* **EXISTE UNAS FUNCIONES PARA HACER MINIMOS Y MAXIMOS. PERO PARA SI LAS UTILIZAS DEBES INICIALIZAR LA MINIMA AL MAXIMO VALOR EL LA MAIMO AL MINIMO VALOR. Estos son los valores min y maximo de kava resgistrado luego tienes que utilizar if**

**Integer.MIN\_VALUE**

**Integer.MAX\_VALUE**

* **Los systems con if**

|  |
| --- |
| **Sytem.println(texto aparte fijo que imprimimos + (Condicion ? lo que hace si se cumple : lo que hace si no se cumple));** |



* **LinkedList**

**Bueno para hacer eliminaciones en el centro de la lista**

Una LinkedList es una estructura de datos donde cada elemento es un nodo que contiene el dato y una referencia al siguiente nodo.

**Características:**

Tamaño dinámico: Crece o se reduce en función de las operaciones.

Ideal para inserciones y eliminaciones en medio de la lista (O(1) si se tiene la referencia al nodo).

Acceso secuencial: Obtener un elemento requiere recorrer la lista desde el primer nodo (O(n)).

**NOMENCLATURA**

|  |
| --- |
| **LinkedList<String> list = new LinkedList<>();** |

**Y para añadir**

|  |
| --- |
| **NombreDelLinkedList.add(dato añadir);** |

**Y para imprimir:**

|  |
| --- |
| **System.out.println(nombredelinkedlist.get(posicion));** |

**Y para eliminar:**

|  |
| --- |
| **nombredelinkedlist.remove(dato);** |

EJ:

**LinkedList<String> list = new LinkedList<>();**

**list.add("Java");**

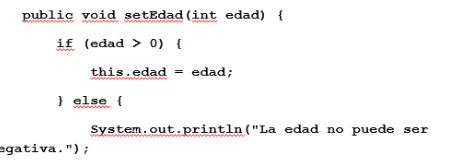
**list.add("Python");**

**System.out.println(list.get(1)); // Imprime Python**

**list.remove("Java"); // Elimina "Java"**

* **Encapsulamiento:**

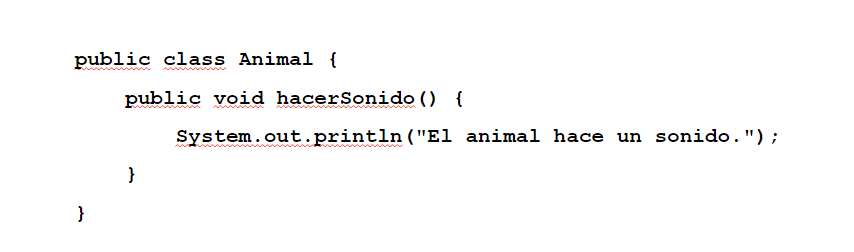
El **encapsulamiento** es una técnica que restringe el acceso directo a los atributos de una clase. En Java, se implementa declarando los atributos como private y proporcionando métodos públicos (**getters** y **setters**) para acceder y modificar esos atributos. TAMBIEN PARA CONTROLAR LOS DATOS. Tipo condiciones en el setter

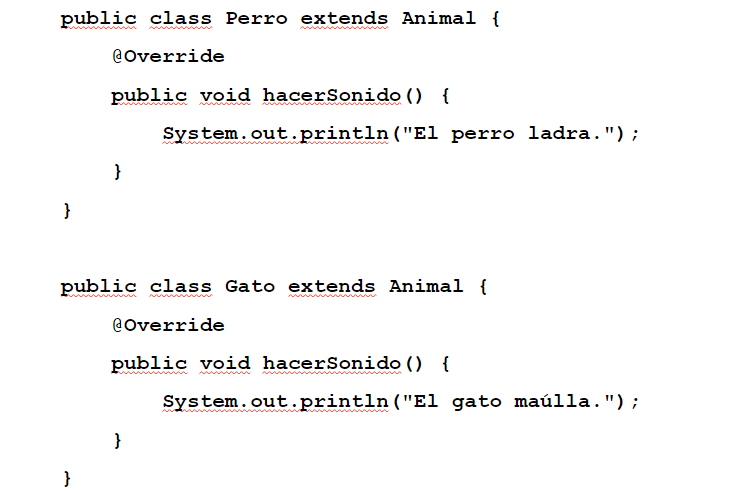


* **clase abstracta**

Una **clase abstracta** no puede ser instanciada directamente y es utilizada como una clase base para otras clases. Puede contener métodos abstractos (sin implementación) que las clases hijas deben sobrescribir. Obligatoriamente debe tener un metodo abstracto. Pero las clases abstractas son para que sean clases muy genericas y ahorrar codigo. Osea unicamente para ser clases madre extender una hija y ya en la hija pones todo el rollo

* El **polimorfismo** permite que una referencia de la clase padre se utilice para referenciar objetos de las clases hijas. En otras palabras, un objeto puede ser tratado como si fuera de su tipo base o de su tipo específico.Tu con esto puedes instanciar que un objeto de la clase padre sea un objeto de una de las hijas y te devolverá lo del método de la hija (si usas overriden)





* **Excepciones Personalizadas:**

Puedes crear tus propias excepciones personalizadas extendiendo la clase Exception.

|  |
| --- |
| **public class EdadInvalidaException extends Exception {**  **public EdadInvalidaException(String mensaje) {**  **super(mensaje);**  **}**  **}**  **public class Persona {**  **private int edad;**    **public void setEdad(int edad) throws EdadInvalidaException {**  **if (edad < 0) {**  **throw new EdadInvalidaException("La edad no puede ser negativa.");**  **}**  **this.edad = edad;**  **}**  **}** |