Informe de Laboratorio 04

Tema: Sort y Listas Enlazadas

Nota

Estudiante	Escuela	Asignatura
• Javier Coronado Peña • Max Sebastian Foroca Mamani • Diego Claudio Nina Suyo	Escuela Profesional de Ingeniería de Sistemas	Estructura de Datos y Algortimos Semestre: I Código: 20190214

Laboratorio	Tema	Duración
04	Sort y Listas Enlazadas	02 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - A	Del 14 Junio 2023	Al 17 Junio 2023

1. Tarea

Ejercicio 1. Utilizar el tipo generico de Lista Enlazada para generar los peores casos y ejecutar el algoritmo de ordenamiento.
Clase Nodo: Clase nodo implementado para usarse en una lista enlazada

Listing 1: Node

```
     // NODO DE LISTA ENLAZADA
// package Ejercicio01;

public class Nodo<T> {
    // Atributos de Nodo
    private T dato;
    private Nodo<T> next;
    // Constructores
    public Nodo(T dato){
        this(dato,null);
    }
    public Nodo(T dato, Nodo<T> next){
        this.dato = dato;
        this.next = next;
    }
    // Getters y Setters de los atributos
    public T getDato() {
        return dato;
```

```
    }

    public void setDato(T dato) {
        this.dato = dato;
    }

    public Nodo<T> getNext() {
        return next;
    }

    public void setNext(Nodo<T> next) {
        this.next = next;
    }
    // Metodo toString: devuelve el toString del dato
    @Override
    public String toString(){
        return this.dato.toString();
    }
}
```

Clase LinkedList:

Listing 2: LinkedList

```
   //Clase linkedList
// package Ejercicio01;

public class LinkedList<T> {
    // Atributos de la clase generica LinkedList
    private Nodo<T> raiz;
    private int size;
    //Constructor
    public LinkedList(){
        this.raiz = null;
        this.size = 0;
    }
    // Metodo size: devuelve la longitud del LinkedList
    public int size() {
   return this.size;
    }
    // Metodo isEmpty: indica si la lista enlazada esta vacia
    public boolean isEmpty() {
        return this.size == 0;
    }
    // Metodo get: devuelve el nodo del indice indicado
    public Nodo<T> get(int indice){
        Nodo<T> aux = this.raiz;
        for(int i = 0; i < indice ; i ++){
            aux = aux.getNext();
        }
        return aux;
    }
    // Metodo add: agrega un nuevo elemento al final de la lista enlazada
    public void add(T dato){
        if(this.size == 0){
            this.raiz = new Nodo<T>(dato);
```

```
            this.size ++;
        }else{
            this.get(this.size()-1).setNext(new Nodo<T>(dato));
            this.size++;
        }
    }
    // Metodo remove: desenlaza al nodo del indice indicado
    public void remove(int indice) {
        if (indice < this.size) {
            if(indice == 0){
                this.raiz = this.raiz.getNext();
            }else{
                this.get(indice-1).setNext(this.get(indice+1));
            }
            this.size--;
        }
    }
    // Metodo toString: devuelve un String con los elementos de la lista enlazada
    //separada por ,
    @Override
    public String toString() {
        String str = "";
        for(Nodo<T> aux = this.raiz; aux != null; aux = aux.getNext())
            str += aux.toString() + ", ";
        return str;
    }
}
```

Clase Ejercicio01: Esta clase contiene los imports que se utilizaran para la ejecución del ejercicio y los métodos main, generarPeorCaso e InsercionSort. Implementados para ser usados con la clase LinkedList genérica.

Listing 3: Main

```
// package Ejercicio01;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Scanner;
import com.panayotis.gnuplot.JavaPlot;
public class Ejercicio01 {
    // Metodo principal (main): indica que puede saltar una exception
    public static void main(String[]args) throws IOException{
        int tamano;
        Scanner teclado = new Scanner(System.in);
        System.out.print("Ingrese el tamano maximo del ultimo arreglo: ");
        tamano = teclado.nextInt();
        ArrayList<LinkedList<Integer>> casos = new
            ArrayList<LinkedList<Integer>>();
        String archivoInsercion = "insercion.txt";
        PrintWriter oS = new PrintWriter(archivoInsercion);
        // Generan peores casos y se agregan al ArrayList
        for(int n = 1; n <= tamano ; n++){
            casos.add(generarPeorCaso(n));
```

```
        }

        Iterator<LinkedList<Integer>> puntero = casos.iterator();
        while(puntero.hasNext()){
            oS.println(String.valueOf(insertionSort(puntero.next())));
        }
        oS.close();

        JavaPlot p = new JavaPlot();
        p.addPlot("\""+archivoInsercion+ "\" with lines");
        p.plot();
    }
    //Metodo generarPeorCaso: genera peores casos de ordenamiento en un LinkedList
    public static LinkedList<Integer> generarPeorCaso(int t){
        LinkedList<Integer> aux = new LinkedList<Integer>();
        for(int i = 0; i < t; i++){
            aux.add(t-i);
        }
        return aux;
    }
    //Metodo insertionSort: ordena LinkedList de menor a mayor
    public static long insertionSort(LinkedList<Integer> list){
        int key;
        int i;
        long nano_startTime =System.nanoTime();
        for(int j = 1; j<list.size(); j = j+1){
            key =(int)(list.get(j).getDato());
            i = j-1;
            while(i>-1&&(int)(list.get(i).getDato())>key){
                list.get(i+1).setDato(list.get(i).getDato());
                i = i-1;
            }
            list.get(i+1).setDato(key);
            System.out.println(list);
        }
        long nano_endTime = System.nanoTime();
        return nano_endTime - nano_startTime;
    }
}
```

Método main: En la cabecera de este método indicamos que puede lanzar una IOException , en el cuerpo se crea la variables tamano que guardará el tamaño máximo del último LinkedList a crear, el ArrayList casos guarda LinkedList de Integers, archivoInsercion que tiene el nombre del archivo txt que guardará los datos del tiempo de ejecución y el objeto oS de la clase PrintWriter, que crea y escribe archivos.

Listing 4: Main

```
  public class Ejercicio01 {
    // Metodo principal (main): indica que puede saltar una exception
    public static void main(String[]args) throws IOException{
        int tamano;
        Scanner teclado = new Scanner(System.in);
        System.out.print("Ingrese el tamano maximo del ultimo arreglo: ");
        tamano = teclado.nextInt();
        ArrayList<LinkedList<Integer>> casos = new
```

```
            ArrayList<LinkedList<Integer>>();
        String archivoInsercion = "insercion.txt";
        PrintWriter oS = new PrintWriter(archivoInsercion);
```

También se crea un for que itera la cantidad de veces del valor de tamano y se añade a casos los datos que retorna el método generarPeorCaso. El objeto puntero se crea usando Iterator y será el que recorrerá el ArrayList casos. Además se tiene un bucle while que repetirá su proceso mientras puntero tiene un siguiente elemento por recorrer, dentro del bucle se escribe en el archivo txt los valores que devuelve el método insertionSort. Y se cierra la transmisión de datos al archivo con el método close. Finalmente se utiliza el objeto p de JavaPlot para agregar los datos guardados en el archivo txt y ejecutarlos en gnuplot.

Listing 5: FOR

```
    // Generan peores casos y se agregan al ArrayList
      for(int n = 1; n <= tamano ; n++){
          casos.add(generarPeorCaso(n));
      }

      Iterator<LinkedList<Integer>> puntero = casos.iterator();
      while(puntero.hasNext()){
          oS.println(String.valueOf(insertionSort(puntero.next())));
      }
      oS.close();

      JavaPlot p = new JavaPlot();
      p.addPlot("\""+archivoInsercion+ "\" with lines");
      p.plot();
  }
```

Metodo generarPeorCaso: Retorna un LinkedList de una longitud enviado en su parámetro, con los datos de mayor a menor (peor caso)

```
  //Metodo generarPeorCaso: genera peores casos de ordenamiento en un LinkedList
  public static LinkedList<Integer> generarPeorCaso(int t){
      LinkedList<Integer> aux = new LinkedList<Integer>();
      for(int i = 0; i < t; i++){
          aux.add(t-i);
      }
      return aux;
  }
```

Metodo insertionSort: Crea variables key e i que almacenan temploralmente el valor actualy el indice,nanostartTime es para medir el tiempo de ejecucion en nanosegundos.

```
//Metodo insertionSort: ordena LinkedList de menor a mayor
    public static long insertionSort(LinkedList<Integer> list){
        int key;
        int i;
        long nano_startTime =System.nanoTime();
        for(int j = 1; j<list.size(); j = j+1){
            key =(int)(list.get(j).getDato());
            i = j-1;
            while(i>-1&&(int)(list.get(i).getDato())>key){
```

```
            list.get(i+1).setDato(list.get(i).getDato());
            i = i-1;
         }
         list.get(i+1).setDato(key);
         //System.out.println(list);
      }
      long nano_endTime = System.nanoTime();
      return nano_endTime - nano_startTime;
   }
```

Ejercicio 2. Utilizar el tipo generico de Doble Lista Enlazada para generar los peores casos y ejecutar el algoritmo de ordenamiento. Clase Nodo: Clase nodo implementado para usarse en una lista enlazada

Listing 6: caption

```
public class Node <E > {
   private E data ;
   private Node <E > nextNode ;
   private Node <E > previousNode ;

   /* Getters y Setter ... */

   Node () {
      this . data = null ;
      this . nextNode = null ;
      this . previousNode = null ;
   }
   Node ( E data ) {
      this . data = data ;
      this . nextNode = null ;
      this . previousNode = null ;
   }
   Node ( E data , Node <E > nextNode ) {
      this . data = data ;
      this . nextNode = nextNode ;
      this . previousNode = null ;
   }
   Node ( E data , Node <E > nextNode , Node <E > previousNode ) {
      this . data = data ;
      this . nextNode = nextNode ;
      this . previousNode = previousNode ;
   }
}
```

Clase DoubleLinkedList

Listing 7: DoubleLinkedList

```
public class DoubleLinkedList<E> {

   private Node<E> head;
   private Node<E> tail;
   private int size;
```

```
public DoubleLinkedList() {
    this.head = null;
    this.tail = null;
    this.size = 0;
}

public Node<E> getHead() {
    return head;
}

public void setHead(Node<E> head) {
    this.head = head;
}

public Node<E> getTail() {
    return tail;
}

public void setTail(Node<E> tail) {
    this.tail = tail;
}

// Metodo size: devuelve la longitud del LinkedList
public int size() {
    return this.size;
}

// Metodo isEmpty: indica si la lista enlazada esta vacia
public boolean isEmpty() {
    return this.size == 0;
}

// Metodo get: devuelve el nodo del indice indicado
public Node<E> get(int indice) {
    if (indice >= 0 && indice < this.size) {
        Node<E> aux;
        if (this.size / 2 > indice) {
            aux = this.head;
            for (int i = 0; i < indice; i++) {
                aux = aux.getNextNode();
            }
        } else {
            aux = this.tail;
            for (int i = this.size - 1; i > indice; i--) {
                aux = aux.getPreviousNode();
            }
        }
        return aux;
    } else {
        System.out.println("No existe el elemento en la lista");
        return null;
    }
}

// Metodo add: agrega un nuevo elemento al final de la lista enlazada
public void add(E dato) {
```

```
        Node<E> newNode = new Node<E>(dato);
        if (isEmpty()) {
            this.head = newNode;
            this.tail = newNode;
        } else {
            this.tail.setNextNode(newNode);
            newNode.setPreviousNode(tail);
            tail = newNode;
        }
        this.size++;
    }

    // Metodo remove: desenlaza al nodo del indice indicado
    public void remove(int indice) {
        if (indice < this.size && indice >= 0) {
            if (indice == 0) {
                this.head = this.head.getNextNode();
                this.head.getPreviousNode().setNextNode(null);
                this.head.setPreviousNode(null);
            } else {
                Node<E> current = this.get(indice);
                current.getNextNode().setPreviousNode(current.getPreviousNode());
                current.getPreviousNode().setNextNode(current.getNextNode());
                current.setPreviousNode(null);
                current.setNextNode(null);
            }
            this.size--;
        } else {
            System.out.println("No se encuentra el elemento");
        }
    }

    // Metodo toString: devuelve un String con los elementos de la lista enlazada
    //separada por ,
    @Override
    public String toString() {
        String str = "{";
        for (Node<E> aux = this.head; aux != null; aux = aux.getNextNode()) {
            str += aux.toString() + ", ";
        }
        return str + "}";
    }
}
```

Clase Test: Esta clase contiene los imports que se utilizaran para la ejecución del ejercicio y los métodos main, generarPeorCaso e InsercionSort. Implementados para ser usados con la clase DoubleLinkedList genérica.

Listing 8: Test

```
import com.panayotis.gnuplot.JavaPlot;
import java.io.*;
import java.util.*;

public class Test {
```

```
    public static void main(String[] args) throws IOException {
        int tamano = 500; // Tamano maximo del arreglo
        String archivoInsercion = "insercion.txt";
        PrintWriter oS = new PrintWriter(archivoInsercion);

        // Generar peores casos
        ArrayList<DoubleLinkedList<Integer>> casos = new ArrayList<>();
        for (int n = 1; n <= tamano; n++) {
            casos.add(generarPeorCaso(n));
        }

        // Ejecutar el algoritmo de ordenamiento en los peores casos
        for (DoubleLinkedList<Integer> caso : casos) {
            oS.println(String.valueOf(insertionSort(caso)));
            //insertionSort(caso);
        }
        oS.close();
        // plot "/Users/richarteq/eclipse-workspace/Algoritmica/insercion.txt"
            with lines
        JavaPlot p = new JavaPlot();
        p.addPlot("\"" + archivoInsercion + "\" with lines");
        p.plot();

//        // Imprimir los resultados
//        for (DoubleLinkedList<Integer> caso : casos) {
//            for (Integer data : caso) {
//                System.out.print(data + " ");
//            }
//            System.out.println();
//        }
    }

    public static long insertionSort(DoubleLinkedList<Integer> list) {
        if (list.size() <= 1) {
            return 0;
        }

        Node<Integer> current = list.getHead().getNextNode();
        long nano_startTime = System.nanoTime();

        while (current != null) {
            Integer key = current.getData();
            Node<Integer> previous = current.getPreviousNode();

            while (previous != null && previous.getData() > key) {
                previous.getNextNode().setData(previous.getData());
                previous = previous.getPreviousNode();
            }

            if (previous == null) {
                list.getHead().setData(key);
            } else {
                previous.getNextNode().setData(key);
            }

            current = current.getNextNode();
```

```
        }

        long nano_endTime = System.nanoTime();
        return nano_endTime - nano_startTime;
    }

    public static DoubleLinkedList<Integer> generarPeorCaso(int t) {
        DoubleLinkedList<Integer> lista = new DoubleLinkedList<>();
        for (int i = t; i >= 1; i--) {
            lista.add(i);
        }
        return lista;
    }
}
```

2. Equipos, materiales y temas utilizados

- Sistemas Operativos: Manjaro Linux x86_64 6.1.31-1, Windows 11 ...
- VIM 9.0.
- Visual Studio Code
- Git 2.40.1
- Netbeans

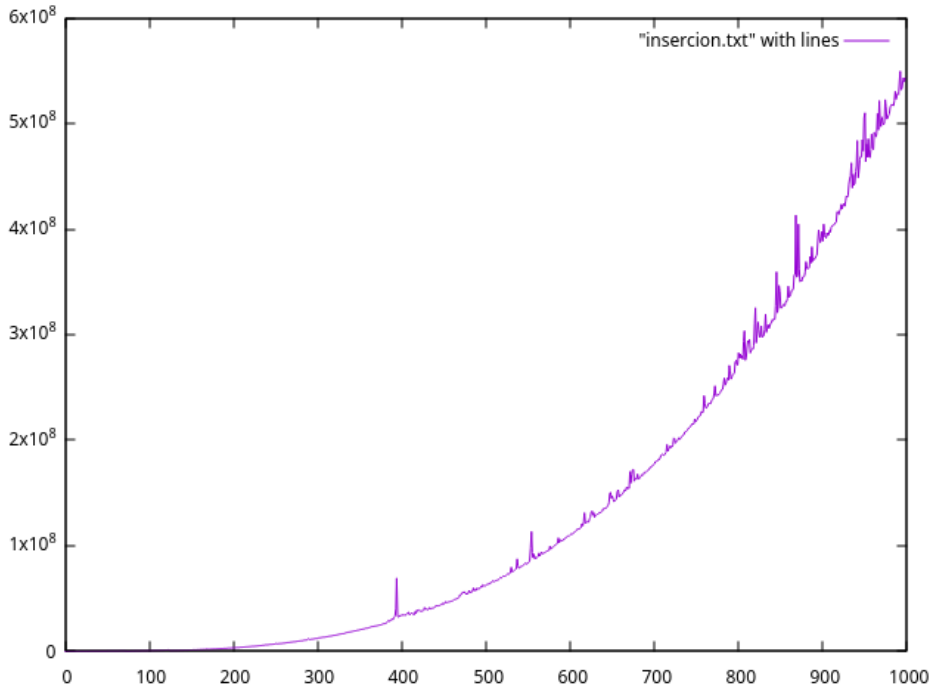
3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
  `https://github.com/DiegoNiS/Laboratorio-EDA-Colab.git`
- URL para el laboratorio 01 en el Repositorio GitHub.
  `https://github.com/DiegoNiS/Laboratorio-EDA-Colab/tree/main/lab04`

4. Actividades con el repositorio GitHub

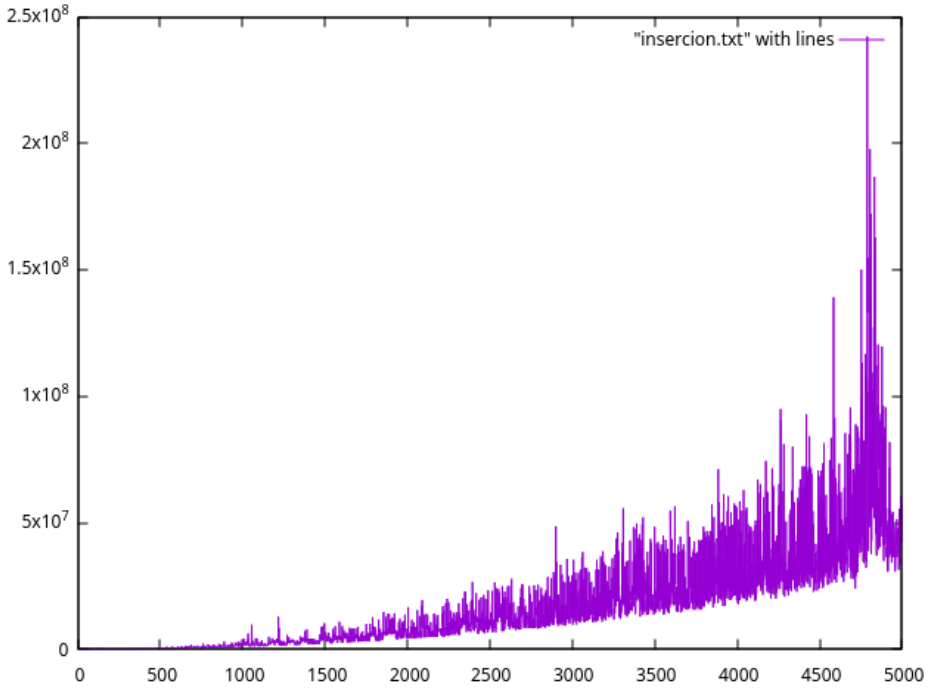
4.1. Implementación del Ejercicio 1

- Mostrando resultado del ejercicio 1,



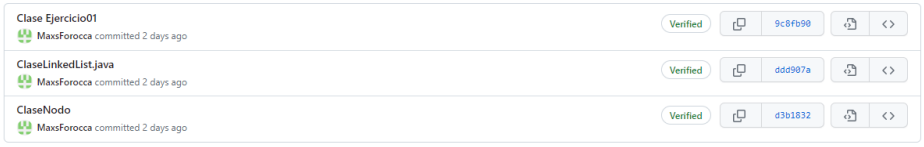
4.2. Implementación del Ejercicio 2

- Mostrando resultado del ejercicio 1,

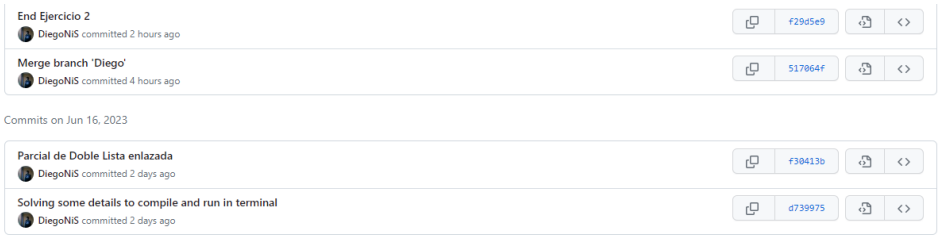


4.3. Commits

- Commit del ejercicio 1.



- Commit del ejercicio 2.



4.4. Estructura de laboratorio 04

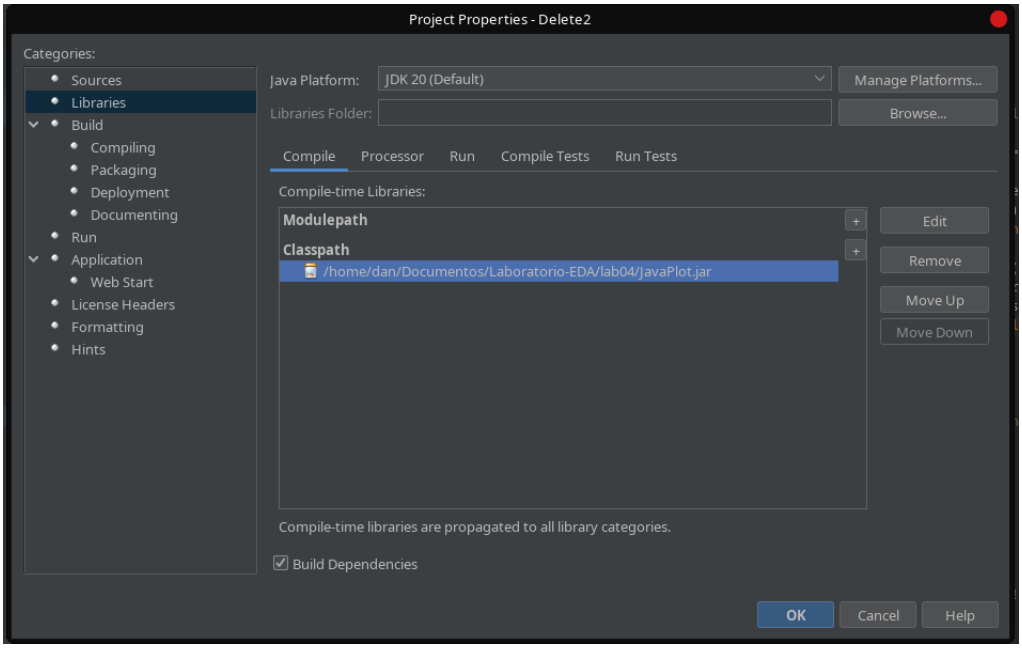
```
lab04/
|--- Ejercicio1
   |--- Ejercicio1.java
   |--- LinkedList.java
   |--- Nodo.java
```

```
|--- Ejercicio2
   |--- DoubleLinkedList.java
   |--- Test.java
   |--- Node.java
   |--- graph.PNG
|--- EjerciciopPrueba
   |--- DoubleLinkedList.java
   |--- insercion.txt
   |--- LinkedList.java
   |--- LinkedListTest.java
   |--- Node.java
   |--- Test.java
|--- latex
    |--- img
        |--- logo_abet.png
        |--- logo_episunsa.png
        |--- logo_unsa.jpg
        |--- CapEjercicio1.png
        |--- Commit1.png
        |--- commit2.png
        |--- graph.png
        |--- net.png
    |--- .gitignore
    |--- Laboratorio-eda-grupo6.pdf
    |--- Laboratorio-eda-grupo6.tex
    |--- actividades.tex
    |--- cabecera.tex
    |--- caratula.tex
    |--- github.tex
    |--- materiales.tex
    |--- preguntas.tex
    |--- referencia.tex
    |--- rubrica.tex
    |--- tarea.tex

|---JavaPlot
|---README
```

5. Cuestionario

- **¿Cómo se ejecutaría sus implementaciones desde terminal(consola)? Por ejemplo en el IDE Netbeans se agrega un jar externo así: ¿Cómo lo haría desde la terminal?**



- Si queremos correrlo desde terminal, por ejemplo, ubicandonos en el ejercicio 2 debemos especificar el classpath al momento de compilarlo y ejecutarlo de la siguiente forma, (No olvidar que en este caso el .jar es `JavaPlot.jar`).

Listing 9: caption

```
Laboratorio EDA:(main) $
Laboratorio EDA:(main) $ javac -cp lab04/JavaPlot.jar *.java
Laboratorio EDA:(main) $ java -cp lab04/JavaPlot.jar:. Test
```

6. Referencias

- `https://www.w3schools.com/`
- `https://www.geeksforgeeks.org/`
- `https://stackoverflow.com`
- `https://www.you.com`