Tarea

Tomando como base el código visto en clase de Lista Simples, realice las modificaciones necesarias en dicho código para tener una Lista Doblemente Enlazada.

La tarea debe entregarse en formato pdf conteniendo el código de las clases Nodo y Lista remarcando los cambios en el código.

Código de la clase Nodo

```
// clase para representar un nodo en una lista
class NodoLista
{
   // miembros de acceso del paquete; Lista puede acceder a ellos directamente
   Object datos; // los datos para este nodo
   NodoLista siguienteNodo; // referencia al siguiente nodo en la lista

   // el constructor crea un objeto NodoLista que hace referencia al objeto
   NodoLista( Object objeto )
   {
      this( objeto, null );
   } // fin del constructor de NodoLista con un argumento

   // el constructor crea un objeto NodoLista que hace referencia a
   // un objeto Object y al siguiente objeto NodoLista
   NodoLista( Object objeto, NodoLista nodo )
   {
      datos = objeto;
      siguienteNodo = nodo;
   } // fin del constructor de NodoLista con dos argumentos

   // devuelve la referencia a datos en el nodo
   Object obtenerObject()
   {
      return datos; // devuelve el objeto Object en este nodo
   } // fin del método obtenerObject

   // devuelve la referencia al siguiente nodo en la lista
   NodoLista obtenerSiguiente()
   {
      return siguienteNodo; // obtiene el siguiente nodo
   } // fin del método obtenerSiguiente
} // fin de la clase NodoLista
```

Código de la clase Lista

```
// definición de la clase Lista
public class Lista
{
   private NodoLista primerNodo;
   private NodoLista ultimoNodo;
   private String nombre; // cadena como "lista", utilizada para imprimir

   // el constructor crea una Lista vacía con el nombre "lista"
   public Lista()
   {
      this( "lista" );
   } // fin del constructor de Lista sin argumentos

   // el constructor crea una Lista vacía con un nombre
   public Lista( String nombreLista )
   {
      nombre = nombreLista;
      primerNodo = ultimoNodo = null;
   } // fin del constructor de Lista con un argumento

   // inserta objeto Object al frente de la Lista
   public void insertarAlFrente( Object elementoInsertar )
   {
      if ( estaVacia() ) // primerNodo y ultimoNodo hacen referencia al mismo objeto
         primerNodo = ultimoNodo = new NodoLista( elementoInsertar );
      else // primerNodo hace referencia al nuevo nodo
         primerNodo = new NodoLista( elementoInsertar, primerNodo );
   } // fin del método insertarAlFrente

   // inserta objeto Object al final del la Lista
   public void insertarAlFinal( Object elementoInsertar )
   {
      if ( estaVacia() ) // primerNodo y ultimoNodo hacen referencia al mismo objeto
         primerNodo = ultimoNodo = new NodoLista( elementoInsertar );
      else // el siguienteNodo de ultimoNodo hace referencia al nuevo nodo
         ultimoNodo = ultimoNodo.siguienteNodo = new NodoLista( elementoInsertar );
   } // fin del método insertarAlFinal
```

```
// elimina el primer nodo de la Lista
public Object eliminarDelFrente() throws ExcepcionListaVacia
{
   if ( estaVacia() ) // lanza excepción si la Lista está vacía
      throw new ExcepcionListaVacia( nombre );

   Object elementoEliminado = primerNodo.datos; // obtiene los datos que se van a
                                                eliminar

   // actualiza las referencias primerNodo y ultimoNodo
   if ( primerNodo == ultimoNodo )
      primerNodo = ultimoNodo = null;
   else
      primerNodo = primerNodo.siguienteNodo;

   return elementoEliminado; // devuelve los datos del nodo eliminado
} // fin del método eliminarDelFrente

// elimina el último nodo de la Lista
public Object eliminarDelFinal() throws ExcepcionListaVacia
{
   if ( estaVacia() ) // lanza excepción si la Lista está vacía
      throw new ExcepcionListaVacia( nombre );

   Object elementoEliminado = ultimoNodo.datos; // obtiene los datos que se van a
                                                eliminar

   // actualiza las referencias primerNodo y ultimoNodo
   if ( primerNodo == ultimoNodo )
      primerNodo = ultimoNodo = null;
   else // localiza el nuevo último nodo
   {
      NodoLista actual = primerNodo;

      // itera mientras el nodo actual no haga referencia a ultimoNodo
      while ( actual.siguienteNodo != ultimoNodo )
         actual = actual.siguienteNodo;
```

```
         ultimoNodo = actual; // actual el nuevo ultimoNodo
         actual.siguienteNodo = null;
      } // fin de else

      return elementoEliminado; // devuelve los datos del nodo eliminado
   } // fin del método eliminarDelFinal

   // determina si la lista está vacía
   public boolean estaVacia()
   {
      return primerNodo == null; // devuelve true si la lista está vacía
   } // fin del método estaVacia

   // imprime el contenido de la lista
   public void imprimir()
   {
      if ( estaVacia() )
      {
         System.out.printf( "%s vacia\n", nombre );
         return;
      } // fin de if

      System.out.printf( "La %s es: ", nombre );
      NodoLista actual = primerNodo;

      // mientras no esté al final de la lista, imprime los datos del nodo actual
      while ( actual != null )
      {
         System.out.printf( "%s ", actual.datos );
         actual = actual.siguienteNodo;
      } // fin de while

      System.out.println( "\n" );
   } // fin del método imprimir
} // fin de la clase Lista
```

Código de la clase ExcepcionListaVacia

```
public class ExcepcionListaVacia extends RuntimeException
{
   // constructor sin argumentos
   public ExcepcionListaVacia()
   {
      this( "Lista" ); // llama al otro constructor de ExcepcionListaVacia
   } // fin del constructor de ExcepcionListaVacia sin argumentos

   // constructor con un argumento
   public ExcepcionListaVacia( String nombre )
   {
      super( nombre + " esta vacia" ); // llama al constructor de la superclase
   } // fin del constructor de ExcepcionListaVacia con un argumento
} // fin de la clase ExcepcionListaVacia
```