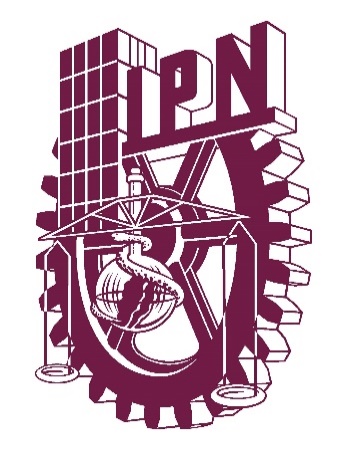
**INSTITUTO POLITÉCNICO NACIONAL**



*“Unidad Interdisciplinaria de Ingeniería Campus Zacatecas”*

*“****Programación Orientada a Objetos (POO)****”*

*List vs ArrayList vs LinkedList*

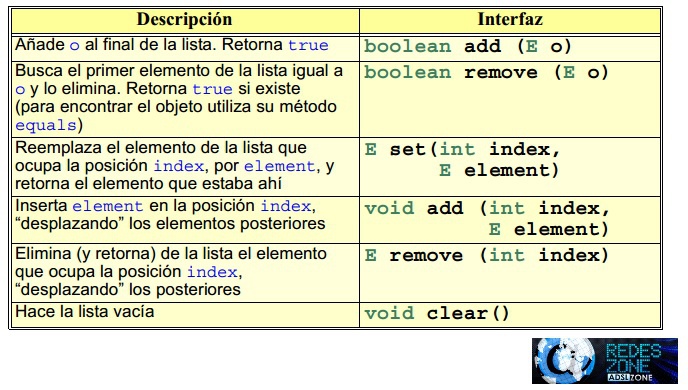
**Profesor: Cruz Lejía Roberto Oswaldo**

* **Alumno:****Jose de Jesus Arellano García**

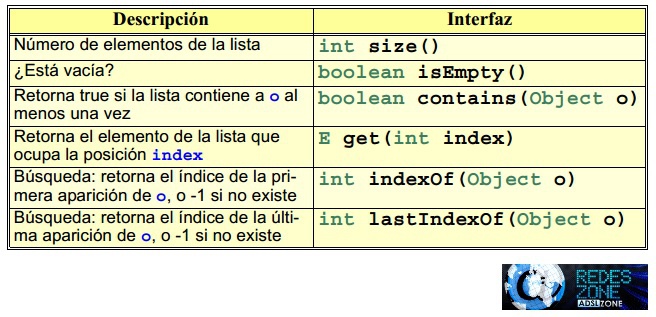
**2CM1**

*24 de 10 del 2019*

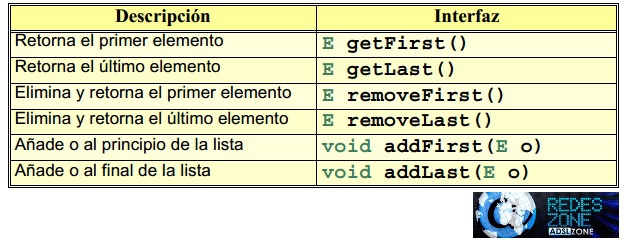
**List** es una colección, y una colección puede ser interfaces y clase abstracta que nos permite identificar los objetos independientemente de la implementación. Es decir, son genéricas.



**ArrayList** es parte del marco de recopilación y está presente en el paquete java.util. Nos proporciona matrices dinámicas en Java. Sin embargo, puede ser más lento que las matrices estándar, pero puede ser útil en programas donde se necesita mucha manipulación en la matriz.



**LinkedList** son estructuras de datos lineales donde los elementos no se almacenan en ubicaciones contiguas y cada elemento es un objeto separado con una parte de datos y una parte de dirección. Los elementos están vinculados mediante punteros y direcciones. Cada elemento se conoce como un nodo. Debido a la dinámica y la facilidad de las inserciones y eliminaciones, son preferibles a las matrices. También tiene algunas desventajas, ya que no se puede acceder directamente a los nodos, sino que debemos comenzar desde la cabeza y seguir a través del enlace para llegar a un nodo al que deseamos acceder.



**LinkedList**

| **Ventajas** | **Desventajas** |
| --- | --- |
| Añadir y remover elementos con un iterador | Uso de memoria adicional por las referencias a los elementos anterior y siguiente |
| Añadir y remover elementos al final de la lista | El acceso a los elementos depende del tamaño de la lista |

**ArrayList**

| **Ventajas** | **Desventajas** |
| --- | --- |
| Añadir elementos | Costos adicionales al añadir o remover elementos |
| Acceso a elementos | La cantidad de memoria considera la capacidad definida para el ArrayList, aunque no contenga elementos |