Listas

Laboratorio Estructuras de Datos

Las listas son estructuras que cuentan con operaciones como agregar, eliminar, obtener un elemento, vaciar, etc.
Hay varios tipos de listas y dependiendo de sus características pueden tener algunas ventajas o desventajas sobre otros tipos de listas.
A continuación encontraras algunos tipos de listas:

- **Lista Simple** : Es aquella que sólo cuenta con la referencia al nodo *siguiente*.
- **Lista Doblemente Ligada**: En este tipo de lista tenemos las referencias a los nodos *anterior* y *siguiente*.
- **Lista Circular** : En ella el último nodo apunta al primero, esta puede ser doblemente ligada o simplemente ligada dependiendo de la implementación.

Ejemplo: Imagina que tuvieras que ir a comprar el mandado si tienes una buena memoria podrías grabarte lo que tienes que comprar mentalmente pero si no es el caso lo más común es hacer una lista, la lista es un tipo de estructura que es muy fácil de usar y de entender, la ventaja de una lista es que puedes acceder a cualquiera de los elementos a lo largo de ella y a diferencia de una pila o cola (estructuras que tienen operaciones de agregar o eliminar que respetan un orden asignado) la lista puede ser modificada de forma diferente a como se fue creando es decir puedes eliminar cualquier elemento en su interior sin importar si es el primero o el último.

Figura 1: Lista de Súper: De la misma forma que en la imagen en una lista puedes eliminar los elementos en un orden diferente al que se agregaron.

En java podemos crear una Lista de la siguiente forma.

- **Listas usando la interfaz List y la clase ArrayList**

```
List<String>pila = new ArrayList<>() // Lista de cadenas
List<Integer>pila = new ArrayList<>() // Lista de enteros
```

- **Listas usando la interfaz List y la clase LinkedList**

```
List<String>pila = new LinkedList<>() // Lista de cadenas
List<Integer>pila = new LinkedList<>() // Lista de enteros
```

Métodos para manejar Listas

- *add(E item):* Este método sirve para guardar un elemento al final de la lista.
- *add(String item, int index):* Este método sirve para guardar un elemento en la posición indicada.
- *remove(int i)* Este método nos permite eliminar el elemento en la posición i.
- *remove(Object o):* Elimina la primera aparición del objeto.
- *get(int i)* Este método nos permite ver el elemento en la posición i.
- *clear():* Elimina todos los elementos de la lista.
- *size():* Regresa el tamaño de la lista.

Código Recurso 01: Lista Simplemente Ligada

Puedes observar la interfaz List y su implementación, recuerda que List es interfaz por lo que en ella sólo definimos el comportamiento de la lista y en la clase Lista implementamos ese comportamiento.

List

En esta interfaz se definen algunos métodos de la lista.

- *agrega(E e):* Este método debe poder agregar un elemento al final de la lista.
- *agrega(int index, E e):* Este método debe poder agregar el elemento en el indice indicado.
- *obtiene(int index):* Este método debe poder regresar el elemento guardado en la posición indicada.

- *elimina(int index):* Este método debe poder eliminar el elemento que se guardó en el indice indicado.
- *tamaño():* Nos regresa el tamaño de la lista.
- *contiene(E e):* Nos dice si el elemento esta contenido en la lista.
- *toString():* Este método no es característico de una lista pero nos permite obtener una representación en cadena de ella para verificar que estemos realizando las otras operaciones de forma correcta.

```java
/**
 * Interfaz que define los metodos que debe tener una lista.
 */
public interface List<E> {

    /**
     * Agrega un elemento al final de la lista.
     * @param e, el elemento a agregar.
     */
    public void agrega(E e);

    /**
     * Agrega un elemento a la lista en una posicion especifica.
     * @param index, la posicion en la que se agregara el elemento.
     * @param e, el elemento a agregar.
     */
    public void agrega(int index, E e);

    /**
     * Obtiene un elemento de la lista en una posicion especifica.
     * @param index, la posicion del elemento a obtener.
     * @return el elemento en la posicion index.
     */
    public E obtiene(int index);

    /**
     * Elimina un elemento de la lista en una posicion especifica.
     * @param index, la posicion del elemento a eliminar.
     * @return el elemento eliminado.
     */
    public E elimina(int index);

    /**
     * Obtiene el tamanio de la lista.
     * @return el tamanio de la lista.
     */
    public int tamanio();

    /**
     * Verifica si la lista contiene un elemento en particular.
     * @return true si la lista contiene el elemento, false en otro caso.
     */
    public boolean contiene(E e);
```

```
44
45      /**
46       * Regresa una representacion en cadena de la lista.
47       * @return cadena, la representacion en cadena de la lista.
48       */
49      public String toString();
50  }
```

Lista

En esta clase está implementada *List* además cuenta con dos clases internas una que representa los nodos y otra que nos dará el iterador para poder movernos por ella.

```
1   /**
2    * Clase que implementa la interfaz List.
3    */
4   public class Lista<E> implements List<E> {
5       /* Cabeza, el primer nodo de la lista */
6       private Nodo cabeza;
7       /* Cola, el ultimo nodo de la lista */
8       private Nodo cola;
9       /* Tamanio, el tamanio de la lista */
10      private int tamanio;
11
12      /**
13       * Clase que representa un nodo de la lista
14       */
15      public class Nodo{...}
16
17      /**
18       * Clase que implementa la interfaz Iterator, nos sirve para poder iterar sobre
19       * la lista.
20       */
21      public class Iterador implements Iterator<E>{...}
22
23      /**
24       * Obtiene un iterador de la lista.
25       * @return un iterador de la lista.
26       */
27      public Iterator<E> iterator() {
28          return new Iterador();
29      }
30
31      /**
32       * Agrega un elemento al final de la lista.
33       * @param e, el elemento a agregar.
34       */
35      public void agrega(E e) {
36          ...
37      }
38
39      // ---- [ Otros Metodos ] ----
40
```

```
41      /**
42       * Regresa una representacion en cadena de la lista.
43       * @return cadena, la representacion en cadena de la lista.
44       */
45      public String toString() {
46          ...
47      }
48  }
```

Clase interna Nodo

El nodo es un objeto que nos sirve para crear la lista, la lista va a ser una serie de nodos conectados de manera lineal. Debido a que la lista que se implemento es una lista simplemente ligada el nodo solo debe tener la referencia a un elemento y al nodo siguiente.

Figura 2: La lista simple está compuesta por nodos que tienen referencias al nodo siguiente y al elemento. En la lista de la imagen puedes observar una lista de enteros cuya longitud es 5, con índices del 0 al 4.

A continuación se muestra la clase Nodo alojada en la clase Lista.

```
1   /**
2    * Clase que define un nodo de la lista.
3    */
4   public class Nodo {
5       /* El elemento dentro del nodo */
6       E elemento;
7       /* El nodo siguiente */
8       Nodo siguiente;
9
10      /**
11       * Constructor de la clase Nodo.
12       *
13       * @param e, el elemento del nodo.
14       */
15      public Nodo(E e) {
16          elemento = e;
17          siguiente = null;
18      }
19
20      /**
21       * Constructor de la clase Nodo.
22       *
```

```
23        * @param e, el elemento del nodo.
24        * @param n, el nodo siguiente.
25        */
26       public Nodo(E e, Nodo n) {
27           elemento = e;
28           siguiente = n;
29       }
30   }
```

Clase interna Iterador

La clase Iterador debe implementar la interfaz Iterator de java, debemos tener un nodo que se llame *siguiente* o *actual* este nodo representara el nodo que sigue después del iterador.

Las operaciones *hasNext()* y *next()* nos permitirán iterar sobre la lista, con *hasNext()* preguntamos si hay un siguiente nodo y si lo hay con *next()* podemos obtener el elemento que esta en ese nodo siguiente y movernos un nodo después.

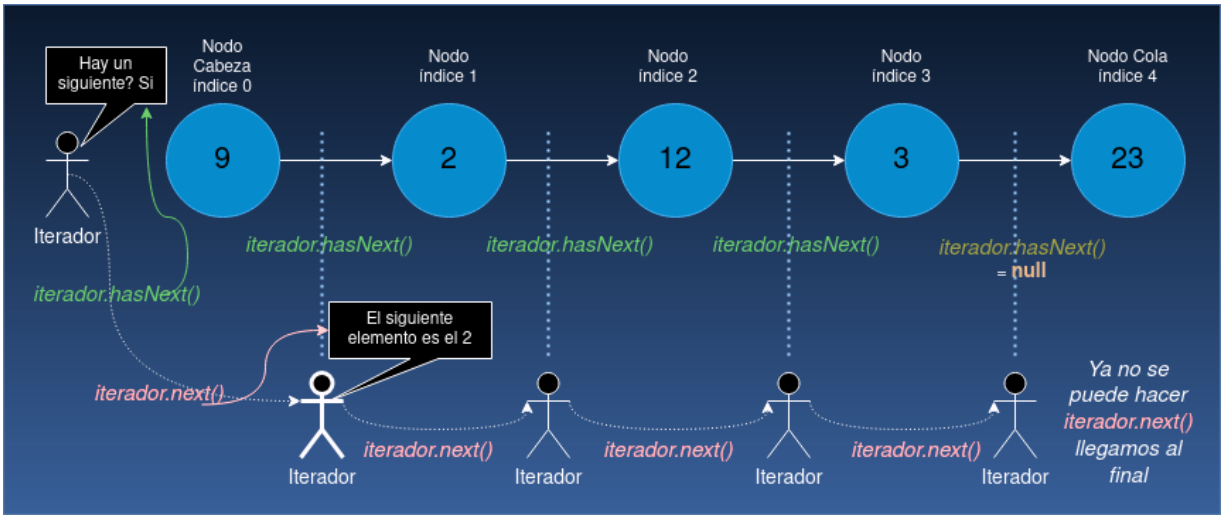

Figura 3: Puedes visualizar el iterador como un ente que esta entre los nodos, al iniciar a iterar debe estar antes del nodo cabeza por lo que el nodo actual/siguiente debe ser la cabeza.

Puedes observar a continuación la clase iterador alojada en la clase Lista.

```
1  /**
2   * Clase que implementa la interfaz Iterator.
3   */
4  public class Iterador implements Iterator<E> {
5      /* El nodo donde se encuentra el iterador */
6      private Nodo actual;
7
8      /**
9       * Constructor de la clase Iterador.
10      */
```

```
    public Iterador() {
        actual = cabeza;
    }

    /**
     * Verifica si hay un siguiente elemento en la lista.
     *
     * @return true si hay un siguiente elemento, false en otro caso.
     */
     public boolean hasNext() {
         return actual != null;
     }

     /**
      * Obtiene el siguiente elemento de la lista.
      *
      * @return el siguiente elemento.
      */
      public E next() {
          if (!hasNext()) {
              throw new NoSuchElementException("No hay siguiente elemento");
          }
          E e = actual.elemento;
          actual = actual.siguiente;
          return e;
      }
  }
```

ClasePrincipal

La clase principal contiene el método main por lo que es la clase que se especifica en el archivo *pom.xml*, el menú contiene cinco opciones las tres primeras sirven para interactuar con una lista ya sea de animes, estudiantes o libros, la cuarta opción tiene el funcionamiento de una lista con ejemplos predeterminados. Hay tres métodos para manejar la interacción del usuario con los diferentes tipos de lista y otros tres métodos que sirven para crear un anime, un estudiante o un libro puedes observar la documentación de cada método.
La estructura de la clase principal es la siguiente:

```
/**
 * Clase principal para mostrar el uso de la lista.
 */
 public class ClasePrincipal {
    /* Variables de clase, colores y formatos */
    private static String RESET = "\u001B[0m";
    private static String RED = "\u001B[31m";
    private static String YELLOW = "\u001B[33m";
    private static String BLUE = "\u001B[34m";
    private static String BOLD = "\u001B[1m";
    private static Scanner entrada = new Scanner(System.in);
    private static List<Anime> listaAnime;
    private static List<Estudiante> listaEstudiante;
    private static List<Libro> listaLibro;
```

```java

    /**
     * Metodo que despliega el primer menu para seleccionar el tipo de lista.
     * @param args, los argumentos de la linea de comandos.
     */
    public static void main(String[] args) {...}

    /* Metodos que despliegan menus por cada tipode lista */
    public static void menuInteraccionAnime() {...}
    public static void menuInteraccionEstudiante() {...}
    public static void menuInteraccionLibro() {...}

    /* Metodos que crean un objeto dependiendo del tipo de la lista */
    public static Anime agregaAnime() {...}
    public static Estudiante agregaEstudiante() {...}
    public static Libro agregaLibro() {...}

    /* Metodos auxiliares para verificar que la cadena represente un numero */
    private static int verificaEntero() {...}
    private static double verificaDouble() {...}

    /* Metodo que imprime las operaciones de una lista en ejemplos predeterminados */
    public static void representacionPrederminado() {...}
}
```

Capturas

La clase principal tiene el siguiente menú:

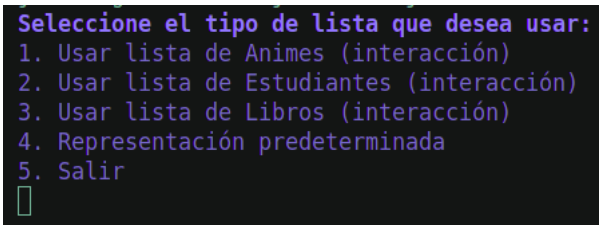

Figura 4: Menú principal del recurso 01.

Para interactuar con el solo escribe el número que corresponde a la opción y pulsa enter. Al elegir alguna de las tres opciones de listas interactivas se debe desplegar un menú similar a este:

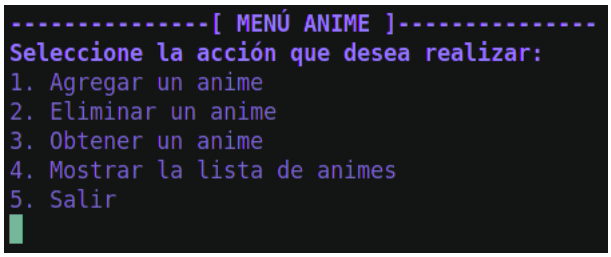

Figura 5: Menú para interactuar con una lista de animes, cada tipo de lista cuenta con su propio menú.

Interactuá con el de la misma forma.
Si eliges la representación predeterminada podrás ver algo así:

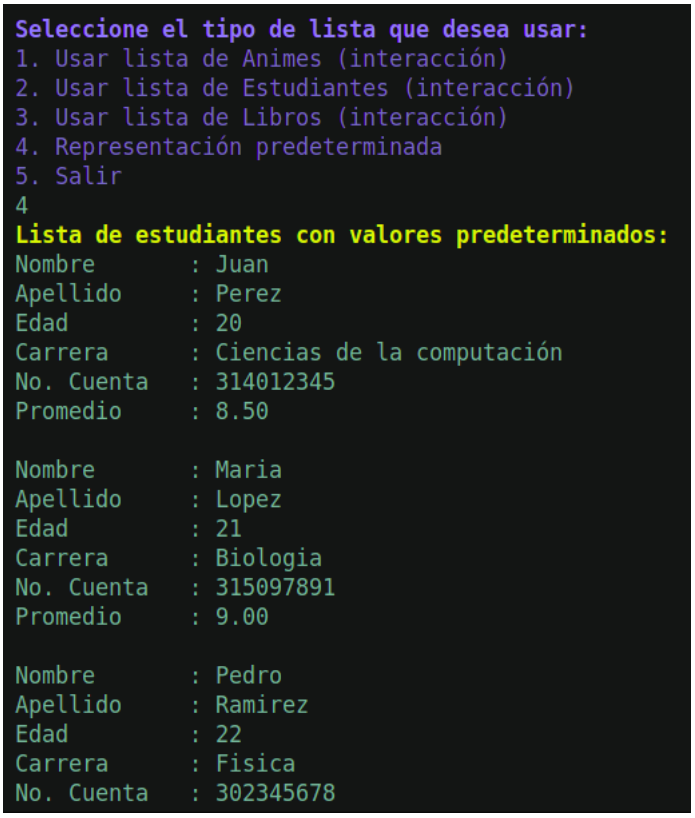

Figura 6: Ejemplos predeterminados opción 4 del menú.