

# Práctica: Listas

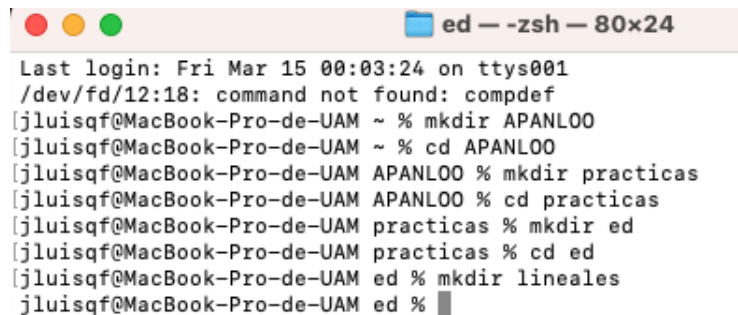
José Luis Quiroz Fabián

Restricciones;

1. Celulares frente a ustedes pantalla con abajo de la mesa.
2. No tener audífonos.
3. Abrir su navegador en la página del API de Java.
4. No tener abierta otra página sin pre-autorización.

## 1 Generación de la estructura del proyecto

Usando la terminal generar un conjunto de carpetas para trabajar en nuestro proyecto (ver Figura 1). Se debe crear la carpeta APANLOO → practicas → ed → lineales.



```
ed — -zsh — 80x24
Last login: Fri Mar 15 00:03:24 on ttys001
/dev/fd/12:18: command not found: compdef
[jluisqf@MacBook-Pro-de-UAM ~ % mkdir APANLOO
[jluisqf@MacBook-Pro-de-UAM ~ % cd APANLOO
[jluisqf@MacBook-Pro-de-UAM APANLOO % mkdir practicas
[jluisqf@MacBook-Pro-de-UAM APANLOO % cd practicas
[jluisqf@MacBook-Pro-de-UAM practicas % mkdir ed
[jluisqf@MacBook-Pro-de-UAM practicas % cd ed
[jluisqf@MacBook-Pro-de-UAM ed % mkdir lineales
[jluisqf@MacBook-Pro-de-UAM ed % █
```

Figure 1: Generación de carpetas

## 2 Generación de los fuentes

Usando nano, vi o vim, en la carpeta **lineales** crear los archivos **Lista.java** y **ArrayList.java**. Observe que el **package practicas.ed.lineales** hace referencia a la carpeta donde se tienen los archivos.

---

```
package practicas.ed.lineales;
```

```

public interface Lista<E> {

    /**
     * Agrega un elemento al final de la coleccion
     * @param e Elemento que se agregara a la coleccion
     */
    public void agregar(E e);
    /**
     * Agrega un elemento al inicio de la coleccion
     * @param e Elemento que se agregara a la coleccion
     */
    public void agregarInicio(E e);
    /**
     * Agrega un elemento al final de la coleccion
     * @param e Elemento que se agregara a la coleccion
     */
    public void agregarFinal(E e);
    /**
     * Agrega un elemento en una determinada posicion
     * @param e Elemento que se agregara a la coleccion
     * @param posicion La posicion del elemento que se va agregar
     */
    public void agregarPosicion(E e,int posicion);
    /**
     * Elimina un elemento del final de la coleccion
     * @param e Elemento que se eliminara la coleccion
     */
    public E eliminar();
    /**
     * Elimina un elemento del inicio de la coleccion
     * @param e Elemento que se eliminara la coleccion
     */
    public E eliminarElementoInicio();
    /**
     * Elimina un elemento del final de la coleccion
     * @param e Elemento que se eliminara la coleccion
     */
    public E eliminarElementoFinal();
    /**
     * Elimina un elemento de un determinada posicion
     * @param posicion La posicion del elemento a eliminar
     */
    public E eliminarElementoPosicion(int posicion);

    /**
     * Regresa verdadero si la coleccion es vacia
     */
    public boolean esVacia();
}

```

```

    /**
     * Regresa el numero de elementos de la coleccion
     */
    public int numElementos();

    /**
     * Elimina todos los elementos de la lista
     */
    public void limpiarLista();

    /**
     * Regresa la coleccion como un arreglo
     */
    public E[] convertirArreglo();

    /**
     * Regresa el elemento en una posicion particular
     * @param posicion Posicion del elemento a regresar
     */
    public E consultar(int posicion);
}

```

---

```

package practicas.ed.lineales;

public class ArrayList<E> implements Lista<E>{

    private static final int MAX=5;
    private int indice=0;
    private Object[] datos=null;

    public ArrayList() {

        this(MAX);

    }

    public ArrayList(int tam) {

        if(tam<0){
            throw new IllegalArgumentException();
        }
        datos = new Object[tam];

    }
    private void asegurarGC(){

        for(int i=0;i<datos.length;i++){

```

```

        datos[i]=null;
    }
}

@Override
public void agregar(E e) {
    Object[] aux=null;
    if(indice==datos.length){

        aux = new Object[datos.length+datos.length/2];
        System.arraycopy(datos,0,aux,0,datos.length);
        //DEBEMOS ELIMINAR TODAS LAS REFERENCIAS...DEJAR LOS
        OBJETOS EN EL "LIMBO"
        asegurarGC();
        datos = aux;

    }
    datos[indice] = e;
    indice++;
}

@Override
public void agregarInicio(E e) {
    // TODO Auto-generated method stub
    throw new UnsupportedOperationException("Unimplemented method
'agregarInicio'");
}

@Override
public void agregarFinal(E e) {
    // TODO Auto-generated method stub
    throw new UnsupportedOperationException("Unimplemented method
'agregarFinal'");
}

@Override
public void agregarPosicion(E e, int posicion) {
    // TODO Auto-generated method stub
    throw new UnsupportedOperationException("Unimplemented method
'agregarPosicion'");
}

@Override
public E eliminar() {
    // TODO Auto-generated method stub
    throw new UnsupportedOperationException("Unimplemented method
'eliminarElemento'");
}

@Override
public E eliminarElementoInicio() {

```

```

        // TODO Auto-generated method stub
        throw new UnsupportedOperationException("Unimplemented method
        'eliminarElementoInicio'");
    }

    @Override
    public E eliminarElementoFinal() {
        // TODO Auto-generated method stub
        throw new UnsupportedOperationException("Unimplemented method
        'eliminarElementoFinal'");
    }

    @Override
    public E eliminarElementoPosicion(int posicion) {
        // TODO Auto-generated method stub
        throw new UnsupportedOperationException("Unimplemented method
        'eliminarElementoPosicion'");
    }

    @Override
    public boolean esVacia() {
        return indice==0;
    }

    @Override
    public int numElementos() {
        return indice;
    }

    @Override
    public void limpiarLista() {
        // TODO Auto-generated method stub
        throw new UnsupportedOperationException("Unimplemented method
        'limpiarLista'");
    }

    @Override
    public E[] convertirArreglo() {
        // TODO Auto-generated method stub
        throw new UnsupportedOperationException("Unimplemented method
        'convertirArreglo'");
    }

    @SuppressWarnings("unchecked")
    @Override
    public E consultar(int posicion) {
        if(!(posicion<0 || posicion >= numElementos())){

            return (E)datos[posicion];

        }else{

```

```

        throw new IndexOutOfBoundsException();
    }
}
}

```

---

En la carpeta **ed** crear el archivo `App.java` el cual contiene el método **main**.

---

```

package practicas.ed;

import practicas.ed.lineales.ArrayList;

public class App {
    public static void main(String[] args){
        System.out.println("Programa de listas!");
        ArrayList<Integer> l1 = new ArrayList<Integer>();
        l1.agregarElemento(5);
        System.out.println(l1.consultar(0));
    }
}

```

---

Observe que esta clase utiliza la clase `ArrayList` que se encuentra en otra carpeta, por lo que debemos importarla mediante **`import practicas.ed.lineales.ArrayList;`**.

### 3 Compilación

Para compilar, regresar a la carpeta `APANLOO` y utilizar el compilador **javac**:

```
javac -d bin -sourcepath . practicas/ed/App.java
```

La opción **-d** seguida de un nombre de directorio le indica al compilador dónde colocar los archivos **.class** generados después de la compilación. En este caso, **bin** es el directorio de destino. Si el directorio **bin** no existe, **javac** intentará crearlo. Usar un directorio separado como **bin** para los archivos compilados ayuda a mantener organizado el espacio de trabajo, separando el código fuente de los archivos binarios.

La opción **-sourcepath** especifica la ruta de directorio donde el compilador buscará archivos de código fuente **.java** para compilar. El **.** después de **-sourcepath** indica el directorio actual como la ruta de búsqueda de los archivos fuente. Esto significa que el compilador buscará en el directorio actual y todos sus subdirectorios para resolver las referencias a otros archivos de código fuente necesarios para la compilación.

La Figura 2 muestra la generación de los archivos binarios **\*.class**.

```
jlluisqf@MacBook-Pro-de-UAM APANLOO % javac -d bin -sourcepath . practicas/ed/App.java
jlluisqf@MacBook-Pro-de-UAM APANLOO %
jlluisqf@MacBook-Pro-de-UAM APANLOO % ls
bin                practicas
jlluisqf@MacBook-Pro-de-UAM APANLOO % cd bin
jlluisqf@MacBook-Pro-de-UAM bin % ls
practicas
jlluisqf@MacBook-Pro-de-UAM bin % cd practicas
jlluisqf@MacBook-Pro-de-UAM practicas % ls
ed
jlluisqf@MacBook-Pro-de-UAM practicas % cd ed
jlluisqf@MacBook-Pro-de-UAM ed % ls
App.class          lineales
jlluisqf@MacBook-Pro-de-UAM ed % cd lineales
jlluisqf@MacBook-Pro-de-UAM lineales % ls
ArrayList.class    Lista.class
jlluisqf@MacBook-Pro-de-UAM lineales % █
```

Figure 2: Generación de binarios class

## 4 Ejecución

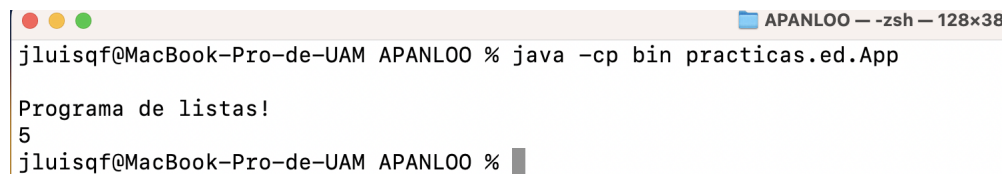
Para ejecutar, regresar a la carpeta APANLOO y utilizar el compilador **javac**:

**java -cp bin practicas.ed.App**

La opción **-cp** (o **-classpath**) especifica la lista de directorios, archivos **JAR**, y archivos **ZIP** que contienen las clases y paquetes necesarios para ejecutar la aplicación. En este caso, **bin** es el directorio que se proporciona como **classpath**, lo que significa que le estás diciendo a la JVM que busque las clases compiladas dentro del directorio **bin**.

El parámetro **uam.apanloo.App** indica que hay una clase llamada **App** en el paquete **uam.apanloo**. La **JVM** buscará esta clase dentro del **classpath** especificado (en este caso, el directorio **bin**), cargará la clase, e intentará ejecutar el método **main** definido en ella.

La Figura 3 muestra la ejecución del proyecto.



```
jlluisqf@MacBook-Pro-de-UAM APANLOO % java -cp bin practicas.ed.App
Programa de listas!
5
jlluisqf@MacBook-Pro-de-UAM APANLOO % █
```

Figure 3: Ejecución de la clase principal.

## 5 Ejercicios

- Implementar todos los métodos de **ArrayList**.
- Repetir los pasos ahora con **LinkedList**.