Introducción Laboratorio

Estructuras de Datos y Algoritmos

Nicolás Núñez

1 de septiembre de 2023

Universidad Diego Portales

Contenidos

1. Introducción

2. Java + IDE + Entregas

3. Ejercicio 1

4. Ejercicio 2

Introducción

Introducción

- La idea del laboratorio es que puedan aplicar lo que aprenden en cátedra y que también desarrollen análisis.
- Inicialmente serán 3 laboratorios (promediados).
- El laboratorio tiene un valor importante dentro de la nota del curso: ¿30 %? (por confirmar)
- Se informará la asistencia a laboratorios (obligatorios).

Sobre Java, el entorno de desarrollo y entregas

Java

- Les recomiendo utilizar el resumen que les pasó el profesor de cátedra: `https://udp.instructure.com/courses/23338/files/3104035`
- Vamos a utilizar muchos elementos que nos proporciona el Lenguaje y *POO*: *clases*, *interfaces*, *comparators*, etc... ¡A repasar!
- **No podrán usar las TDA** (*Tipos de datos abstractos*) **definidas por Java**. **Van a tener que implementar todo desde cero**. Así que es bueno que empiecen a crear sus propias estructuras que reutilizarán durante el semenstre :P.

IDE

Esto sólo son recomendaciones...

- *Para mi (ayudante 🤓): Visual Studio Code + JDK para compilar* **¡GOD!**
- *No me quiero complicar la vida: IntelliJ IDEA*
- *Otros: Eclipse, Netbeans...*

Lo importante: que estén cómodos haciendo código.

Entregas

- Cuando entreguen código, entreguen los archivos necesarios (¡¡¡*sólo los* ***.java****!!!!*).
- El código no va en un PDF, Word, etc. -> Se pierde el formato...
- Esto quiere decir que los **.class** no van.
- No olviden el PDF con el informe, siempre se les pedirá.

Ejercicio 1

Lista enlazada simple

Construya una lista enlazada simple a partir de la siguiente *Interface*:

```
1  public interface LinkedList<T> {
2      void InsertAtHead(T dato);
3      void InsertAtEnd(T dato);
4      boolean DeleteOnHead();
5      boolean DeleteAtEnd();
6      boolean contains(T dato);
7      int size();
8      boolean isEmpty();
9  }
```

Descripciones

`insertAtStart(T data)` Este método inserta un elemento al inicio de la lista. Toma los datos a insertar como argumento.

`insertAtEnd(T data)` Este método inserta un elemento al final de la lista. Al igual que el método anterior, toma los datos a insertar como argumento.

`deleteAtStart()` Elimina un elemento al inicio de la lista. Este método devuelve `true` si la operación fue exitosa y `false` en caso contrario.

`deleteAtEnd()` Elimina un elemento al final de la lista. Devuelve `true` si la operación fue exitosa y `false` en caso contrario.

`contains(T data)` Busca un elemento en la lista. Devuelve `true` si el elemento se encuentra y `false` en caso contrario.

Descripciones

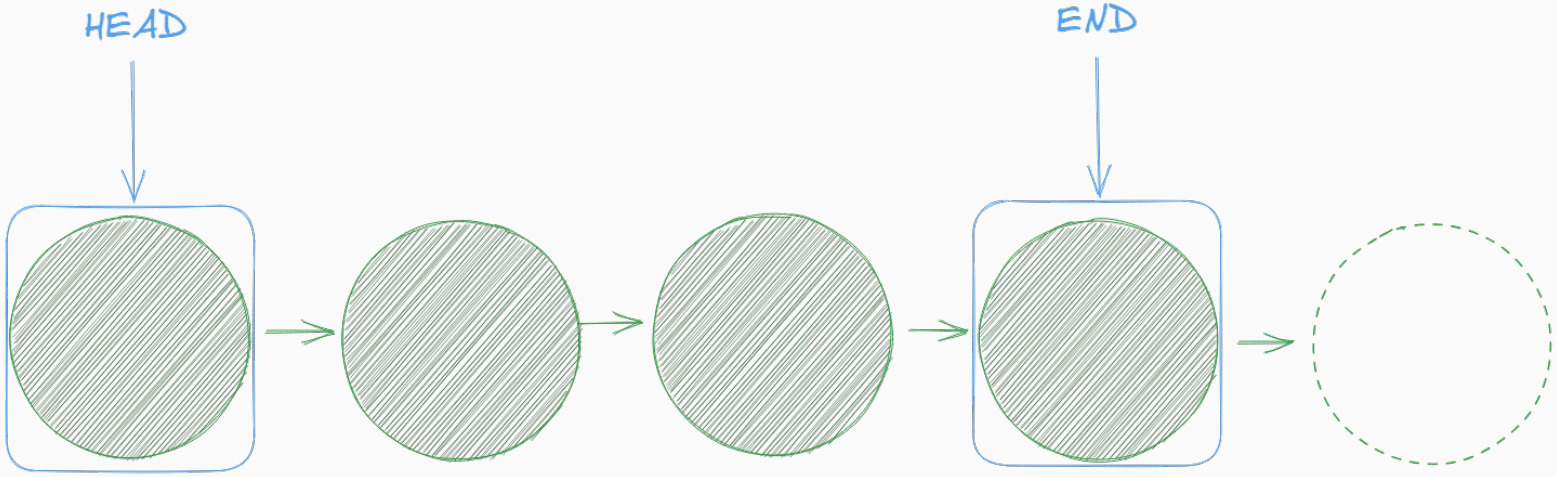
`size()` Retorna el número de elementos en la lista, es decir, el tamaño de la lista.

`isEmpty()` Indica si la lista está vacía. Devuelve `true` si no hay elementos en la lista y `false` en caso contrario.

Solución

Primero... ¿Qué elementos necesitamos para armar una lista enlazada? ¿Los tenemos?

R: Necesitamos primero crear una clase Nodo!



Solución - Clase Node

```java
public class Node<T> {
    private T data;
    private Node<T> next;
    public Node(T data) {
        this.data = data;
        this.next = null;
    }
    public T getData() {
        return data;
    }
    public void setData(T data) {
        this.data = data;
    }
    public Node<T> getNext() {
        return next;
    }
    public void setNext(Node<T> next) {
        this.next = next;
    }
}
```

Solución - Implementación de algunas funciones de Linked List

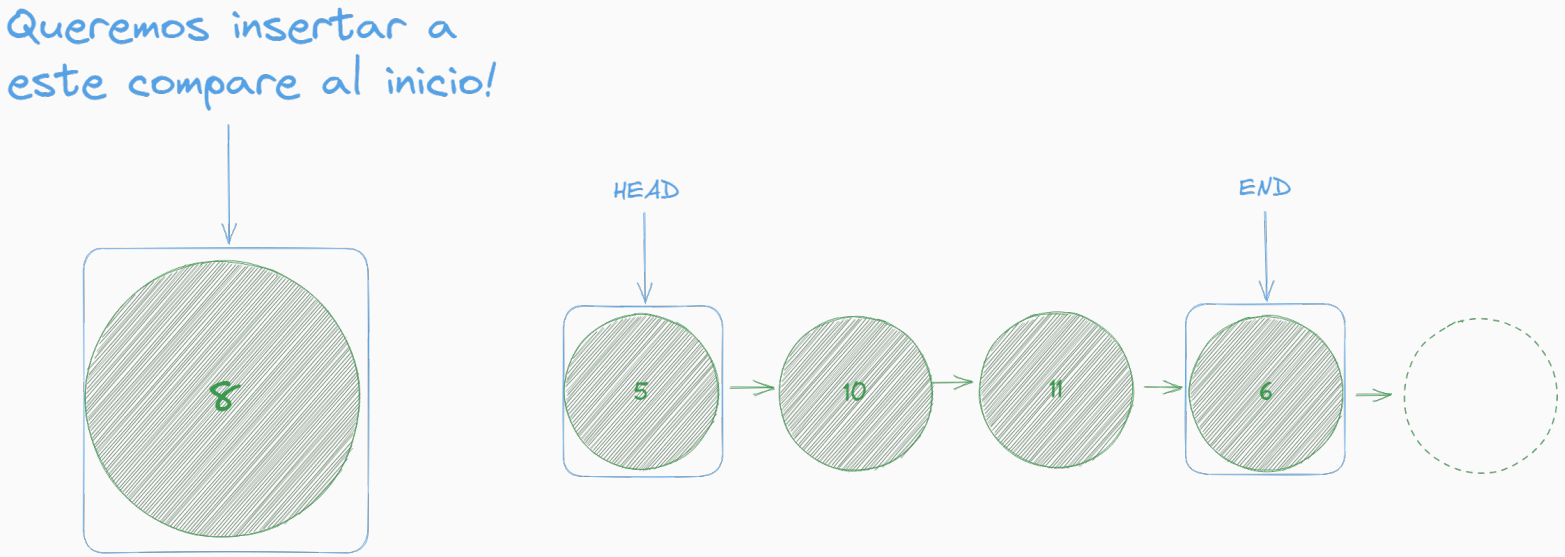
```java
public class SimpleLinkedList<T> implements LinkedList<T>
    {
    private Node<T> head;

    public SimpleLinkedList() {
        this.head = null;
    }

    @Override
    public void InsertAtHead(T dato) {
        Node<T> newNode = new Node<>(dato);
        newNode.setNext(head);
        head = newNode;
    }
    ...
```

Solución - Algunos detalles de sintaxis

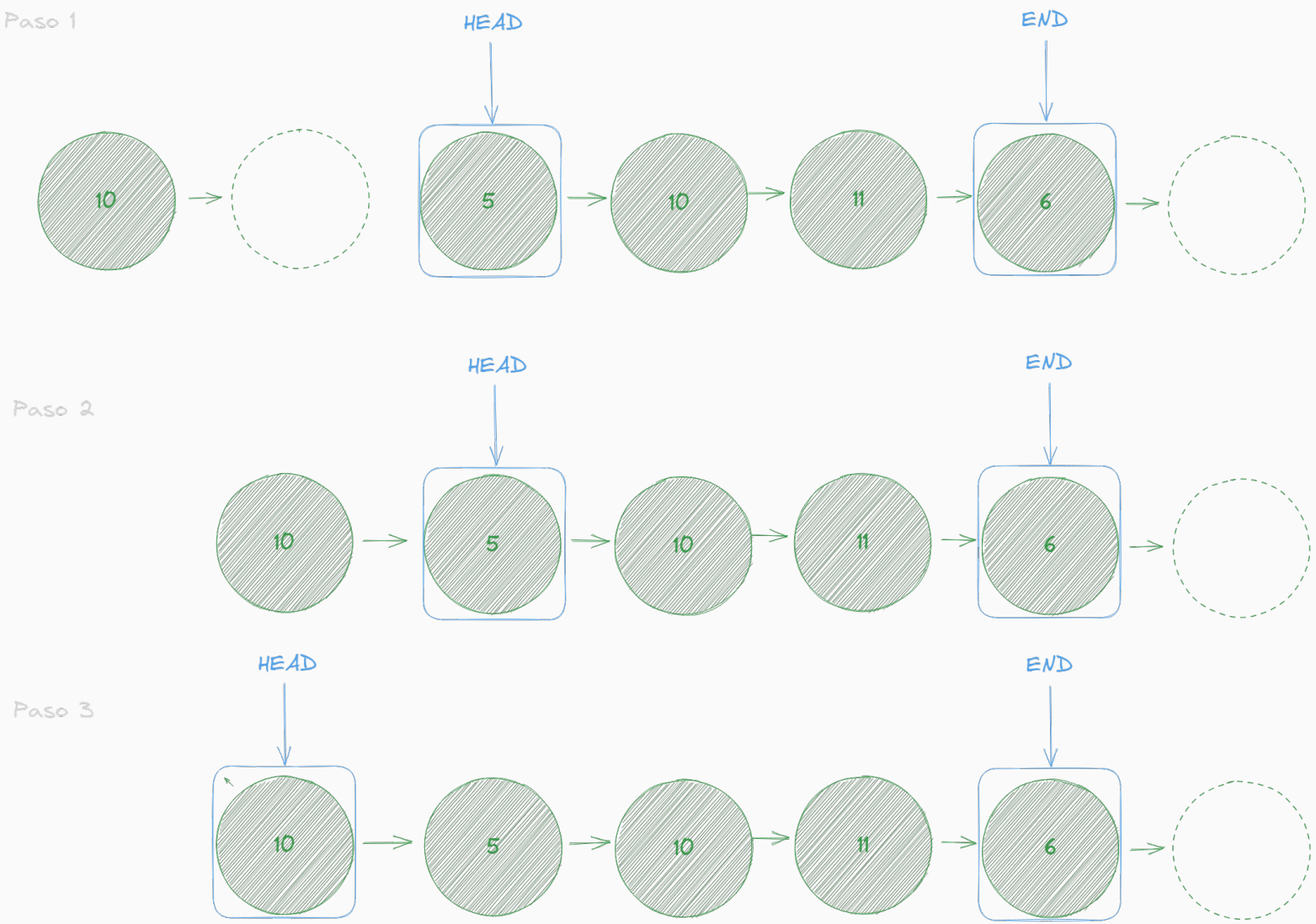
`T` es un tipo de dato cualquiera. Esto es útil para definir tipos estructuras de datos genéricas... Porque podríamos guardar en el nodo un: *Integer*, *String* o incluso un objeto! De todas formas, si usteden lo desean, podrían poner `<Integer>` o `<String>` en vez de `<T>`

`@Override` permite *sobrescribir* un método de una clase. En este caso, queremos sobrescribir los métodos que se heredan desde la superclase creada desde la *Interface*.

Solución - ¿Qué estamos haciendo?



Solución - ¿Qué estamos haciendo?



Solución

- ¡Los demás métodos complételos usted!

Ejercicio 2

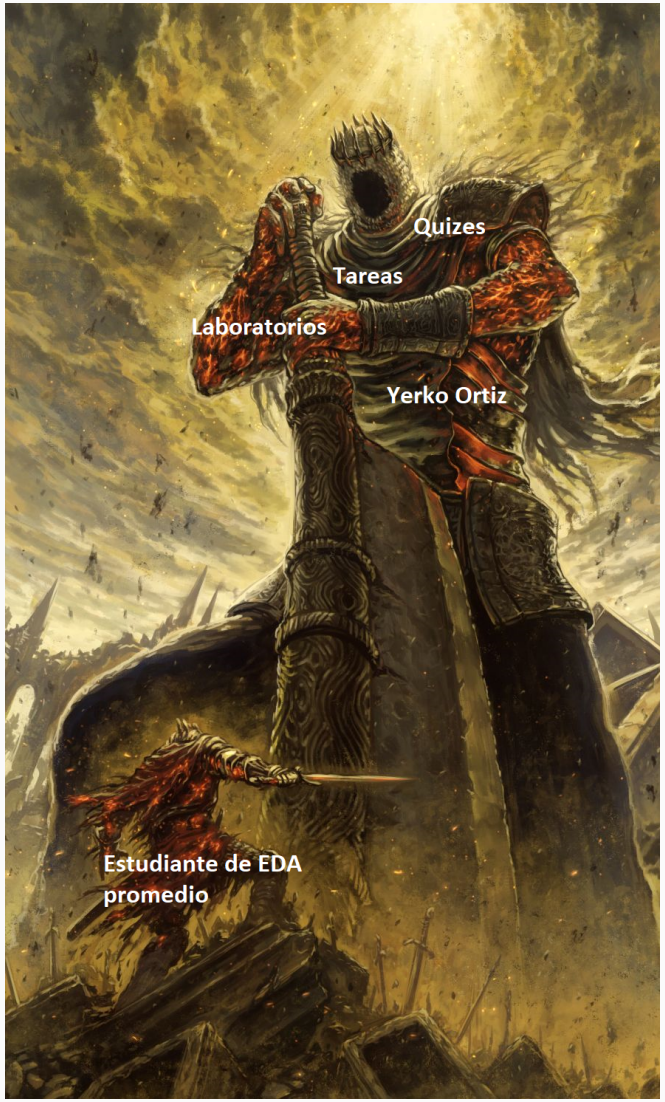
Ejercicio 2

Estos ejercicios son la continuación del ejercicio 1.

- En términos de Big Oh O:
  1. ¿Qué complejidad tiene insertar un elemento en *Head* en una *Lista enlazada simple*?
  2. ¿Qué complejidad tiene insertar un elemento al final en una *Lista enlazada simple*? ¿De qué depende?
  3. ¿Qué complejidad tiene buscar un elemento en una *Lista enlazada simple*? ¿De qué depende?
  4. Si por ejemplo los nodos contuviesen un número entero ¿Qué complejidad tiene insertar los elementos de manera ordenada?
  5. Si por ejemplo los nodos contuviesen un número entero ¿Qué complejidad tiene eliminar un elemento específico sin saber dónde se encuentra?

Próximo capítulo...

Empezar el Laboratorio 1...



La frase del día

"Comentar el código es como limpiar el cuarto de baño; nadie quiere hacerlo, pero el resultado es siempre una experiencia más agradable para uno mismo y sus invitados" – ***Ryan Campbell***