Práctica 2: Encapsulamiento y Abstracción

Programación III - UNLP

Cursada 2013

Todos los ejercicios de la presente práctica deberán hacerse en `Java` dentro de un mismo proyecto llamado "`Prog3_2013`". Se recomienda realizar cada ejercicio en un paquete diferente, y se recomienda el uso de `Eclipse`.

Además, se recomienda realizar todas las practicas de la presente materia dentro de un mismo proyecto de `Eclipse`, a fin de poder reutilizar código escrito entre diferentes ejercicios y prácticas.

1. Considere la siguiente especificación de operaciones de una lista de enteros:

```
1  public abstract class ListaDeEnteros {
2          // Se prepara para iterar los elementos de la lista.
3          public abstract void comenzar ();
4
5          // Avanza al próximo elemento de la lista.
6          public abstract void proximo ();
7
8          // Determina si llegó o no al final de la lista.
9          public abstract boolean fin();
10
11         // Retorna el elemento actual.
12         public abstract Integer elemento ();
13
14         // Retorna el elemento de la posición indicada (la posición
               va desde 0 hasta n-1).
15         public abstract Integer elemento (int pos);
16
17         // Agrega el elemento en la posición actual y retorna true
               si pudo agregar y false si no pudo agregar.
18         public abstract boolean agregar (Integer elem);
19
20         // Agrega el elemento en la posición indicada y retorna true
                si pudo agregar y false; si no pudo agregar.
21         public abstract boolean agregar (Integer elem, int pos);
22
23         // Elimina el elemento actual y retorna true si pudo
               eliminar y false en caso contrario.
24         public abstract boolean eliminar ();
25
26         // Elimina el elemento de la posición indicada y retorna
               true si pudo eliminar y false en caso contrario.
27         public abstract boolean eliminar (int pos);
```

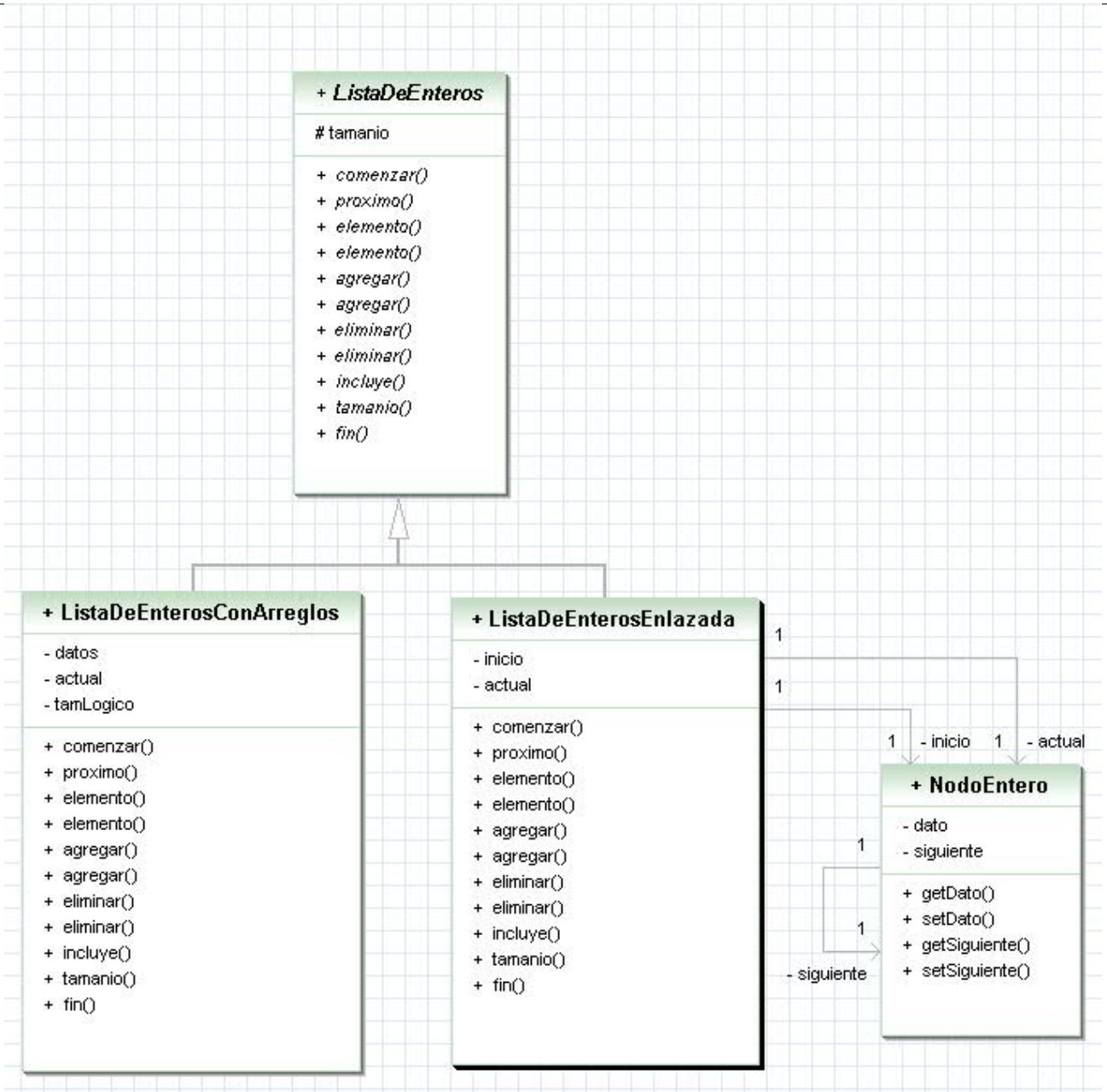

Figura 1: Lista de Enteros

```
28
29          // Retorna true si la lista está vacía, false en caso
                contrario.
30          public abstract boolean esVacia();
31
32          // Retorna true si elem está contenido en la lista, false en
                caso contrario.
33          public abstract boolean incluye (Integer elem);
34
35          // Retorna la longitud de la lista.
36          public abstract int tamanio ();
37  }
```

a) Implemente en JAVA (pase por máquina) una clase abstracta llamada `ListaDeEnteros` de acuerdo a la especificación dada ubicada dentro del paquete `practica2.ejercicio1`.

b) Escriba una clase llamada `ListaDeEnterosConArreglos` como subclase de `ListaDeEnteros` dentro del paquete anterior, de manera que implemente todos los métodos definidos en la superclase haciendo uso de arreglos de longitud fija.

c) Escriba una clase llamada `TestListaDeEnterosConArreglos` que reciba en su método main una secuencia de números, los agregue a un objeto de tipo `ListaDeEnterosConArreglos` y luego imprima los elementos de dicha lista.

d) Escriba una clase llamada `ListaDeEnterosEnlazada` como otra subclase de `ListaDeEnteros` dentro del paquete `practica2.ejercicio1`, de manera que implemente todos los métodos definidos en la superclase pero haciendo uso de una estructura recursiva[1]. En este caso la lista tendrá una referencia al elemento inicial y al actual. Luego, se definirá otra clase recursiva que contenga el elemento y que referencie a una instancia similar a sí misma.

e) Escriba una clase llamada `TestListaDeEnterosEnlazada` que reciba en su método main una secuencia de números, los agregue a un objeto de tipo `ListaDeEnterosEnlazada` y luego imprima los elementos de dicha lista.

f) ¿Es posible implementar el método incluye en la clase `ListaDeEnteros` de manera que solo utilice otras operaciones de lista y no dependa de la estructura interna elegida en las subclases? Si fuera posible, ¿funcionaria el envío de este mensaje a un objeto de tipo `ListaDeEnterosConArreglos`?

2. Sea la siguiente especificación de `PilaDeEnteros`:

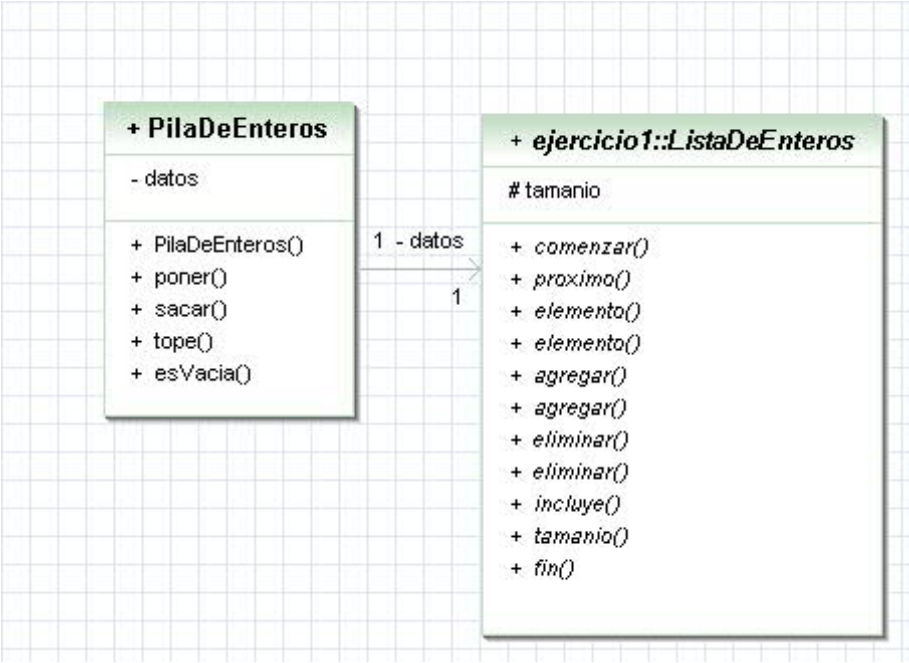

Figura 2: Pila de Enteros

```
1 public class PilaDeEnteros {
2         private ListaDeEnteros datos;
3
```

[1]Una lista enlazada con punteros o referencias entre nodos es una estructura recursiva.

```
4         public PilaDeEnteros() { /* ... */ }
5
6         // Agrega elem a la pila.
7         public void poner(Integer elem) { /* ... */ }
8
9         //  Elimina y devuelve el elemento en el tope de la pila.
10        public int sacar() { /* ... */ }
11
12        //  Devuelve el elemento en el tope de la pila sin
              eliminarlo.
13        public tope tope() { /* ... */ }
14
15        // Retorna true si la pila está vacía, false en caso
              contrario.
16        public boolean esVacia() { /* ... */ }
17 }
```

a) Implemente en JAVA (pase por máquina) la clase PilaDeEnteros (utilizando alguna subclase de ListaDeEnteros) de acuerdo a la especificación dada ubicada dentro del paquete `practica2.ejercicio2`.

b) Escriba una clase llamada `TestPilaDeEnteros` para ejecutar el siguiente código:

```
1  public class TestPilaDeEnteros {
2          public static void main(String[] args) {
3                  PilaDeEnteros p1, p2;
4                  Integer valor2;
5                  p1 = new PilaDeEnteros();
6                  p1.poner(1);
7                  p1.poner(2);
8                  p2 = p1;
9                  valor2 = p2.sacar();
10                 System.out.println("El valor del tope de la pila
                       p1 es: " + p1.sacar());
11         }
12 }
```

c) ¿Qué valor imprime? ¿Qué conclusión saca?

3. Sea la siguiente especificación de una `ColaDeEnteros`:

```
1  public class ColaDeEnteros {
2         private ListaDeEnteros datos = ...;
3
4         public ColaDeEnteros() { /* ... */ }
5
6         // Agrega elem a la cola.
7         public void poner(Integer elem) { /* ... */ }
8
9         // Elimina y devuelve el  primer elemento de la cola.
10        public int sacar() { /* ... */ }
11
```

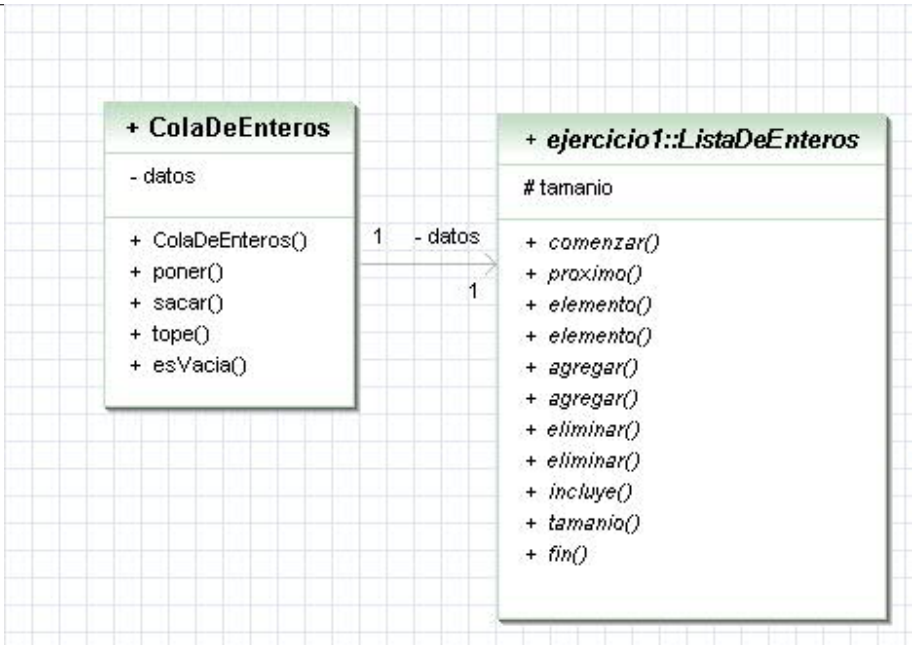

Figura 3: Cola de Enteros

```
12          // Devuelve el elemento en el tope de la cola sin
                eliminarlo.
13          public int tope() { /* ... */ }
14
15          // Retorna true si la Cola está vacía, false en caso
                contrario.
16          public boolean esVacia() { /* ... */ }
17  }
```

a) Implemente en Java (pase por máquina) la clase `ColaDeEnteros` (utilizando alguna subclase de `ListaDeEnteros`) de acuerdo a la especificación dada. Defina esta clase adentro del paquete `practica2.ejercicio3`.

4. El algoritmo conocido como *Criba de Eratóstenes* permite la obtención de todos los números primos menores que un número dado. Y para conseguir esto se procede del modo siguiente:

   Haga una lista de todos los naturales desde 2 hasta un número n dado:

   2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 ... n

   Marque el 2 como primer primo y tache de ahí en adelante todos sus múltiplos:

   ~~[2]~~ 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~ 11 ~~12~~ 13 ~~14~~ 15 ~~16~~ 17 ~~18~~ 19 ~~20~~ 21 ... n

   De los no tachados que siguen a 2, el primero es el siguiente primo. Marque el 3 como primo y tache todos los múltiplos de éste que no haya tachado en el paso anterior:

   ~~2~~ ~~[3]~~ 4 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~ 11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~ ~~21~~ ... n

   El siguiente número a marcar como primo es el 5. Continúe de esta manera hasta haber marcado un número mayor que la raíz cuadrada de n y entonces marque como numeros primos todos los números mayores que 2 que hasta ese momento hayan permanecido sin ser tachados. Los números marcados son todos los primos entre 2 y n.

a) Escriba una clase llamada `CribaDeEratostenes` dentro del paquete `practica2.ejercicio4` como un método llamado `obtenerPrimos()` que tome como parámetro una instancia de `ListaDeEnteros` que contenga los primeros 1000 números naturales y retorne la lista de los primos correspondientes siguiendo el procedimiento antes descrito.

```
1 public static ListaDeEnteros obtenerPrimos(ListaDeEnteros
    numeros) { /*...*/ }
```

5. Entre todas sus fascinaciones, al Faraón le atrae la idea de ver qué grupos de objetos a su alrededor se pueden dividir en dos grupos de igual tamaño. Como Ingeniero/a del Faraón, es tu tarea escribir un programa en Java que, a partir de una lista de enteros que representa la cantidad de elementos de cada grupo, le permita al Faraón descubrir cuántos grupos se pueden dividir en dos sin tener un elemento sobrante (ni tener que dividir el elemento sobrante por la mitad).

```
int contar(ListaDeEnteros L) { ... }
```

6. Se dice que una operación es de **orden constante** cuando, sobre una estructura de datos, no se recorren los elementos de la estructura para averiguar lo pedido.

   Por ejemplo:

   i- Acceder a un elemento de un vector en una posición específica es una operación de **orden constante**.

   ii- la operación de conocer la longitud de una lista puede ser **orden constante** si y solo si este valor se lo almacena en una variable, y cuando se quiere consultar la longitud de la lista se consulta el valor de la variable. De otra forma, recorrer todos los elementos de la lista para conocer cuántos contiene **no es** una operación de tiempo constante, sino que se llama **orden lineal**.

   Se pide:

   (a) Implementar una versión de la clase `PilaDeEnteros` llamada `PilaDeEnterosCte` donde todas las operaciones sean de tiempo constante.

   (b) Implementar una subclase de `PilaDeEnteros` llamada `PilaMin` que permita obtener el mínimo elemento presente en la estructura, también en tiempo constante.

7. Defina una clase llamada `ParGenerico` dentro del paquete `practica2.ejercicio7` con 2 variables de instancia de tipo genérico. Dicha clase representan una dupla de dos elementos del mismo tipo.

   a) Defina un constructor con 2 argumentos que permita crear un par con 2 valores.

   b) Sobrescriba el método `public boolean equals(Object)` de la clase `Object` en `ParGenerico` de manera que la implementación sirva para comparar dos instancias de este tipo.

   c) Sobrescriba el `public String toString()` de la clase `Object` en `ParGenerico` de manera que imprima su contenido con el siguiente formato: "`par[val1, val2]`".

8. Sea la especificación de **Lista Genérica**:

```
1  public abstract class ListaGenerica<T> {
2          /// Se prepara para iterar los elementos de la lista.
3          public abstract void comenzar ();
4
5          /// Avanza al próximo elemento de la lista.
6          public abstract void proximo ();
7
8          /// Determina si llegó o no al final de la lista.
9          public abstract boolean fin();
10
11         /// Retorna el elemento actual.
12         public abstract T elemento ();
13
14         /// Retorna el elemento de la posición indicada (la posición
               va desde 0 hasta n-1).
15         public abstract T elemento (int pos);
16
17         /// Agrega el elemento en la posición actual y retorna true
               si pudo agregar y false si no pudo agregar.
18         public abstract boolean agregar (T elem);
19
20         /// Agrega el elemento en la posición indicada y retorna
               true si pudo agregar y false; si no pudo agregar.
21         public abstract boolean agregar (T elem, int pos);
22
23         /// Elimina el elemento actual y retorna true si pudo
               eliminar y false en caso contrario.
24         public abstract boolean eliminar ();
25
26         /// Elimina el elemento de la posición indicada y retorna
               true si pudo eliminar y false en caso contrario.
27         public abstract boolean eliminar (int pos);
28
29         /// Retorna true si la lista está vacía, false en caso
               contrario.
30         public abstract boolean esVacia();
31
32         /// Retorna true si elem está contenido en la lista, false
               en caso contrario.
33         public abstract boolean incluye (T elem);
34
35         /// Retorna la longitud de la lista.
36         public abstract int tamanio ();
37 }
```

a) Implemente en Java (pase por máquina) una clase abstracta llamada `ListaGenerica` de acuerdo a la especificación dada ubicada dentro del paquete `practica2.ejercicio8`.

b) Escriba una clase llamada `ListaEnlazadaGenerica` como subclase de `ListaGenerica` dentro del paquete anterior, de manera que implemente todos los métodos definidos en la

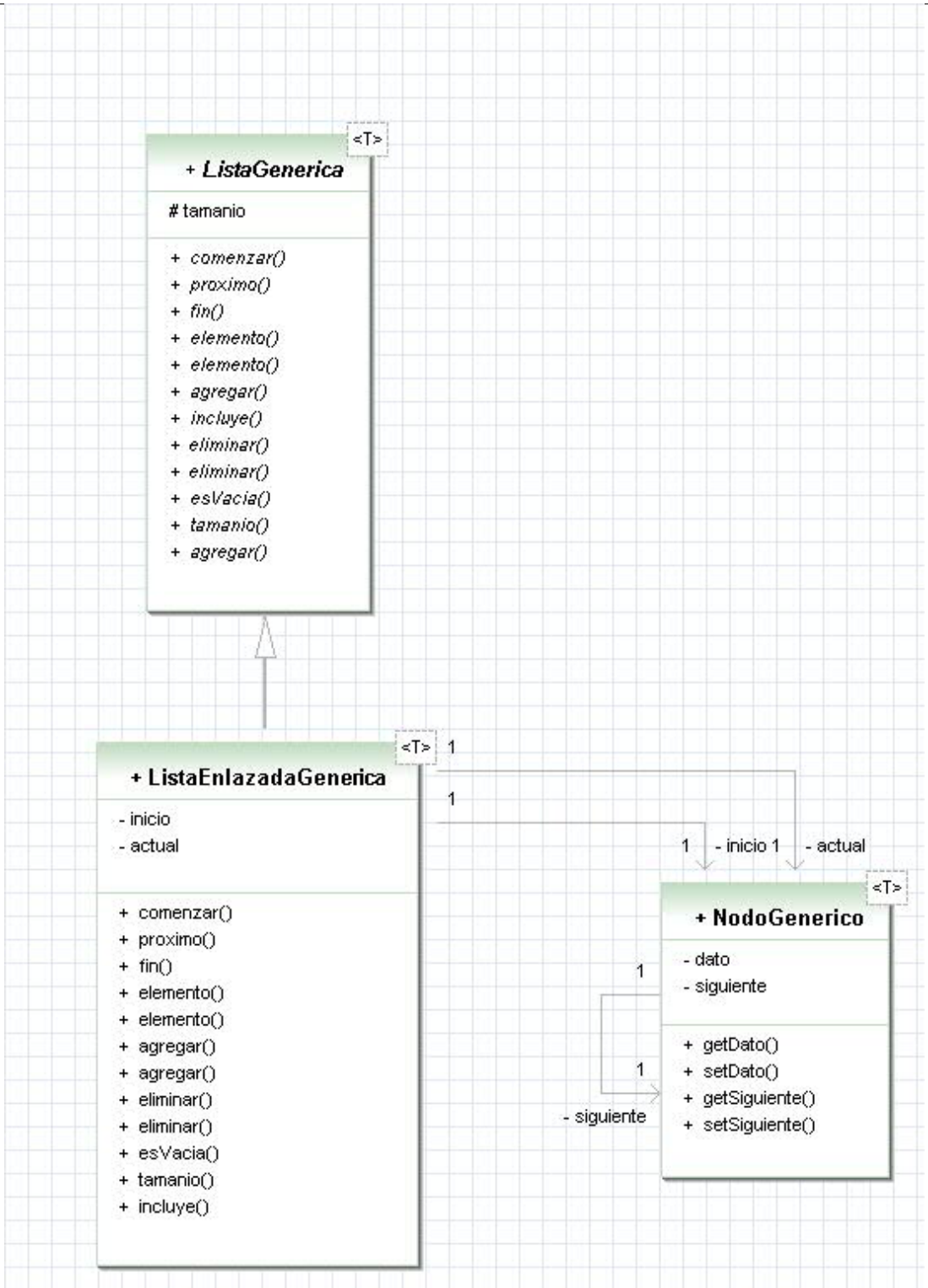

Figura 4: Lista Genérica

superclase pero haciendo uso de una estructura recursiva.

c) Escriba una clase llamada `TestListaEnlazadaGenerica` que reciba en su método `main` una secuencia de `String`s, los agregue a un objeto de tipo `ListaEnlazadaGenerica` y luego imprima los elementos de dicha lista.

9. Considere un string de caracteres `S`, el cual comprende únicamente los caracteres: `(,),[,],{,}`. Decimos que `S` está balanceado si tiene alguna de las siguientes formas:

- `S = ''`, `S` es el string de longitud cero.
- `S = '(T)'`
- `S = '[T]'`
- `S = '{T}'`
- `S = 'TU'`

Donde ambos `T` y `U` son strings balanceados. Por ejemplo, "`{ ( ) [ ( ) ] }`" está balanceado, pero "`( [ ) ]`" no lo está.

a) Implemente una clase llamada `TestBalanceo` dentro del paquete `practica2.ejercicio9` (pase por máquina), cuyo objetivo es determinar si un string dado está balanceado. El string a verificar es un parámetro de entrada (no es un dato predefinido).

Nota: En caso de ser necesario implemente las estructuras de datos que necesite para resolver el ejercicio.

10. Una cola circular es una estructura en la cual, además de poder agregarse y eliminarse elementos como en una cola común, existe la operación rotación, la cual consiste en devolver el tope y encolarlo a la vez.

a) Defina usando Java la clase `ColaCircular` e implemente dos constructores, uno para crear una cola vacía y otro para crearla con un único elemento. Utilice `ListaGenerica<T>`.

b) Implemente las operaciones `poner(T elem)`, `sacar()` y `rotar()`.

c) Puede afirmar que la operación `rotar()` es de **orden lineal**? Justifique.

1. Uso de JUnits

Para instalar los tests de la Práctica 2, hace falta descargar el archivo `tests_TP2_2013.zip` desde el Moodle y descomprimirlo dentro del directorio `src/` del proyecto, de forma que el árbol del proyecto quede como la imagen:

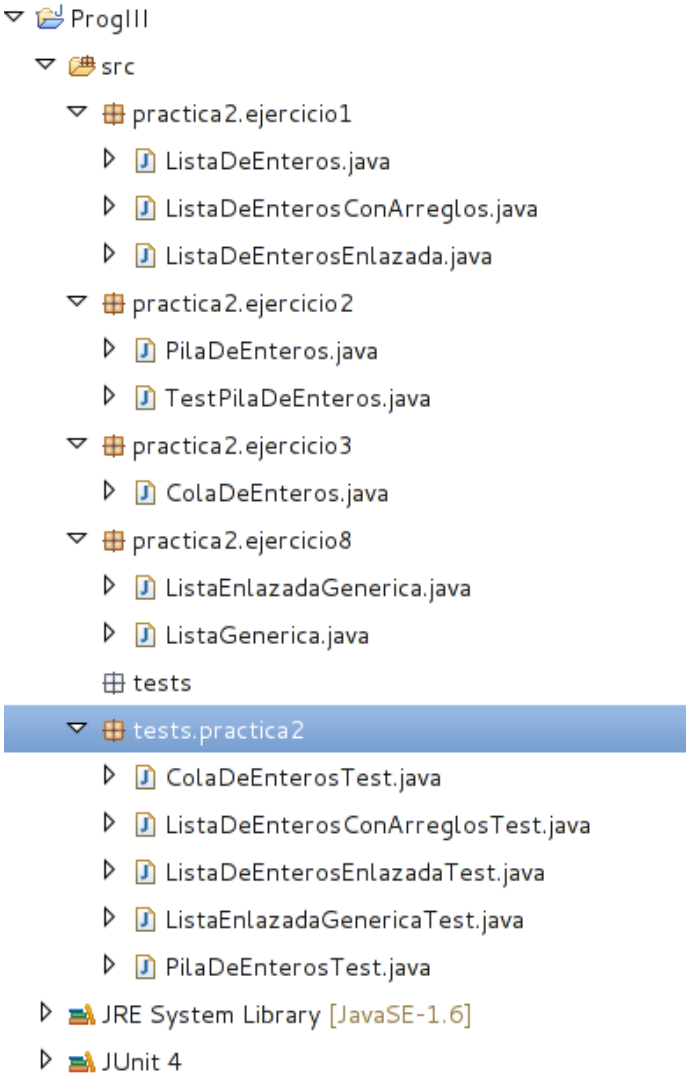

Figura 5: Proyecto con los tests de JUnit