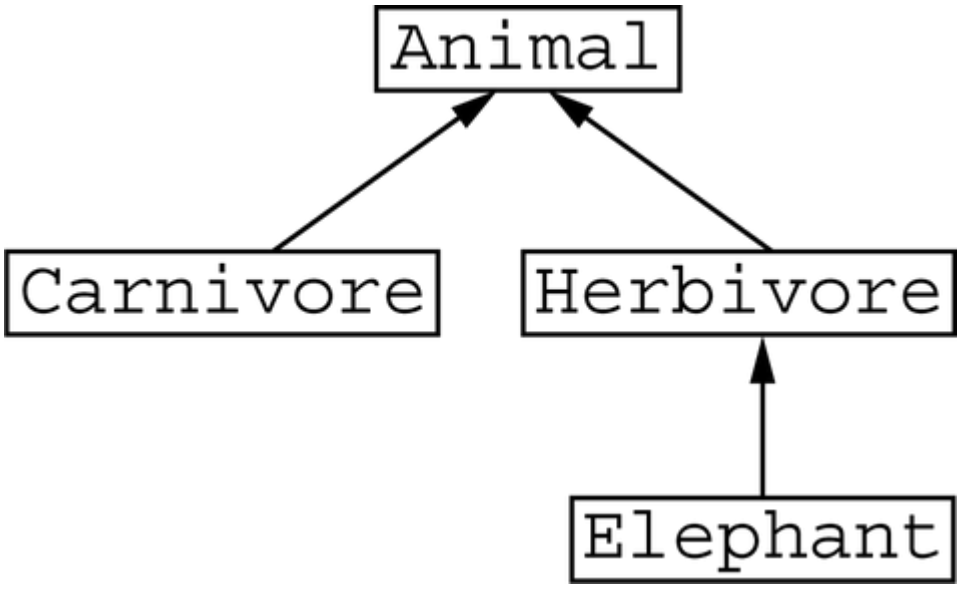
The call `StringUtil.replaceNameNickname( people)` should return the following string and `String str` doesn't change:

```
"After Hank drove Liz to dinner in Jackson City, Hank paid for
an appetizer and Liz paid for dessert."
```

In writing your solution, you must use the method `apcsReplaceAll` specified in the `StringUtil` class. Assume that `apcsReplaceAll` works as specified, regardless of what you wrote in part (a).

Complete method `replaceNameNickname` below.

```
/** @param str a String
  *  @param people  array of references to  Person  objects
  *  @return  a copy of  str  modified so that each occurrence of a first
  *                      people  is replaced by the corresponding nickname
  *            name in   and str doesn't change
  */
public String replaceNameNickname(Person[] people)
```



2.All `Animal` objects have the following attributes.

A `String` variable indicating whether the animal is a carnivore or a herbivore
A `String` variable representing the animal species (e.g., lion, giraffe, zebra)
A `String` variable representing the name of the individual animal (e.g., Lisa, Gary, Percy)

The `Animal` class also contains a `toString` method that indicates the state of an animal.

The following table shows the intended behavior of the `Animal` class.

Statement	Result
`Animal lisa = new Animal("carnivore", "lion", "Lisa");`	A new `Animal` object is created.
`lisa.toString();`	The string "`Lisa the lion is a carnivore`" is returned.

(a) Write the complete `Animal` class. Your implementation must meet all specifications and conform to the behavior shown in the table.

The `Herbivore` class is a subclass of `Animal`. The `Herbivore` class does not contain any attributes or methods other than those found in the `Animal` class.

The constructor to the `Herbivore` class accepts two parameters for the species and name of the herbivore. The constructor passes those parameters along with the string literal "`herbivore`" to the `Animal` class to construct the object.

The following table shows the intended behavior of the `Herbivore` class.

Statement	Result
`Herbivore gary = new Herbivore("giraffe", "Gary");`	A new `Herbivore` object is created.
`gary.toString();`	The string `"Gary the giraffe is a herbivore"` is returned.

(b) Write the complete `Herbivore` class. Your implementation must meet all specifications and conform to the behavior shown in the table.

The `Elephant` class is a subclass of `Herbivore`. The `Elephant` class contains one additional attribute not found in `Herbivore`: a `double` variable representing the length of the elephant's tusks, in meters.

The constructor to the `Elephant` class accepts two parameters for the name and tusk length of the elephant. The constructor passes those parameters along with the string literal `"elephant"` to the `Herbivore` class to construct the object.

The following table shows the intended behavior of the `Elephant` class.

Statement	Result
`Elephant percy = new Elephant("Percy", 2.0);`	A new `Elephant` object is created.
`percy.toString();`	The string `"Percy the elephant is a herbivore with tusks 2.0 meters long"` is returned.

(c) Write the complete `Elephant` class. Your implementation must meet all specifications and conform to the behavior shown in the table.

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 i  increasing  rder. Each non-zero element is represented by exactly one entry in the list.
   */
  private List<SparseArrayEntry> entries;

  /** Constructs an empty SparseArray. */
  publicSparseArray(            int [][] arr2)

  {  /* to be implemented in part (b) */  }

  /** 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)
  {  /* to be implemented in part (a)  */  }

  /** Removes the column col from the sparse array.
   *  Precondition: 0   col < getNumCols()
   */
  public void removeColumn(int col)
  {  }

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

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

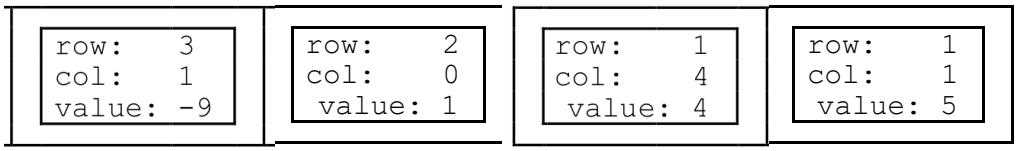
	0	1	2	3	4
0					
1		5			4
2	1				
3		-9			
4					
5					

The sample array can be represented by a `SparseArray` object, `sparse`, with the following instance variable values. The items in `entries` are in increasing order;

`numRows: 6`

`numCols: 5`

`entries:`



(a) Write the `SparseArray` method `getValueAt`. The method returns the value of the sparse array element at a given row and column in the sparse array. If the list `entries` contains an entry with the specified row and column, the value associated with the entry is returned. If there is no entry in `entries` corresponding to the specified row and column, 0 is returned.

Complete method `getValueAt` below.

```
/** 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)
```

(b) Write the `SparseArray` constructor method. Make sure the list in increasing order by values

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

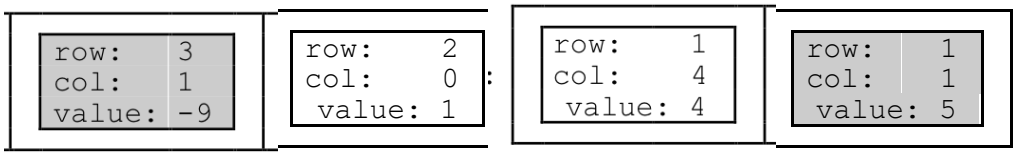
	0	1	2	3	4
0	0	0	0	0	0
1	0	5	0	0	4
2	1	0	0	0	0
3	0	-9	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

entries in `entries`, formed in increasing order by values.

`numRows: 6`

`numCols: 5`

`entries:`



```
public SparseArray(                int[][] arr2)
```

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
SparseArray(          int[][] arr2)
public int getNumRows()
public int getNumCols()
public int getValueAt(int row, int
col)
```

4. This question involves manipulation of one-dimensional and two-dimensional arrays. In part (a), you will write a method to shift the elements of a one-dimensional array. In parts (b) you will write methods to shift the elements of a two-dimensional array.

(a) Consider the following incomplete `ArrayUtil` class, which contains a `static shiftArray` method.

```
public class ArrayUtil
{
  /** Shifts each array element to the next higher index, discarding the
   *  original last element, and inserts the new number at the front.
   *  @param arr the array to manipulate
   *             Precondition: arr.length > 0
   *  @param num the new number to insert at the front of arr
   *  Postcondition: The original elements of arr have been shifted to
   *                 the next higher index, and arr[0] == num.
   *                 The original element at the highest index has been
   *                 discarded.
   */
  public static void shiftArray(int[] arr, int num)
  {  /* to be implemented in part (a) */  }

  // There may be methods that are not shown.
}
```

Write the `ArrayUtil` method `shiftArray`. This method stores the integer `num` at the front of the array `arr` after shifting each of the original elements to the position with the next higher index. The element originally at the highest index is lost.

For example, if `arr` is the array `{11, 12, 13, 14, 15}` and `num` is 27, the call to `shiftArray` changes `arr` as shown below.

Before call arr:	0	1	2	3	4
	11	12	13	14	15

After call arr:	0	1	2	3	4
	27	11	12	13	14

Complete method `shiftArray` below.

```
/** Shifts each array element to the next higher index, discarding the
 *  original last element, and inserts the new number at the front.
 *  @param arr the array to manipulate
 *             Precondition: arr.length > 0
 *  @Param num the new number to insert at the front of arr
 *  Postcondition: The original elements of arr have been shifted to
 *                 the next higher index, and arr[0] == num.
 *                 The original element at the highest index has been
 *                 discarded.
 */
public static void shiftArray(int[] arr, int num)
```

(b) Consider the following incomplete `NumberMatrix` class, which represents a two-dimensional matrix of integers. Assume that the matrix contains at least one integer.

```
public class NumberMatrix
{
  private int[][] matrix;

  /** Constructs a number matrix. */
  public NumberMatrix(int[][] m)
  {  matrix = m;  }

  /** Shifts each matrix element to the next position in row-major order
   *  and inserts the new number at the front. The last element in the last
   *  row is discarded.
   *  @param num the new number to insert at the front of matrix
   *  Postcondition: The original elements of matrix have been shifted to
   *                 the next higher position in column order, and
   *                 matrix[0][0] == num.
   *                 The original last element in the last row is discarded.
   */
  public void shiftMatrix(int num)
  {  /* to be implemented in part (b) */  }

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

Write the `NumberMatrix` method `shiftMatrix`. This method stores a new value `num` into the two-dimensional array `matrix` after shifting the elements to the next higher position in column-major order. The element originally at the last position in column order is lost.

For example, if `m1` is a reference to a `NumberMatrix` object, then the call `m1.shiftMatrix(48)` will change the values in `matrix` as shown below.

Before call

	0	1	2
0	13	17	21
1	14	18	22
2	15	19	23
3	16	20	24

After call

	0	1	2
0	48	16	20
1	13	17	21
2	14	18	22
3	15	19	23

In writing `shiftMatrix`, you must call the `shiftArray` method in part (a). Assume that `shiftArray` works correctly regardless of what you wrote in part (a).

Complete method `shiftMatrix` below.

```
/** Shifts each matrix element to the next position in column -major order
 *  and inserts the new number at the front. The last element in the last
 *  row is discarded.
 *  @param num the new number to insert at the front of matrix
 *  Postcondition: The original elements of matrix have been shifted
 *                 to the next higher position in column -major order, and
 *                 matrix[0][0] == num.
 *                 The original last element in the last row is discarded.
 */
public void shiftMatrix(int num)
```