Prácticas 1

7a.

```
package ejercicio7;

import java.util.ArrayList;

public class TestArrayList {
    public static void main(String[] args) {
        ArrayList<Integer> numeros = new ArrayList<>();

        for (int i = 0; i < args.length; i++) {
            numeros.add(Integer.parseInt(args[i]));
        }
        System.out.println(numeros);
    }

}
```

7b.

Si en lugar de un ArrayList hubiera utilizado el Linked List, habrian varias diferencias y mi programa hubiera sido mas lento, ya que en un ArrayList los accesos por indice son mucho más rápidos en comparación con el LinkedList. Pero si lo vemos desde la perspectiva de insertar o eliminar numeros a la lista de numeros entonces si uso LinkedList habria sido más rápido ya que insertar/eliminar es más eficiente con LinkedList que con Array List

7c.

For tradicional:

```
for (int i = 0; i < numeros.size(); i++) {
    System.out.println(numeros.get(i));
}
```

For-each:

```
for (Integer num : numeros) {
    System.out.println(num);
}
```

Streams:

```
numeros.forEach(numero → System.out.printlnt(numero));
```

ListIterator También existe

7d.

```
package ejercicio7;

import java.util.ArrayList;

public class TestArrayList {

    // Podrido de escribir todo el rato System.out.println xdd
    public static void Impr(String x) {
        System.out.println(x);
    }

    public static void ImprimirLista(ArrayList<Estudiante> x) {
        for (Estudiante e : x) {
            System.out.println(e);
        }
    }
```

```
//    public Estudiante(String ape, String nom, String gmail, String facu) {

   public static void gestionarEstudiantes() {

      ArrayList<Estudiante> estudiantes = new ArrayList<>();

      Estudiante es1 = new
            Estudiante("garcia","rodrigo","adorh@gmail.com", "UNLP");
      Estudiante es2 = new
            Estudiante("maria","hermandad","2356fdsg@gmail.com", "UNLP");

      Estudiante es3 = new
            Estudiante("lautaro","hermandad","93jf@gmail.com", "UNLP");

      estudiantes.add(es1);
      estudiantes.add(es2);
      estudiantes.add(es3);

      ArrayList<Estudiante> estudiantes_copia = estudiantes;

      Impr("estudiantes original" + "\n");
      ImprimirLista(estudiantes);
      Impr("estudiantes copia" + "\n");
      ImprimirLista(estudiantes_copia);



      es1.setApellido("charly");

      System.out.println("Checkpoint para la impresion devuelta con cambios"
+ "\n");

      Impr("estudiantes original" + "\n");
      ImprimirLista(estudiantes);
      Impr("estudiantes copia" + "\n");
      ImprimirLista(estudiantes_copia);
```

```
    }

    public static void main(String[] args) {

        gestionarEstudiantes();
    }
}
```

Preguntas:

- Vuelva a imprimir el contenido de la lista original y el contenido de la nueva lista.
  ¿Qué conclusiones obtiene a partir de lo realizado?

Respuesta: La conclusión que puedo sacar es:

Debido a que al generar la "nueva lista" se realizó una copia por referencia (copia superficial), ambas listas (la original y la copia) apuntan al mismo objeto en memoria y, por ende, contienen las mismas referencias a los objetos `Estudiante`. De ese modo, cualquier cambio realizado en un objeto a través de una de las listas (por ejemplo, modificar el apellido de un estudiante) se refleja en ambas listas.

- ¿Cuántas formas de copiar una lista existen? ¿Qué diferencias existen entre ellas?

Respuesta: Estas son las diferentes formas de copiar una lista:

- **Copia superficial:** Se copian las referencias a los objetos de la lista original; así, ambas listas comparten los mismos objetos.
- **Copia profunda:** Se crean nuevos objetos replicando la información de los objetos originales, de manera que las dos listas sean completamente independientes.

En este ejercicio, al asignar

```
ArrayList<Estudiante> estudiantes_copia = estudiantes;
```

Se realizó una copia superficial, lo que explica por qué los cambios en la lista original (o en los objetos que contiene) también se reflejan en la lista "copiada".

7f.

He creado una clase diferente llamada “Lista invertida” y agregué dos métodos, los cuales uno es recursivo y otro no:

```
package ejercicio7;

import java.util.ArrayList;

public class ListaInvertida {
    // Este es el método no recursivo
    public static boolean esCapicua(ArrayList<Integer> numeros) {
        int i = 0;
        int j = numeros.size() - 1;

        while (i < j) {

            if (!numeros.get(i).equals(numeros.get(j))) {
                return false;
            }
            i++;
            j--;
        }

        return true;
    }
    // Este es el método recursivo
    public static boolean esCapicuaConRecursion(ArrayList<Integer> numeros,
int inicio, int fin) {

        if (inicio >= fin) {
```

```
            return true;
        }
        if (!numeros.get(inicio).equals(numeros.get(fin))) {
            return false;
        }
        return esCapicuaConRecursion(numeros, inicio+1, fin - 1);
    }


    public static void main(String[] args) {
        ArrayList<Integer> lista1 = new ArrayList<>();
        lista1.add(1);
        lista1.add(2);
        lista1.add(3);
        lista1.add(2);
        lista1.add(1);

        ArrayList<Integer> lista2 = new ArrayList<>();
        lista2.add(1);
        lista2.add(2);
        lista2.add(3);



        System.out.println("Lista1 es capicua (Metodo no recursivo): " + esCapic
ua(lista1));
        System.out.println("Lista2 es capicua (Metodo recursivo): " + esCapicua
ConRecursion(lista2, 0, lista2.size() - 1));
    }
}
```

7g.

```
package ejercicio7;

import java.util.ArrayList;
```

```
import java.util.List;

public class EjercicioSucesion {

    public static boolean esPar(int n) {
        if (n % 2 == 0) return true;
        else return false;
    }

    public List<Integer> calcularSucesion (int n) {
        List<Integer> sucesion = new ArrayList<>();

        calcularRecursion(n, sucesion);

        return sucesion;
    }

    public void calcularRecursion(int n, List<Integer> sucesion) {
        sucesion.add(n);

        if (n == 1) return;

        if (esPar(n)) {
            calcularRecursion(n / 2, sucesion);
        } else {
            calcularRecursion(n * 3 + 1, sucesion);
        }

    }

}
```

7h.

```
package ejercicio7;

import java.util.ArrayList;
import java.util.List;

public class InvertirUnArray {


    public static void invertirArrayList(List<Integer> lista, int inicio, int fin) {
        if (inicio > fin) return;


        int temp = lista.get(inicio);
        lista.set(inicio, lista.get(fin));
        lista.set(fin, temp);

        invertirArrayList(lista, inicio +1, fin -1);
    }

    public static void main(String[] args) {
        List<Integer> lista = new ArrayList<>(List.of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10));
        System.out.println("Orden normal de la Lista: " + lista);
        invertirArrayList(lista, 0, lista.size() -1);

        System.out.println("Lista invertida: " + lista);


    }
}
```

7i.

```
package ejercicio7;

import java.util.LinkedList;
import java.util.List;

public class SumaElementos {



    public static int sumarLinkedList(LinkedList<Integer> lista) {
        if (lista.isEmpty()) return 0;
        else {
            int primero = lista.removeFirst();
            return primero + sumarLinkedList(lista);
        }
    }


    public static void main(String[] args) {
        LinkedList<Integer> lista = new LinkedList<>(List.of(1, 2, 3));
        System.out.println("Orden normal de la Lista: " + lista);

        System.out.println(sumarLinkedList(lista));
    }
}
```

7j.

```
package ejercicio7;

import java.util.ArrayList;
import java.util.List;
```

```
public class ListaOrdenada2Elementos {



    public static ArrayList<Integer> combinarOrdenado(ArrayList<Integer> lista
1, ArrayList<Integer> lista2) {
        ArrayList<Integer> resultado = new ArrayList<>();


        int i = 0;
        int j = 0;

        while (i < lista1.size() && j < lista2.size()) {
            if (lista1.get(i) <= lista2.get(j)) {
                resultado.add(lista1.get(i));
                i++;
            } else {
                resultado.add(lista2.get(j));
                j++;
            }
        }

        while (i < lista1.size()) {
            resultado.add(lista1.get(i));
            i++;
        }

        while (j < lista2.size()) {
            resultado.add(lista2.get(j));
            j++;
        }

        return resultado;
    }
```

```
    public static void main(String[] args) {
        ArrayList<Integer> lista1 = new ArrayList<>(List.of(1, 2, 7, 10, 14, 45));
        ArrayList<Integer> lista2 = new ArrayList<>(List.of(8, 9, 12, 14, 18));

        ArrayList<Integer> resultado = combinarOrdenado(lista1, lista2);

        System.out.println("lista ordenada: " + resultado);
    }
}
```

8.

```
package ejercicio8;

import java.util.ArrayList;
import java.util.List;
import java.util.NoSuchElementException;

public class Queue<T> {

    protected List<T> data;

    public Queue() {
        data = new ArrayList<>();
    }

    public void enqueue(T dato) {
        data.add(dato);
    }

    public T dequeue() {
        if (isEmpty()) throw new NoSuchElementException("La cola está vacia");
        else return data.remove(0);
    }
```

```
    public T head() {
        if (isEmpty()) throw new NoSuchElementException("cola vacia");
        else return data.get(0);
    }

    public int size() {
        return data.size();
    }

    public boolean isEmpty() {
        return data.isEmpty();
    }

    public String toString() {
        return data.toString();
    }


}
```

```
package ejercicio8;

public class CircularQueue<T> extends Queue<T> {

    public T shift() {
        if (isEmpty()) {
            throw new java.util.NoSuchElementException("La cola esta vacia");
        }
        T first = data.remove(0);
        data.add(first);
        return first;
    }
```

```
}
```

```
package ejercicio8;

public class DoubleEndedQueue<T> extends Queue<T> {

    public void enqueueFirst(T dato) {
        data.add(0, dato);
    }
}
```

9.

```
package ej9;

import java.util.Stack;

public class TestBalanceo {

    public static boolean estaBalanceado(String cadena) {
        Stack<Character> pila = new Stack<>();

        for (char character : cadena.toCharArray()) {
            if (esApertura(character)) {
                pila.push(character);
            } else if (esCierre(character)) {
                if (pila.isEmpty()) return false;

                char apertura = pila.pop();

                if (!coinciden(apertura, character)) {
                    return false;
                }
```

```
            }
        }

        return pila.isEmpty();

    }


    private static boolean coinciden(char apertura, char cierre) {
        return (apertura == '(' && cierre == ')') ||
               (apertura == '[' && cierre == ']') ||
               (apertura == '{' && cierre == '}');
    }

    private static boolean esApertura(char caracter) {
        return (caracter == '(' || caracter == '[' || caracter == '{');
    }

    private static boolean esCierre(char caracter) {
        return (caracter == ')' || caracter == ']' || caracter == '}');

    }

    public static void main(String[] args) {
        String prueba1 = "{( )[ ( ) ]}";
        String prueba2 = "( [ ) ]";

        System.out.println("Prueba 1 balanceado: " + estaBalanceado(prueba1));
// true
        System.out.println("Prueba 2 balanceado: " + estaBalanceado(prueba2));
// false
    }
}
```