

Nombre

- Sebastian Oña

Clase01: 30 Abril del 2024

Visual Studio Code

Visual Studio Code es un software de Microsoft que edita códigos fuente disponibles para Windows, Linux y macOS. Este programa no necesita tanto espacio para su instalación a diferencia de Visual Studio. Es compatible con JavaScript y Node.js y extensiones a otros lenguajes como Python. **Características**

- **IntelliSense:** esta característica le ofrece al programador el autocompletado y resaltado de sintaxis, lo que lo hace más rápido a la hora de escribir un código. Proporciona sugerencias de códigos y terminaciones inteligentes con relaciones a variables y funciones.
- **Depuración:** esta herramienta ayuda a detectar errores en los códigos, para evitar revisar línea por línea, así como también detecta errores mínimos antes de ejecutar la depuración en sí.
- **Extensiones:** VS Code es un editor potente por las extensiones que maneja, las cuales permiten personalizar y agregar funciones adicionales de forma aislada y modular. Esto hace que sea más fácil programar en diferentes lenguajes, agregar nuevos temas al editor y conectar con otros servicios.

Comandos visual code

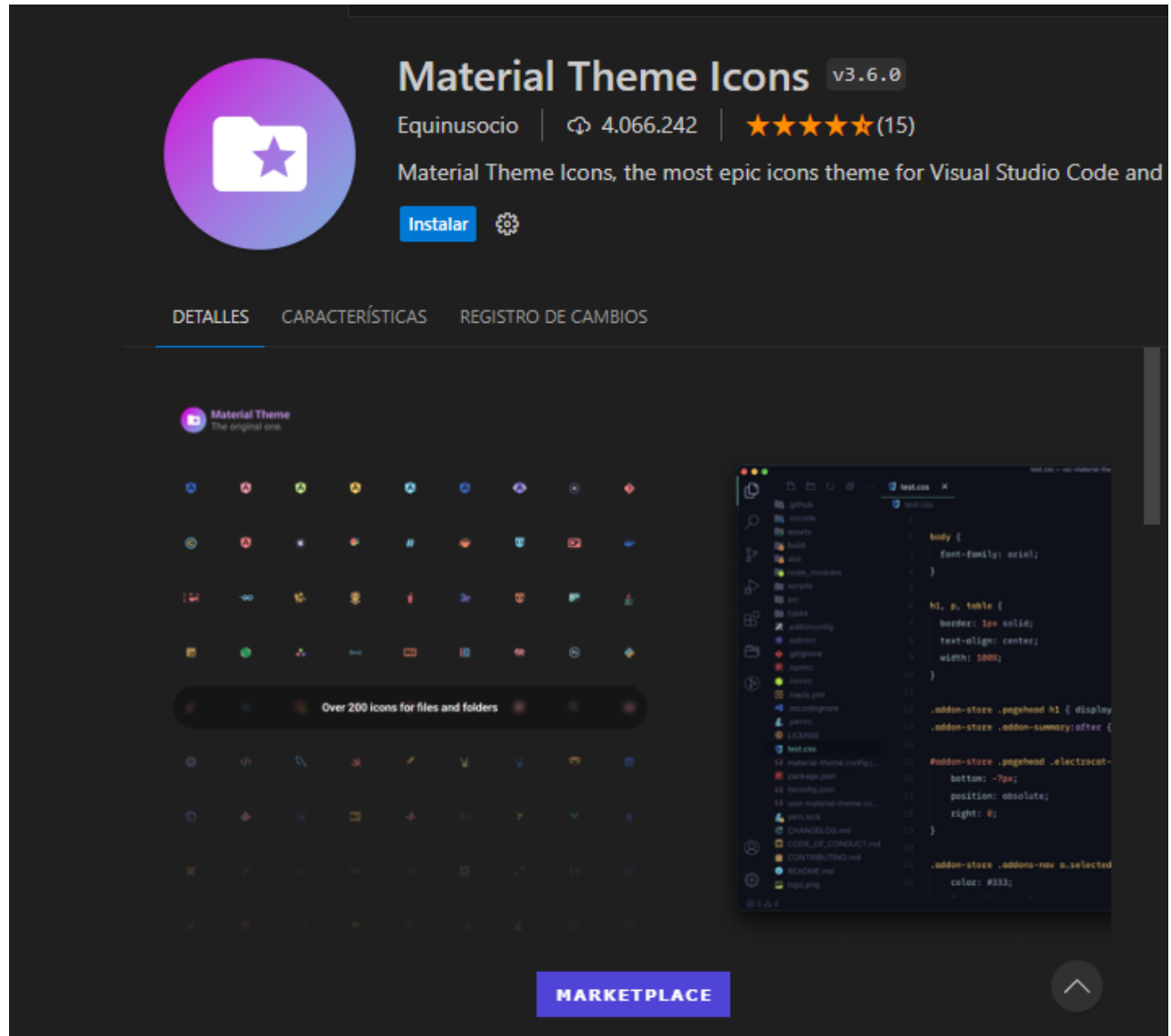
- **Windows:** ==CTRL+SHIFT+P== : Abre el panel de control que tiene VSC.
- **Windows:** ==CTRL+P== : Abre panel para cambiar de diferente archivo que tengas en tu carpeta.
- **Windows:** ==CTRL+B== : La pestaña que te muestra tus archivos.
- **Windows:** ==CTRL+D== : Te permite cambiar el nombre de varias palabras/variables del mismo nombre.
- **Windows:** ==CTRL+F== : Te ayuda a buscar palabra/variable en el código.
- **Windows:** ==CTRL+S== : Es la opción para guardar.
- **Windows:** ==ALT + or -== : Si está abierta pestañas te permite cambiar a otra.
- **Windows:** ==ALT + up or down== : Mueve toda la línea de acuerdo a la flecha.
- **Windows:** ==SHIFT+ALT+ up or down== : Copia toda la línea a la siguiente.
- **Windows:** ==SHIFT+ALT+A== : Pone para comentar una línea.
- **Windows:** ==CTRL+K+C== : Pone toda la línea como comentario.
- **Windows:** ==CTRL+T== : Permite mostrar todos los símbolos.
- **Windows:** ==CTRL+SPACE== : Te ayuda autocompletar una palabra.
- **Windows:** ==CTRL+x== : Borra una línea.

- **windows:** ==CLICK en el archivo + F2== : Cambia el nombre del archivo.

Personalizacion de nuestro entorno de trabajo

Visual estudio nos permite configurar nuestra inferzas o entorno de trabaja a nuestro gusto mediante una gran variedad de extensiones o opcion que posee el mismo visul studio code para esto veremos algunas extenciones que se pueden utilizar para esta personalizacion.

- **Extension para colocar icononos a cada uno de nuestros archivos**

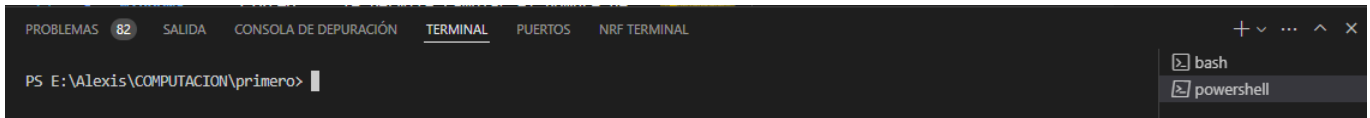


- Aqui podemos ver como quedaria una carpeta con archivos e imagenes con iconos diferente para ser identificados de forma facil

![] (Captura8.PNG)

Uso del terminal y tambien del "Bash"

El terminal o consola es el entorno donde trabajamos directamente con lineas de comando



Este es el terminal por defecto que nos otorga visual studio code el cual pertenece a windows, es decir aquí podremos solo ejecutar comando que pertenezcan a este sistema operativo el cual lleva por nombre powershell

Git Bash

Es una aplicación para entornos de Microsoft Windows que ofrece una capa de emulación para una experiencia de líneas de comandos de Git.

- **¿Qué es Bash?** Bash es el acrónimo en inglés de Bourne Again Shell.
- **¿Qué es Shell?** Es una aplicación de terminal que se utiliza como interfaz con un sistema operativo mediante comandos escritos.

Comandos de la terminal

- pwd : Me permite saber en que directorio estoy trabajando.
- touch : Para crear un archivo. Ej: `touch readme.md`
- code : Para crear archivos. Ej: `code readme.md`
- touch "" >> texto2.txt : Dentro de las comillas escribo lo que quiero que se escriba al crear el archivo --
> Ej: `touch "hola mundo" >> texto2.txt`

Git y Github



Git es un sistema de control de versiones rastrea los cambios que se han hecho en un conjunto de archivos es decir, un proyecto.

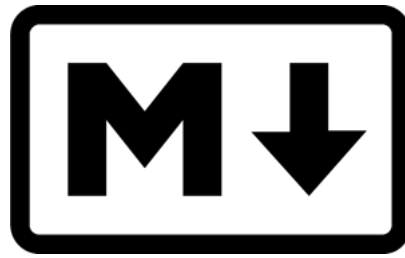
GitHub es un servicio que almacena repositorios en sus servidores y control de versiones usando git.

Recomendaciones de otros comandos que tiene markdown

- Si queremos enumerar hacemos lo siguiente:
 1. Línea 1
 2. Línea 2
 1. Línea 1
 2. línea 2
 3. Línea 3
- Si queremos poner otro signo para ideas hacemos lo siguiente:
 - Línea 1
 - Línea 2
 - Línea 1
 - línea 2
 - Línea 3

Clase03: 31 Abril del 2024

Markdown



Es un formato que nos permite generar archivos xml y html utilizando una sintáxis bastante clara y sencilla y sobre todo fácil de escribir.

- markdown (para convertir a pdf mi markdown):

1. Windows: CTRL+SHIFT+P
2. Despues busco la opcion de "markdown export pdf"
3. Se crea automaticamente un archivo pdf de tu markdown

- Para poner cursiva se debe encerrar entre asteriscos:

palabra o texto = *palabra o texto*

- Para poner en negrillas se debe encerrar entre doble asteriscos:

****palabra o texto**** = **palabra o texto**

- Para poner cursiva y negrillas se debe encerrar entre triple asteriscos:

******palabra o texto****** = ***palabra o texto***

- Windows: ALT+9+6 (') or windows: ALT+1+2+6 (~)

- Nos permite poner una parte del codigo para analizar despues de (```) tenemos que poner el lenguaje de progrmacion. Ejemplos: 1.

```
```\n java\n public int sumar(){\n   int i=10;\n }\n```\n\n2.\n\n```\n java\n public clas Hola{\n /* clase principal */\n   public class void main{\n     System.out.println();\n   }\n }\n```\n
```

- Para hacer cuadros:
  - Se organiza el texto en columnas y filas. Las filas se obtienen con salto de línea de texto y las columnas se obtienen encerrando el texto entre barras: |Texto|
  - Al finalizar la primera fila en donde tenemos los encabezados, se digita una siguiente fila de líneas entrecortadas para dar formato de título dentro de la tabla: |----|

```
|Columna 1 | columna 2 |
|-----|-----|
| A | B |
| C | D |
```

Para lo cual obtendremos:

Columna 1	columna 2
A	B
C	D

- Insertar link o enlace:
  - Si queremos solo insertar el hipervínculo al URL debemos encerrar entre <> la dirección.

```
<https://www.google.com>
```

De lo que obtenemos: <https://www.google.com>

- Si queremos aplicar un hipervínculo a un texto con una dirección, el texto se encierra entre corchetes, el enlace entre parentesis y si deseamos aplicar un mensaje sobre el texto cuando se pose el cursor, se encierra dicho mensaje entre comillas.

```
[Buscador](https://www.google.com "Google")
```

De lo que obtenemos: [Buscador](https://www.google.com)

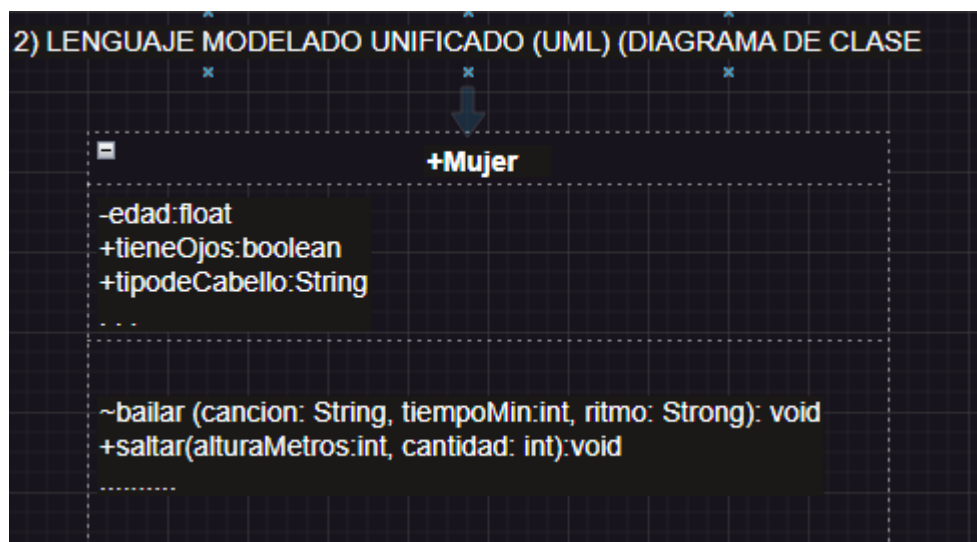
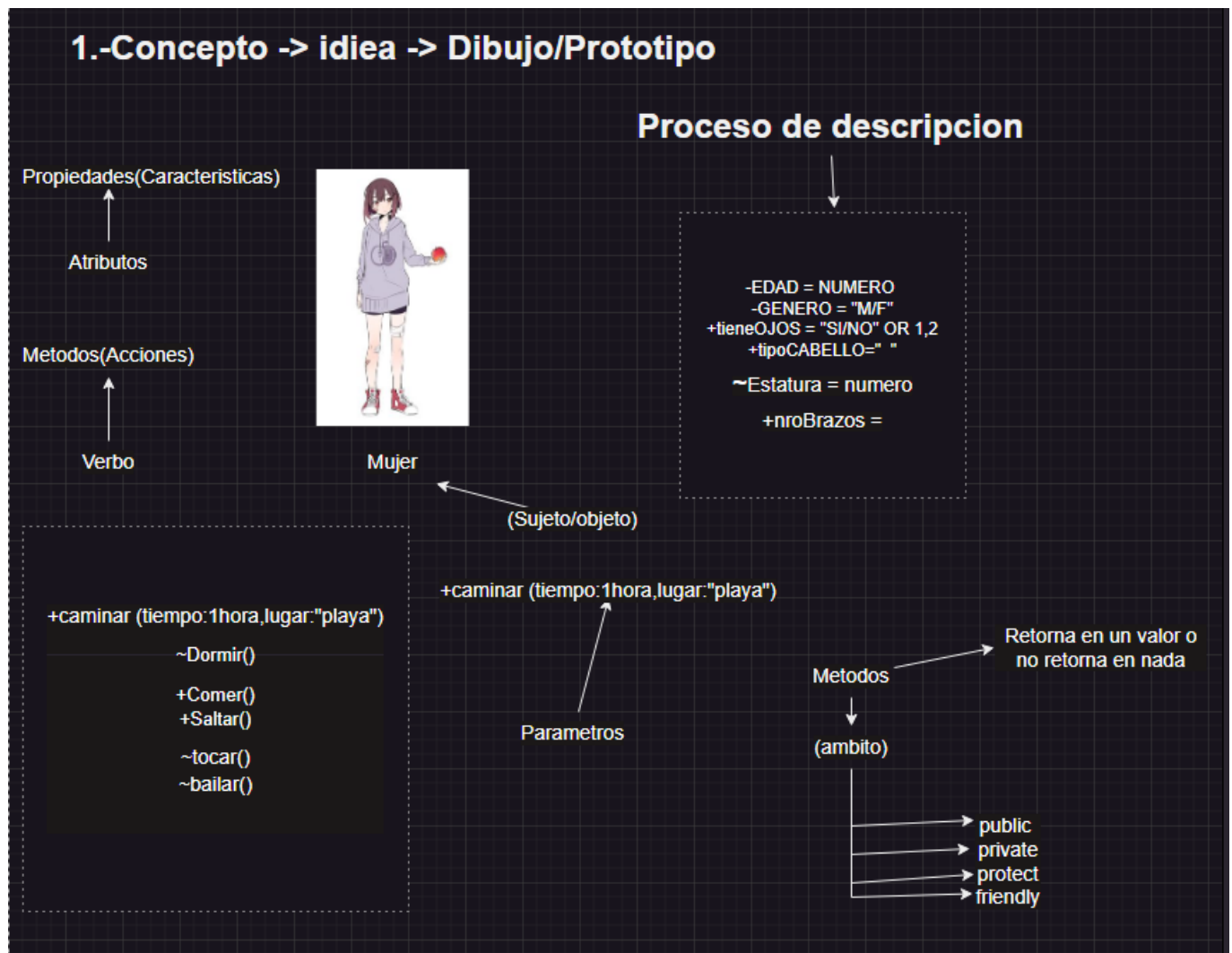
- Comandos de git
  1. git init : Crea un carpeta oculta en tu carpeta de proyecto y va aparecer tus archivo con "U".
  2. git status : Te muestra que archivos has guardado --> (rojo: no esta guardado and verde: esta guardado).
  3. git commit : Para guardar el proyecto.
  4. git checkout -b "Nombre de rama" : Crea y cambia de rama.
  5. git branch : Crea una rama.
  6. git push : Subo todo al github.

7. git pull : Nos ayuda a traer los cambios hechos en la nube.
8. git clone url : Baja el archivo de algun proyecto.
9. git ignore : Dentro del archivo puedes poner archivos q va ignorar o no va a guardar.
10. git config --global user.name "nombre" : Colocar nombre de usuario.
11. git config user.name : Verificar si esta su nombre de usuario. (**Es lo mismo para el email solo cambia user.email**)
12. git config --global init.default branch "main" : Cambiar de master a main.
13. git commit -m Descripcion :Agregar cambios al repositorio local y una descripcion del cambio.
14. git log :Muestra historial de commits. Si agregas "--oneline", aparece mas compacto.
15. git add. : Agregar cambios al commit.
16. git config --global core.editor : Asociar al vscode.
17. git branch -m nombre actual nombre nuevo : Cambiar nombre de rama sin estar en la rama a la que se va a cambiar.
18. git branch -d : Eliminar rama (locales no publicadas).
19. git merge nombre-de-rama : Fusionar rama con main.
20. git remote : Ver nombre del repositorio remoto.
21. git fetch origin : Crear una rama local con los cambios de la rama remota.
22. git push origin -d : Eliminar rama remota.

## Resumenes de las siguientes clases

---

P.O.O



## Ambitos

- **public:** Indica que los miembros (variables o métodos) son accesibles desde cualquier clase o paquete. Son visibles para todos.
- **protected:** Los miembros con esta visibilidad son accesibles dentro del mismo paquete o por subclases, incluso si están en paquetes diferentes.
- **private:** Limita el acceso a los miembros solo dentro de la misma clase. No son visibles ni accesibles desde ninguna otra clase.

## UML

Los diagramas UML (Lenguaje Unificado de Modelado) son una herramienta fundamental en la programación orientada a objetos, especialmente en Java. Permiten visualizar la estructura y el diseño de un sistema de software antes de codificarlo, lo que ayuda a planificar y comunicar ideas de manera clara entre los desarrolladores. UML apoya la abstracción y el diseño de clases, relaciones, patrones de diseño y otros aspectos de POO, facilitando así la comprensión y el desarrollo del software

### 3) Código

```
``` java
public class Mujer(){
    private float Mujer edad;
    public boolean tieneOjos;
    public String tipoCablello;

    protect String bailar( String cancion, int tiempoMin, String ritmo){
        return "Baile fenomenal";
    }
}
```
```

#### Resumen porque es importante P.O.O

- Es importante porque proporciona una estructura clara para los programas que facilita el desarrollo y mantenimiento del código, permite la reutilización de código y hace que el software sea más flexible y fácil de modificar. Además, la POO puede ayudar a manejar sistemas complejos al dividirlos en partes más pequeñas y manejables, conocidas como objetos

## Tipos de datos Importantes para progrmamar

---

### Strings

- `charAt(index)`: Obtener un caracter
- `length()`: Longitud de la cadena
- `substring(inicio, fin)`: Extraer parte de la cadena
- `indexOf(cadena)`: Buscar una cadena en otra
- `equalsIgnoreCase(cadena)`: Comparar cadenas sin distinciones entre mayusculas y minusculas

### Scanner

- `Scanner stdIn = new Scanner(System.in)`
- `nextInt()` : Leer enteros
- `nextDouble()` : Leer double
- `nextLine()` : Salto de linea



## Declaracion de Arrays

- `int [] arr ; // Sin asignar memoria`
- `arr=new int[5]; // Asignando memoria al array`
- `arr[i]` : Acceder a los elementos del array
- Operadores Aritmeticos
- `+`, `-`, `*`, `/` (Solo numeros)
- `%` (Modulo)
- `++` `--` (Operador de incremento / decremento)
- Metodos para trabajar con arrays
- `Arrays.toString(array);` Imprime el contenido del array como String
- `Arrays.sort(array);` Ordena el array
- `Arrays.binarySearch(array, elementoABuscar);` Devuelve la posicion donde se encuentra el elemento o -1 si no lo encuentra

## Listas

- List list; Es una lista genérica que puede contener cualquier tipo de dato
- `add(Object o):` Agrega un elemento a la lista
- `remove(Object o):` Elimina un elemento de la lista
- `contains(Object o):` Devuelve true si contiene el objeto y false en caso contrario
- `size():` Devuelve la cantidad de elementos en la lista
- `isEmpty():` Devuelve true si esta vacia y false en caso contrario
- `get(int index):` Devuelve el elemento en la posicion indicada por "index"
- `set(int index, Object o):` Coloca el elemento "o" en la posición "index"
- `indexOf(Object o):` Devuelve la primera posición donde se encuentra el elemento "o", devuelve -1 si no esta presente.
- `lastIndexOf(Object o):` Igual que anterior pero busca desde el final hacia atrás.

## Secuencias de escape

```
\t mover el cursor al siguiente tabulador
\n salto de línea (enter)
\r avanza a la primera columna en el renglón actual
\" imprime una literal que utiliza comilla doble
\' imprime una literal con comilla sencilla
\\ imprime una diagonal invertida
```

## Getter/Setter

---

los getters y setters son métodos que se utilizan para obtener y establecer el valor de las variables privadas de una clase, respectivamente. Esto se hace para proporcionar un control más seguro del acceso a las variables

```
public class Ejemplo {
 private int numero;

 // Getter
 public int getNumero() {
 return numero;
 }

 // Setter
 public void setNumero(int nuevoNumero) {
 numero = nuevoNumero;
 }
}
```

## Constructores

---

Los constructores en Java son métodos especiales que se utilizan para inicializar objetos. Cuando se crea un nuevo objeto, el constructor establece los valores iniciales de los atributos del objeto y puede realizar cualquier configuración o inicialización necesaria.

Las ventajas de tener constructores en Java incluyen:

1. **Inicialización Controlada:** Los constructores proporcionan un lugar centralizado para inicializar los atributos de un objeto, asegurando que el objeto esté en un estado válido desde el momento de su creación.
2. **Sobrecarga de Constructores:** Java permite la sobrecarga de constructores, lo que significa que puedes tener múltiples constructores con diferentes listas de parámetros, ofreciendo flexibilidad en la inicialización de objetos.
3. **Legibilidad del Código:** Los constructores hacen que el código sea más legible al proporcionar una clara indicación de cómo se deben crear e inicializar los objetos.

- Ejemplo de código:

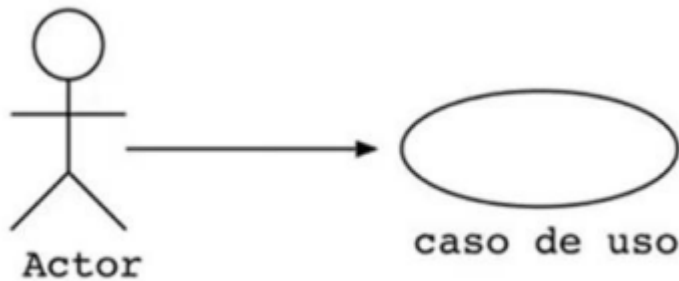
```
public class Coche {
 private String marca;
 private String modelo;
 private int año;

 // Constructor
 public Coche(String marca, String modelo, int año) {
 this.marca = marca;
 this.modelo = modelo;
 this.año = año;
 }
}
```

- como podemos ver al crear el contructor debemos poner public seguido del nombre de la clase y toma tres parámetros (marca, modelo y año) y los asigna a las variables de instancia del objeto

## Diagrama de caso de uso

---



### Componentes

actor .- actua con el sistema,estimula al sistema con algun evento o recibe informacion de sistema. Es un actor externo al sistema

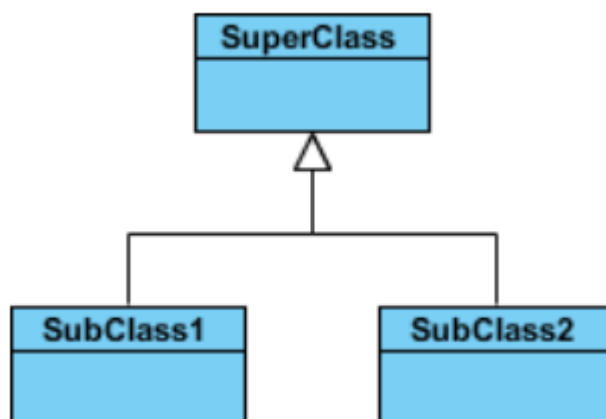
### relaciones

- extends.- se utiliza para representar una relación opcional o condicional entre dos casos de uso, donde un caso de uso extiende la funcionalidad de otro sin ser obligatorio
- include.- se usa para indicar que un caso de uso contiene la funcionalidad de otro caso de uso, lo que significa que siempre se ejecutará como parte del caso de uso que lo incluye. Ambas relaciones ayudan a simplificar los diagramas y a reutilizar comportamientos comunes entre diferentes casos de uso

## Herencia

---

La herencia en Java es un principio de la programación orientada a objetos que permite que una clase (llamada subclase o clase derivada) herede atributos y métodos de otra clase (llamada superclase o clase



base)

### Importancia

1. Reutilización de Código: Permite a las subclases utilizar métodos y variables de la superclase, evitando la duplicación de código
2. Organización Lógica: Facilita la creación de una jerarquía de clases que refleja relaciones del mundo real, lo que hace que el código sea más intuitivo y fácil de entender.
3. Polimorfismo: La herencia permite que objetos de diferentes subclases sean tratados como objetos de la superclase, lo que es útil para implementar operaciones generalizadas.

## Extends en Herencia

La palabra clave `extends` se utiliza en la declaración de una clase para indicar que esa clase va a heredar de otra clase. La herencia permite que la nueva clase, conocida como subclase, adquiera los atributos y métodos de la clase existente, llamada superclase. Esto facilita la reutilización de código y la creación de jerarquías de clases.

```
class Superclase {
 // Atributos y métodos de la superclase
}

class Subclase extends Superclase {
 // La Subclase hereda atributos y métodos de la Superclase
}
```

## @override

la anotación `@Override` se utiliza para indicar que el método de una subclase está sobrescribiendo un método de su superclase. Sirve varios propósitos:

1. Claridad: Hace explícito para los desarrolladores que el método está destinado a sobrescribir otro, lo que mejora la legibilidad del código.
2. Seguridad en la Compilación: Si el método sobrescrito no coincide con ningún método en la superclase, el compilador generará un error, previniendo posibles errores en tiempo de ejecución.
3. Mantenimiento del Código: Facilita el mantenimiento del código al asegurar que los cambios en las firmas de los métodos de la superclase se reflejen en las subclases.

```
class Superclase {
 public void mostrarMensaje() {
 System.out.println("Mensaje de la superclase");
 }
}

class Subclase extends Superclase {
 @Override
 public void mostrarMensaje() {
 System.out.println("Mensaje sobrescrito de la subclase");
 }
}
```

- Subclase está sobrescribiendo el método mostrarMensaje() de Superclase. La anotación @Override indica que este es el comportamiento deseado







# Package

Es una forma de organizar clases relacionadas o interfaces en grupos lógicos, lo que facilita su mantenimiento y organización. Los packages ayudan a evitar conflictos de nombres, ya que dos clases pueden tener el mismo nombre pero en diferentes packages. Además, los packages pueden controlar el acceso con modificadores de acceso como public, private, y protected

ventajas importantes:

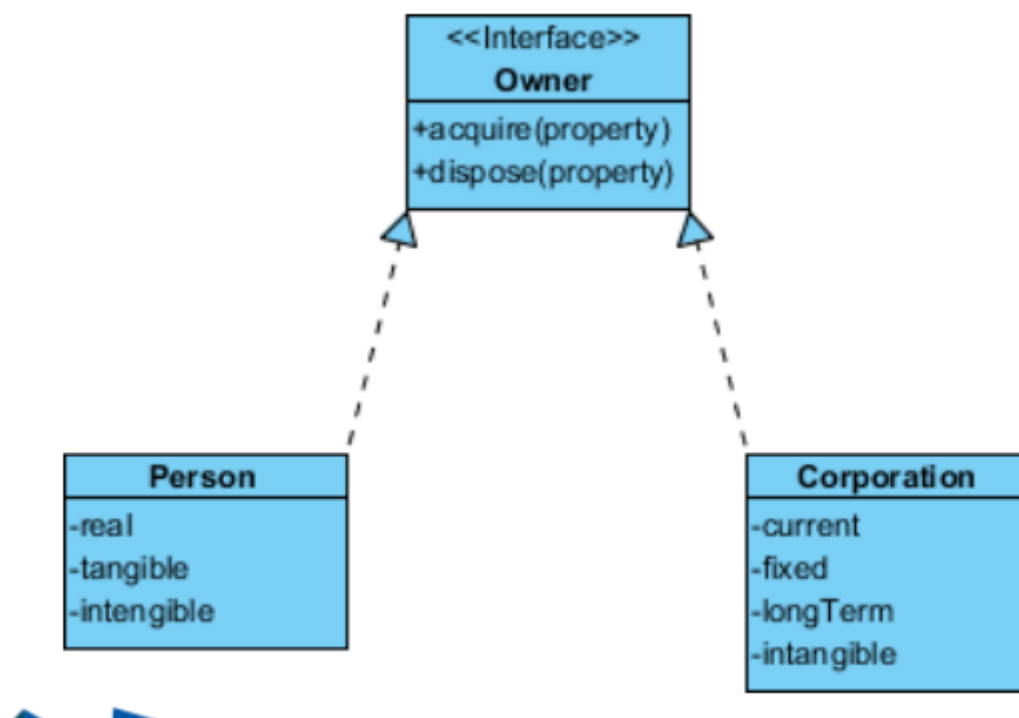
1. Organización: Permite clasificar las clases e interfaces por funcionalidad, lo que facilita la búsqueda y el uso de estas.
2. Evitar conflictos de nombres: Como cada package crea un nuevo espacio de nombres, puedes tener clases con el mismo nombre en diferentes packages sin que haya un conflicto.
3. Control de acceso: Los packages permiten restringir el acceso a clases y miembros de clases utilizando modificadores de acceso, lo que mejora la seguridad del código.
4. Reutilización: Los packages facilitan la reutilización de código al permitir que las clases sean fácilmente accesibles desde otros proyectos. Mantenimiento: Un sistema bien organizado en packages es más fácil de mantener y actualizar.

# Relaciones entre clases

| Class Diagram Relationship Type | Notation                                                                             |
|---------------------------------|--------------------------------------------------------------------------------------|
| Association                     |  |
| Inheritance                     |  |
| Realization/ Implementation     |  |
| Dependency                      |  |
| Aggregation                     |  |
| Composition                     |  |

# Realization/ Implementation

---



Una interfaz es una referencia de tipo que se utiliza para definir un contrato que las clases pueden implementar. Las interfaces sirven para:

1. Definir Métodos Abstractos: Una interfaz declara métodos sin implementarlos, dejando que las clases que la implementan definan el comportamiento concreto.
2. Separación de Contrato y Implementación: Permiten separar lo que se debe hacer de cómo se debe hacer, proporcionando una capa de abstracción.
3. Polimorfismo: Las interfaces permiten que diferentes clases sean tratadas como del mismo tipo si implementan la misma interfaz, facilitando el polimorfismo.

## 2do Bimestre

---

### Clase 1 : 18 de Junio del 2024

#### Herencia

#### Cardinalidad

La cardinalidad se refiere a la evaluación cuantitativa de las relaciones entre distintas entidades de datos dentro de un esquema o modelo. Específicamente, denota el número de ocurrencias de una entidad de datos en relación con otra entidad.

- En términos más simples, la cardinalidad describe cuántas instancias de una entidad de datos están relacionadas con otra entidad. Puede ser:

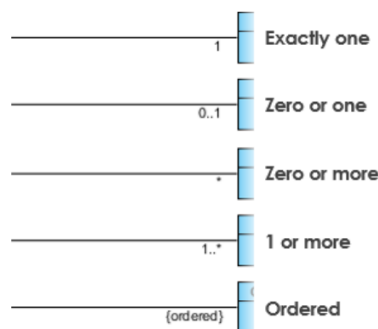
1. Uno a uno ("1..1"): Cada instancia de una entidad se relaciona con exactamente una instancia de otra entidad.

2. Uno a muchos("1..n" o "1..\*"): Cada instancia de una entidad se relaciona con una o más instancias de otra entidad.
3. Muchos a uno("n..1"): Varias instancias de una entidad se relacionan con una única instancia de otra entidad.
4. Muchos a muchos("n..m"): Varias instancias de una entidad se relacionan con varias instancias de otra entidad.

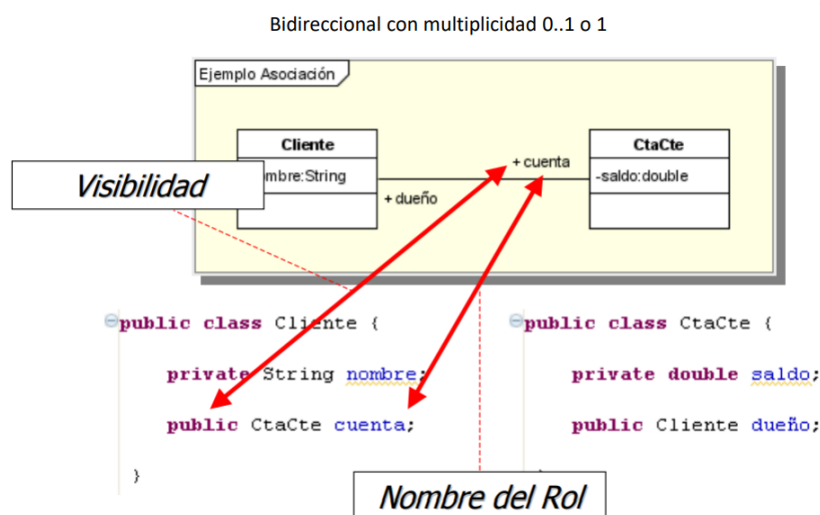
- Otros terminos:

- 1 --> sólo uno
- 0..1 --> cero o uno
- n --> Indica cuántas relaciones pueden haber.
- (\*) --> cero o más
- 0..\* --> cero o más (lo mismo que el anterior)

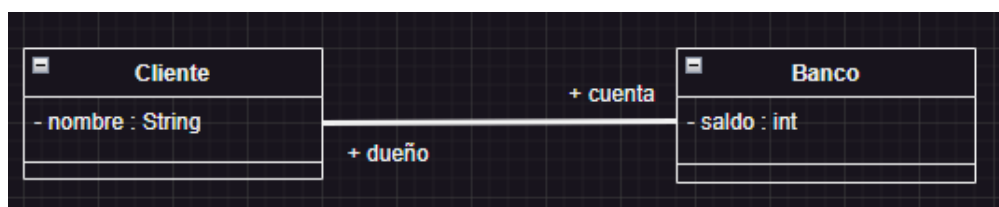
- Aquí una imagen como ejemplo:



## Asociacion (Cardinalidad)



- Bidireccional con cardinalidad 0..1 o 1



Como se veria en codigo:

```
public class Cliente {
 private String nombre;
 public Banco cuenta;
}

public class Banco {
 private int saldo;
 public Cliente dueño;
}
```

- Direccional con multiplicidad 0..1 o 1



Como se veria en codigo:

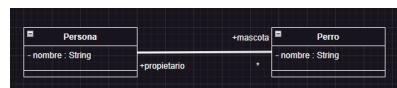
```
public class Usuario {
 private String nombre;
 public Clave clave;
}

public class Clave {
 private int codigo;
}
```

- Bidireccional con multiplicidad

- Ejemplos:

1. UML + Asociación (multiplicidad / cardinalidad : 0,1,+, 0..\*)



Como se veria en codigo:

```
public class Persona {
 private String nombre;
 public java.util.List<Perro> mastota = new ArrayList<Perro>();
}

public class Perro {
 private String nombre;
 public Persona propietario;
}
```

2. UML +Asociación (multiplicidad / cardinalidad : 0,1,+, 0..\*)



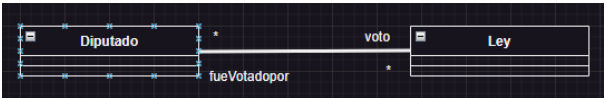
Como se veria en codigo:

```
public class Persona {
 private String nombre;
 public java.util.List<Perro> mastota = new ArrayList<Perro>();
}

public class Perro {
 private String nombre;
 public Persona propietario;
}
```

3. UML + Asociación (multiplicidad / cardinalidad : 0,1,+, 0..\*)



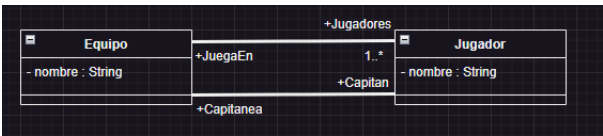


Como se veria en codigo:

```
1 import java.util.ArrayList;
2
3 You, 9 seconds ago | 1 author (You)
4 public class Diputado {
5 public java.util.List<Ley> voto = new ArrayList<Ley>();
6 }
7
8 You, 20 seconds ago | 1 author (You)
9 public class Ley {
10 public java.util.List<fueVotadoPor> voto = new ArrayList<fueVotadoPor>();
11 }
12 You, last week | 1 author (You)
```

- Relaciones

1. UML + Asociación (multiplicidad / cardinalidad : 0,1,+, 0.., 1..)



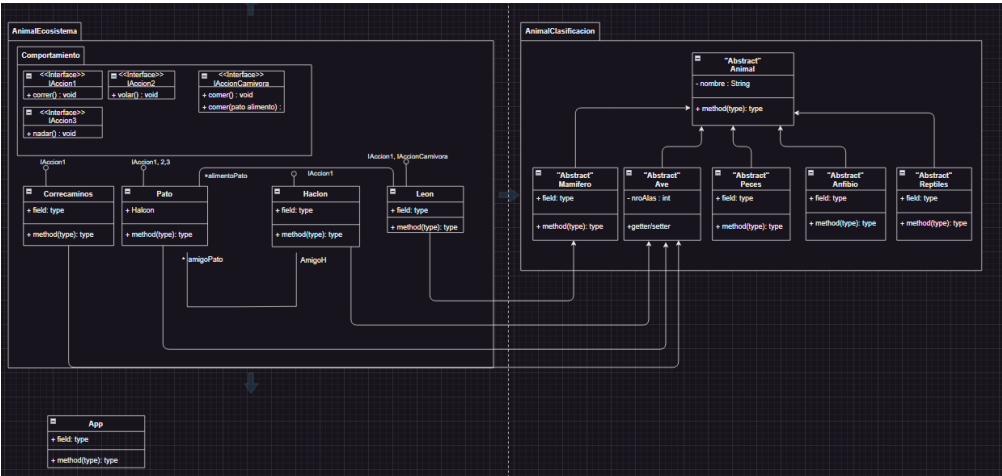
Como se veria en codigo:

```
public class Equipo {
 public Jugador capitane;
 public java.util.List<jugadores> voto = new ArrayList<jugadores>();
}

3 public class Jugador {
4 public Jugador capitanea;
5 public Jugador juegaEn;
6 }
```

Correccion del taller

En la siguiente imagen podemos apreciar como debio verse el diagrama de clase, del taller realizado la anterior clase y aumentado lo que vimos en la clase del dia de hoy:



Las siguientes imagenes vamos a ver como se veria en el visualcode y en codigo:

- Visual Code:



- Codigo

Una de las cosas importantes mientras realizabamos el diagrama de Clase, es entender en cual resultar conveniente poner una variable en esa clase, ya que si muchas hay diferentes clases puedan heredar esa variable y asi ahorrar lineas de codigos

```
package AnimalClasificacion;

You, 2 weeks ago | 1 author (You)
public abstract class Animal {
 private String nombre;

 public Animal(){
 this.nombre = "";
 }

 public String getNombre() {
 return nombre;
 }

 public void setNombre(String nombre) {
 this.nombre = nombre.toUpperCase();
 }
}
```

Entonces al momento que las demas clases como mamifero,ave,etc. Heredan esa variable y asi no tienes que crearla en cada una de ellas y si leon hereda de mamifero entonces leon tambien tendra la propiedad.

- Codigo de las clases que mandamos a llamar en la App principal

```
public class Pato extends Ave implements IAccion1,IAccion2,IAccion3{

 public Halcon amigoHalcon;

 public Pato(String nombre){
 setNombre(nombre);
 amigoHalcon = new Halcon(nombre:"Halconsillo");
 }

 @Override
 public String toString(){
 return "\n Clase: " + getClass().getName()
 + "\n nombre: " + getNombre()
 + "\n NroAlas: " + getNroAlas();
 }

 @Override
 public void nadar() {
 System.out.println(x:"Pato nadando");
 }

 @Override
 public void volar() {
 System.out.println(x:"Pato volando");
 }

 @Override
 public void correr() {
 System.out.println(x:"Pato esta corriendo.....");
 }
}

You, 2 weeks ago | 1 author (You)
public class Leon extends Mamifero implements IAccion1,IAccionCarnivora{

 public Pato alimentoPatuno;

 public Leon(String nombre){
 setNombre(nombre);
 alimentoPatuno = new Pato(nombre:"Lucas");
 }

 @Override
 public String toString(){
 return "\n Clase: " + getClass().getName()
 + "\n nombre: " + getNombre();
 }

 @Override
 public void comer() {
 System.out.println(x:"El león come ");
 }

 @Override
 public void comer(Pato alimentoPatuno) {
 System.out.println("Leon come pato : " + alimentoPatuno.toString());
 }

 @Override
 public void correr() {
 System.out.println(x:"leon corre");
 }
}
```

- En estas clases aparece algo interesante que es el toString()

El método toString() en Java es una función que devuelve una representación en cadena de texto de un objeto. Sirve para obtener información legible sobre el objeto y facilita su impresión o visualización en pantalla<sup>1</sup>. Aquí tienes algunos detalles importantes:

¿Qué es el método toString() en Java?

1. El método toString() se utiliza para convertir cualquier objeto Java en una cadena de texto.
2. Todos los objetos heredan este método de la clase Object.
3. Siempre que intentes imprimir o transformar a cadena un objeto, estarás utilizando implícitamente el método toString().

## Clase 2:

Repaso para la prueba

## Clase 3:

PRUEBA

## Clase 4:

Clase de 30 min

## Clase 5:

CLASE DE 1 HORA

## Clase 6:

Clase de 1 hora

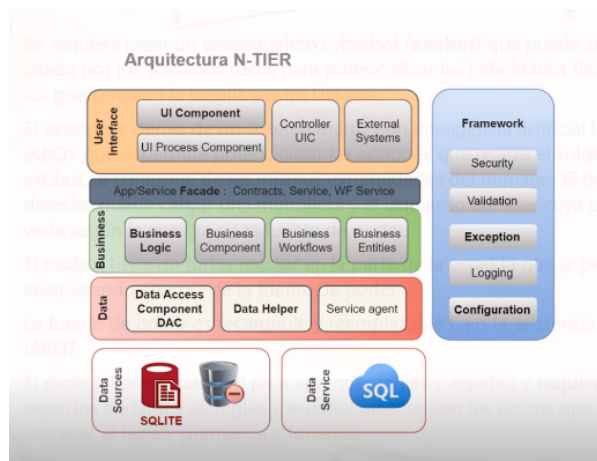
## Clase 7:

Taller Grupal(Aplicando lo aprendido lo visto en las tres ultimas Clases)

## Clase 8:

Correccion del diagrama de Clase

### Base de Datos



- Una característica es que base de datos se guarda en el disco duro (datos estático)/persistir -> tablas -> reglas
- P.O.O se guarda en la RAM (datos dinámicos) -> Clases(objetos)
- Atómico Significa mientras sea único

pongamos un ejemplo:

1. Persona(cedula(1),nombre1(1),nombre2(1),Apellido(1), TipoSangre(1), .....)

esto que quiere decir que cada dato la persona solo tiene un nombre1, un nombre 2 y asi sucesivamente entonces si pasa que uno de esos tienes mayor a uno entonces te podria dar error

- Reglas:

- 1....1
- 1....\*
- n....n

- Resolucion

1. 1.....1 (P.K)<---->(P.K)

```
|idP | Cedula | Nombre | Apellido| |idP|TipSangre|S| |----|-----|-----|-----| |---|-----|---| | 1 | 17547 |
Pepe | Andrade | |1 | A+ | | | 2 | 18734 | Ana | Soto | |2 | B- | | | 3 | 17372 | Maria | Perez | |3 | A | |
```

- Un consejo importante es elegir el (P.k) en este caso seria recomendable usar la columna idP porque no es una columna importante ya que al elgir la columna Cedula puede pasar que ingresas mal un numero y no despues va a ser muy dificil modificar ese numero y puede empezar a darte problemas en un futuro.

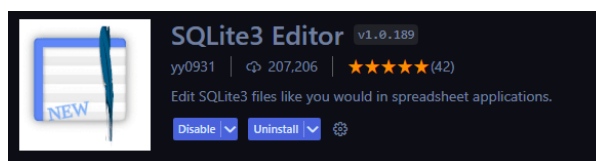
## Clase nro 9

Copiar los diagrmas de mi cuaderno a la compu

## Clase nro 10

Aplicamos lo visto en las anteriores dos clase de Base de Datos:

Primero es instalas la siguiente extencion:

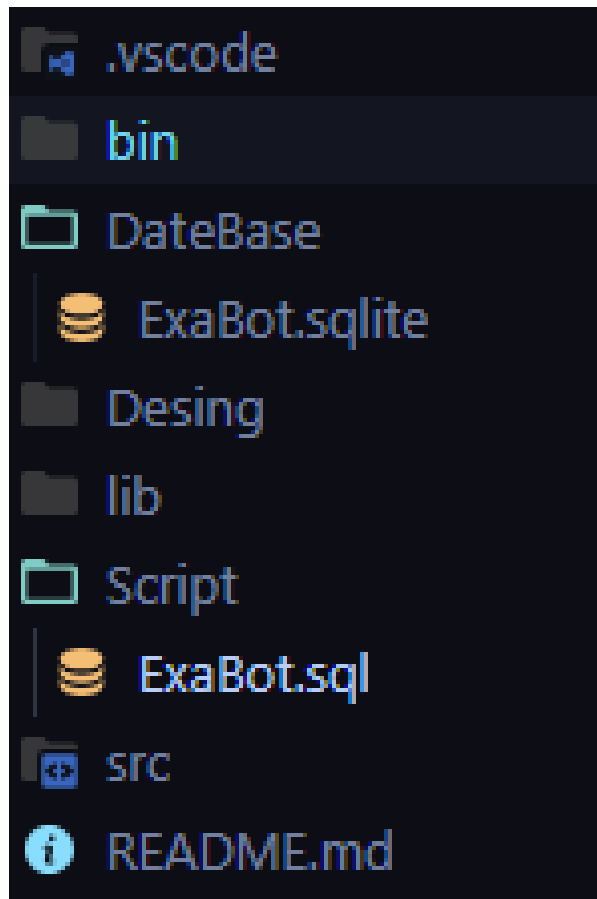


Entonces para este ejercicio vamos a crear un nuevo proyecto llamado EXABOT, despues creamos la carpeta desing, Script y Datebase

Script: Esta carpeta es donde vamos a crear con codigo nuestras tablas de base de datos por decirlo de una manera.

DateBase: Se va a conectar con el archivo de Scrpit para que se vea visualmente nuestra base de datos

En la siguiente imagen podemos ver como se debes crear los archivos



En esta clase el inge nos pase lo que ya tenia hecho de los Script:

```

-- Connect | [?] Connect and Open Panel
-- database: ../DataBase/ExaBot.sqlite
/*
 * Copyright
 * autor:
 * fecha:
 */

DROP TABLE IF EXISTS ExaBot;
DROP TABLE IF EXISTS IABot;
DROP TABLE IF EXISTS Persona;
DROP TABLE IF EXISTS PersonaTipo;

CREATE TABLE IABot (
 IdIABot INTEGER PRIMARY KEY autoincrement
 ,Nombre TEXT NOT NULL UNIQUE
 ,Observacion TEXT
 ,Estado VARCHAR(1) DEFAULT('A') CHECK (Estado IN ('A','X'))
 ,FechaCrea DATETIME DEFAULT current_timestamp
);

CREATE TABLE ExaBot (
 IdExaBot INTEGER PRIMARY KEY autoincrement
 ,IdIABot INTEGER NOT NULL
 ,Nombre TEXT NOT NULL
 ,Serie TEXT NOT NULL
 ,Estado VARCHAR(1) DEFAULT('A') CHECK (Estado IN ('A','X'))
 ,FechaCrea DATETIME DEFAULT current_timestamp
 ,CONSTRAINT fk_IABot FOREIGN KEY (IdIABot) REFERENCES IABot(IdIABot)
);

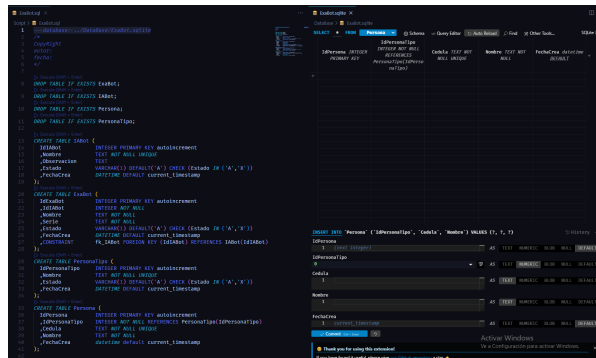
CREATE TABLE PersonaTipo (
 IdPersonaTipo INTEGER PRIMARY KEY autoincrement
 ,Nombre TEXT NOT NULL UNIQUE
 ,Estado VARCHAR(1) DEFAULT('A') CHECK (Estado IN ('A','X'))
 ,FechaCrea DATETIME DEFAULT current_timestamp
);

CREATE TABLE Persona (
 IdPersona INTEGER PRIMARY KEY autoincrement
 ,IdPersonaTipo INTEGER NOT NULL REFERENCES PersonaTipo(IdPersonaTipo)
 ,Cedula TEXT NOT NULL UNIQUE
 ,Nombre TEXT NOT NULL
 ,FechaCrea datetime default current_timestamp
);

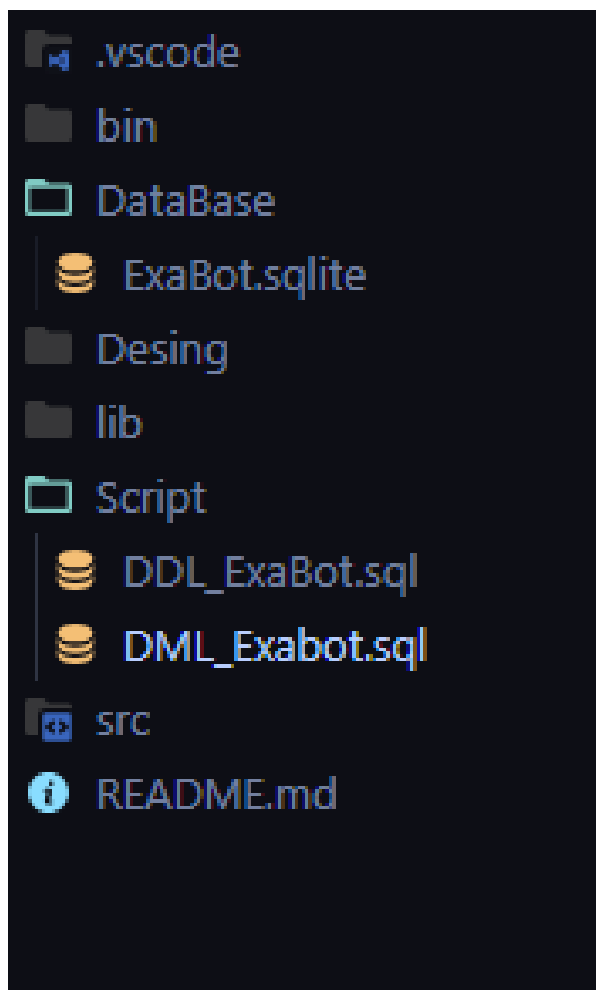
```

Punto importante a tener en cuenta es:

- Esta linea de codigo conecta al archivo ExaBot.sqlite para poder ver visualmente nuestras tablas que vamos creando en nuestra base de datos:
- Asi se iria viendo una vez mandemos a correr:

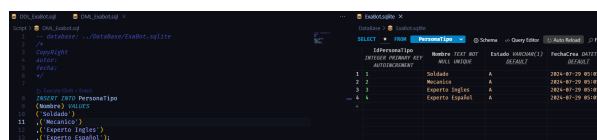


- La siguiente línea de código ayuda a borrar una tabla siempre y cuando exista, lo que ocurre es que al crear una tabla y puede pasar que te falte agregar algo de datos mas y mandas a correr te sale error, entonces la solución es borrar la tabla y volver a crearla y eso lo haces con este comando:
- Vamos a crear otro archivo en la carpeta Script para ingresar datos en la tabla



Partes importantes de códigos que nos ayudara mientras vamos llenando datos en nuestras tablas:

1. INSERT INTO NOMBRETABLEA: Nos va ayudar a ingresar datos en nuestras tablas creadas. Ejemplo:



2. SELECT \* FROM PersonaTipo; -> En lo que nos va a ayudar este código es buscar la tabla que queramos y hacé facilarnos ver que tipo de información hay que llenar en una tabla.

Ejemplo:

```

SELECT * FROM PersonaTipo

```

| IdPersonaTipo | Nombre          | Estado | FechaCreacion       |
|---------------|-----------------|--------|---------------------|
| 1             | Salvador        | A      | 2024-07-29 09:00:00 |
| 2             | Mecanico        | A      | 2024-07-29 09:00:00 |
| 3             | Experto Ingles  | A      | 2024-07-29 09:00:00 |
| 4             | Experto Espanol | A      | 2024-07-29 09:00:00 |

3. `SELECT count (*) FROM PersonaTipo;` -> Este en cuestion nos va a ayudar para saber cuantos datos tenemos en una tabla. Ejemplo:

```

SELECT count (*) FROM PersonaTipo;

```

| Count (*) |
|-----------|
| 4         |

4. `SELECT * FROM PersonaTipo WHERE IdPersonaTipo=2;` -> Este nos ayuda a encontrar datos en la tabla que hemos creado y el =2 puedes cambiarlo con ❤️ y te mostrara todo los datos menores que ese numero. Ejemplo:

```

SELECT * FROM PersonaTipo WHERE IdPersonaTipo=2;

```

| IdPersonaTipo | Nombre   | Estado | FechaCreacion       |
|---------------|----------|--------|---------------------|
| 2             | Mecanico | A      | 2024-07-29 09:00:00 |

o

```

SELECT * FROM PersonaTipo WHERE IdPersonaTipo<2;

```

| IdPersonaTipo | Nombre   | Estado | FechaCreacion       |
|---------------|----------|--------|---------------------|
| 1             | Salvador | A      | 2024-07-29 09:00:00 |

o

5. `SELECT Nombre FROM PersonaTipo WHERE Nombre like:`

- o "s%" -> si pones despues de like significa que buscara todas las palabras que comiencen con "s".

Ejemplo:

```

SELECT Nombre FROM PersonaTipo WHERE Nombre like "s%";

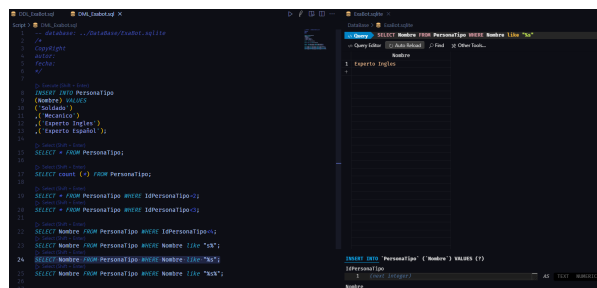
```

| Nombre   |
|----------|
| Salvador |

- o "%s" -> si pones despues de like significa que buscara todas las palabras que terminen con "s".

Ejemplo:





- The screenshot shows a SQL IDE with a script to insert data into the 'Personalipo' table. The script includes comments in Spanish and SQL commands for inserting data for various experts. The right pane shows the 'Execute' button and a list of tables in the database.

```

--> SQL_Instrucciones
--> Database: //Database/Exadot.sql
-->
--> Copyright:
-->
--> Author:
--> Fecha:
-->
--> Descripción:
INSERT INTO Personalipo
(Numero) VALUES
('Soldado'),
('Recaudador'),
('Experto Ingles'),
('Experto Español');
-->
--> Select:
SELECT * FROM Personalipo;
-->
--> Select(Show ->):
SELECT count (*) FROM Personalipo;
-->
--> Select:
SELECT * FROM Personalipo WHERE IdPersonalipo=1;
SELECT * FROM Personalipo WHERE IdPersonalipo=2;
-->
--> Select(Show ->):
SELECT Nombre FROM Personalipo WHERE IdPersonalipo=1;
SELECT Nombre FROM Personalipo WHERE Nombre like "S%";
SELECT Nombre FROM Personalipo WHERE Nombre like "N%";
SELECT Nombre FROM Personalipo WHERE Nombre like "N%";

```

On the right, the 'Database' pane shows a list of tables: 'Personalipo', 'Experto Ingles', 'Experto Español', 'Experto Ingles', 'Soldado', and 'Recaudador'. The 'Execute' button is highlighted.

- 
- The screenshot shows a SQL IDE with a query editor on the left and a results pane on the right. The query editor contains the following SQL code:
- ```

1 SELECT * FROM PERSONS
2 WHERE
3     (NAME LIKE 'C%' OR NAME LIKE 'M')
4     AND (AGE BETWEEN 20 AND 30)
5     AND (GENDER = 'M')
6     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
7     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
8     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
9     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
10    AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
11
12 SELECT * FROM PERSONS
13 WHERE
14     (NAME LIKE 'C%' OR NAME LIKE 'M')
15     AND (AGE BETWEEN 20 AND 30)
16     AND (GENDER = 'M')
17     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
18     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
19     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
20     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
21
22 SELECT * FROM PERSONS
23 WHERE
24     (NAME LIKE 'C%' OR NAME LIKE 'M')
25     AND (AGE BETWEEN 20 AND 30)
26     AND (GENDER = 'M')
27     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
28     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
29     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
30     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
31
32 SELECT * FROM PERSONS
33 WHERE
34     (NAME LIKE 'C%' OR NAME LIKE 'M')
35     AND (AGE BETWEEN 20 AND 30)
36     AND (GENDER = 'M')
37     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
38     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
39     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
40     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
41
42 SELECT * FROM PERSONS
43 WHERE
44     (NAME LIKE 'C%' OR NAME LIKE 'M')
45     AND (AGE BETWEEN 20 AND 30)
46     AND (GENDER = 'M')
47     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
48     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
49     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
50     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
51
52 SELECT * FROM PERSONS
53 WHERE
54     (NAME LIKE 'C%' OR NAME LIKE 'M')
55     AND (AGE BETWEEN 20 AND 30)
56     AND (GENDER = 'M')
57     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
58     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
59     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
60     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
61
62 SELECT * FROM PERSONS
63 WHERE
64     (NAME LIKE 'C%' OR NAME LIKE 'M')
65     AND (AGE BETWEEN 20 AND 30)
66     AND (GENDER = 'M')
67     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
68     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
69     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
70     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
71
72 SELECT * FROM PERSONS
73 WHERE
74     (NAME LIKE 'C%' OR NAME LIKE 'M')
75     AND (AGE BETWEEN 20 AND 30)
76     AND (GENDER = 'M')
77     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
78     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
79     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
80     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
81
82 SELECT * FROM PERSONS
83 WHERE
84     (NAME LIKE 'C%' OR NAME LIKE 'M')
85     AND (AGE BETWEEN 20 AND 30)
86     AND (GENDER = 'M')
87     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
88     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
89     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
90     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
91
92 SELECT * FROM PERSONS
93 WHERE
94     (NAME LIKE 'C%' OR NAME LIKE 'M')
95     AND (AGE BETWEEN 20 AND 30)
96     AND (GENDER = 'M')
97     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
98     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
99     AND (CITY LIKE 'M%' OR CITY LIKE 'S%')
100    AND (CITY LIKE 'M%' OR CITY LIKE 'S%')

```
- The results pane displays a table with the following data:
- | id | name | age | gender | city | email | phone | password | created_at | updated_at |
|----|----------------|-----|--------|---------------|----------------------------|------------|------------|---------------------|---------------------|
| 1 | John Doe | 25 | M | New York | john.doe@example.com | 1234567890 | 1234567890 | 2023-01-01 10:00:00 | 2023-01-01 10:00:00 |
| 2 | Jane Smith | 28 | F | Los Angeles | jane.smith@example.com | 9876543210 | 9876543210 | 2023-01-01 10:00:00 | 2023-01-01 10:00:00 |
| 3 | Mike Johnson | 22 | M | Chicago | mike.johnson@example.com | 5678901234 | 5678901234 | 2023-01-01 10:00:00 | 2023-01-01 10:00:00 |
| 4 | Sarah Brown | 27 | F | San Francisco | sarah.brown@example.com | 4321098765 | 4321098765 | 2023-01-01 10:00:00 | 2023-01-01 10:00:00 |
| 5 | David Wilson | 24 | M | Seattle | david.wilson@example.com | 3210987654 | 3210987654 | 2023-01-01 10:00:00 | 2023-01-01 10:00:00 |
| 6 | Emily Davis | 26 | F | Portland | emily.davis@example.com | 2109876543 | 2109876543 | 2023-01-01 10:00:00 | 2023-01-01 10:00:00 |
| 7 | Chris Miller | 23 | M | Denver | chris.miller@example.com | 1098765432 | 1098765432 | 2023-01-01 10:00:00 | 2023-01-01 10:00:00 |
| 8 | Alexander Lee | 29 | M | Phoenix | alexander.lee@example.com | 0987654321 | 0987654321 | 2023-01-01 10:00:00 | 2023-01-01 10:00:00 |
| 9 | Olivia White | 21 | F | San Diego | olivia.white@example.com | 8765432109 | 8765432109 | 2023-01-01 10:00:00 | 2023-01-01 10:00:00 |
| 10 | Benjamin Green | 26 | M | San Jose | benjamin.green@example.com | 7654321098 | 7654321098 | 2023-01-01 10:00:00 | 2023-01-01 10:00:00 |

- [illegible]

- 25 / 34

```

SQL_LabBotlog M SQL_LabBotlog M
Script SQL_LabBotlog
1 database: ...Database\LabBot.sql
2 /
3 Copyright
4 autor:
5 fecha:
6
7
8 SELECT PersonaTipo.Nombre ,Persona.Nombre
9 FROM PersonaTipo
10 JOIN Persona ON PersonaTipo.IdPersonaTipo = Persona.IdPersona

```

9. SELECT pt.Nombre 'Cargo' ,p.Nombre 'Nombres' FROM PersonaTipo pt JOIN Persona p on pt.IdPersonaTipo = p.IdPersona;

-> Este es la forma optimizada del anterior codigo y como pueden ver entre comillas podemos poner que nombre quieres q tenga la columna y tambien que puedes simplificar su nombre poniendo alado su simplificacion EJEMPLO:

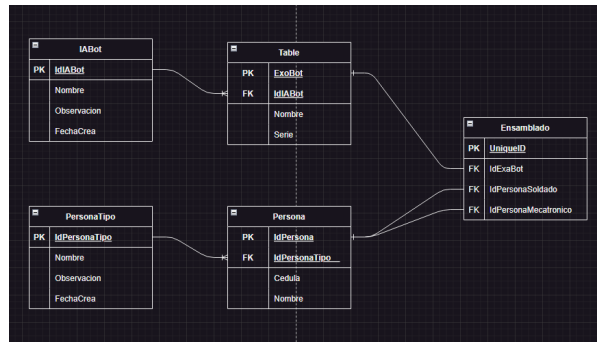
```

SQL_LabBotlog M SQL_LabBotlog M
Script SQL_LabBotlog
1 database: ...Database\LabBot.sql
2 /
3 Copyright
4 autor:
5 fecha:
6
7
8 SELECT pt.Nombre 'Cargo' ,p.Nombre 'Nombres'
9 FROM PersonaTipo pt
10 JOIN Persona p on pt.IdPersonaTipo = p.IdPersona

```

Clase nro 11

Diagrama MER:



DataHelper

1. Helper Objects en Java: En Java, los objetos auxiliares (helper objects) son clases que se utilizan para realizar operaciones comunes que son compartidas por varias clases. Estos objetos ayudan a mantener el código limpio y modular. Algunos ejemplos de uso de objetos auxiliares son:
 - Clases de utilidad (Utility Classes): Son clases con métodos estáticos que no tienen dependencias. Por ejemplo, una clase llamada IOUtils podría contener métodos estáticos para cerrar flujos de entrada/salida sin lanzar excepciones.
 - Clases de ayuda (Helper Classes): Estas clases tienen dependencias y, por lo tanto, pueden tener métodos no estáticos. Por ejemplo, una clase llamada PriceHelper podría calcular precios para diferentes productos o pedidos.
2. DataHelper en el contexto de bases de datos: En el contexto de acceso a datos y bases de datos, el término DataHelper puede referirse a un patrón de diseño que encapsula el manejo de la base de datos. Estos objetos ayudan a acceder a los datos de manera más organizada y modular. Por ejemplo, un DataHelper podría manejar la conexión a una base de datos y ejecutar consultas o transacciones.

DAO

1. El DAO es una clase o interfaz que encapsula la lógica de acceso a datos. Su objetivo principal es proporcionar una interfaz consistente y reutilizable para interactuar con la base de datos.

2. Un DAO generalmente ofrece operaciones CRUD para una entidad específica (por ejemplo, una tabla en la base de datos).
3. La implementación concreta del DAO se comunica con el DataHelper (como una conexión a la base de datos) para ejecutar consultas y transacciones.

CRUD

CRUD es un acrónimo que se utiliza en programación y sistemas de bases de datos para describir las cuatro operaciones básicas de almacenamiento persistente:

1. Crear (Create): Se refiere a la acción de agregar nuevos registros o entradas a una base de datos. Por ejemplo, crear un nuevo usuario en una aplicación.
2. Leer o Recuperar (Read o Retrieve): Implica obtener información existente de la base de datos. Por ejemplo, leer los detalles de un producto o mostrar una lista de usuarios.
3. Actualizar (Update): Significa modificar o cambiar los datos existentes en la base de datos. Por ejemplo, actualizar la dirección de un cliente o cambiar el precio de un artículo.
4. Borrar o Eliminar (Delete o Destroy): Se refiere a la acción de eliminar registros de la base de datos. Por ejemplo, eliminar una publicación de un blog o dar de baja a un cliente.

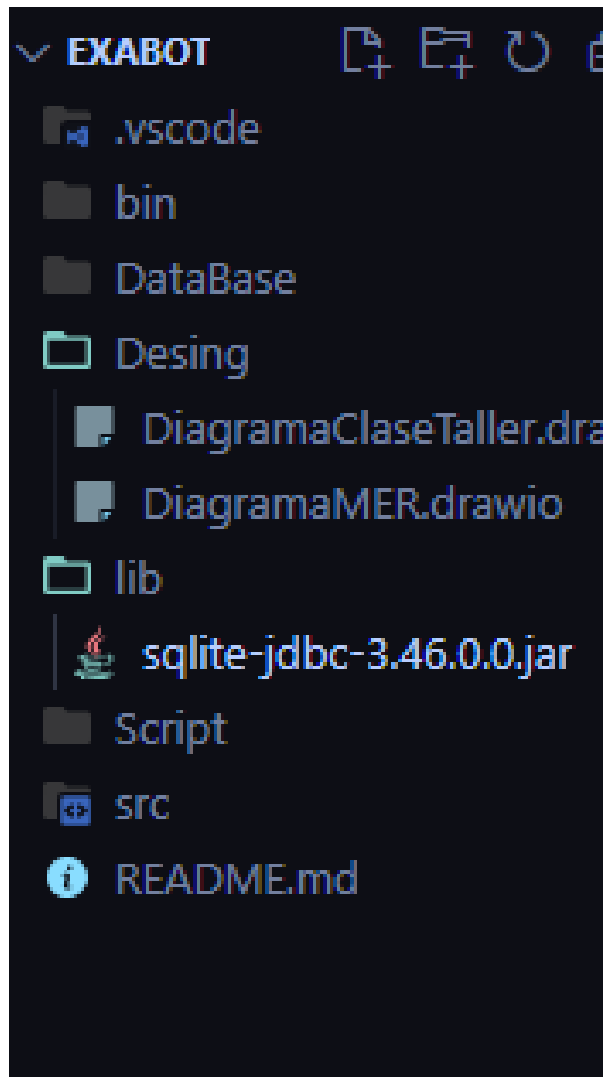
JDBC

JDBC (Java Database Connectivity) es una API (Interfaz de Programación de Aplicaciones) para el lenguaje de programación Java. Su objetivo es permitir que las aplicaciones Java se conecten a bases de datos y ejecuten consultas de manera independiente del sistema operativo o la base de datos específica. Aquí tienes algunos puntos clave sobre JDBC:

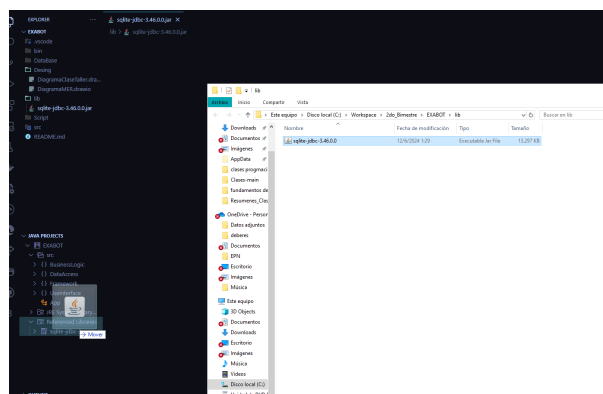
- Definición: JDBC proporciona métodos para consultar y actualizar datos en una base de datos, especialmente orientada a bases de datos relacionales.
- Parte de Java SE: Forma parte de la plataforma Java Standard Edition (Java SE) de Oracle Corporation.
- Versiones: Ha evolucionado a lo largo del tiempo, con versiones como JDBC 3.0, JDBC 4.0, JDBC 4.1, JDBC 4.2 y la más reciente, JDBC 4.3.
- Funcionalidad: Permite crear conexiones a bases de datos, ejecutar declaraciones SQL (como SELECT, INSERT, UPDATE y DELETE) y trabajar con procedimientos almacenados.
- Implementaciones múltiples: Dado que JDBC es principalmente una colección de definiciones de interfaces, permite que múltiples implementaciones de estas interfaces - coexistan y sean utilizadas por la misma aplicación en tiempo de ejecución.

Librería

Después de buscar en la librería los drivers jdbc para descargar el archivo y arrastrarlo a la librería de nuestro proyecto



Puede suceder que al abrir el proyecto en otra maquina pierda la referencia de donde esta el jdbc asi que hacemos click derecho y ponemos ubicacion del archivo y luego arrastramos hasta donde esta la parte de javaproject en ReferenceLibraries



DTO

Un DTO (Data Transfer Object) es un objeto utilizado para encapsular datos y enviarlos de un subsistema de una aplicación a otro. Aquí tienes algunos puntos clave sobre los DTOs:

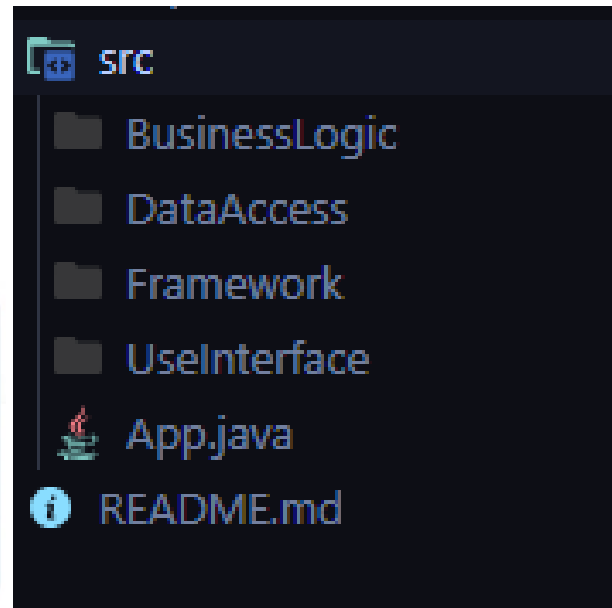
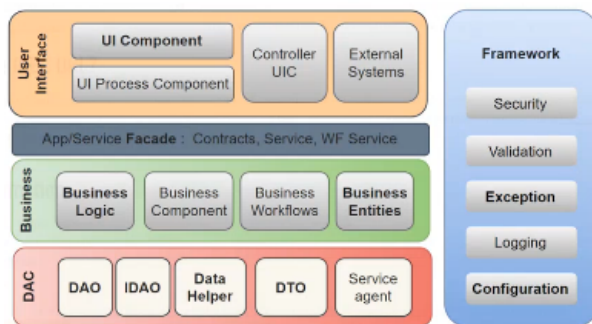
- **Propósito:** Los DTOs se utilizan principalmente en la capa de Servicios de una aplicación N-Tier para transferir datos entre sí y la capa de Interfaz de Usuario (UI).
- **Beneficios:**

- Reducen la cantidad de datos que deben enviarse a través de la red en aplicaciones distribuidas.
 - Son útiles como modelos en el patrón MVC (Modelo-Vista-Controlador).
 - También pueden encapsular parámetros para llamadas a métodos.
- Assemblers: Al usar el patrón DTO, se suelen emplear ensambladores de DTOs. Estos convierten objetos del dominio en DTOs y viceversa.

Proyecto

Como podemos ver en la imagen los demas van hacer package que vamos a crear en src:

Arquitectura N-TIER



Esto es lo que vamos a copiar en la clase abstract `SQLiteDataHelper`

```

1 package DataAccess;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5 import java.sql.SQLException;
6
7 public abstract class SQLiteDataHelper {
8
9     private static String DBPathConnection = "jdbc:sqlite:Database/ExaBot.sqlite";
10     private static Connection conn = null;
11     protected SQLiteDataHelper() {}
12
13     protected static synchronized Connection openConnection() throws Exception {
14         try {
15             if (conn == null) {
16                 conn = DriverManager.getConnection(DBPathConnection);
17             } catch (SQLException e) {}
18             throw e; //new Exception(e, "SQLiteDataHelper", "Fallo la conexión con la base de datos")
19         }
20         return conn;
21     }
22
23     protected static void closeConnection() throws Exception {
24         try {
25             if (conn != null) {
26                 conn.close();
27             } catch (SQLException e) {}
28             throw e; //new Exception(e, "SQLiteDataHelper", "Fallo la conexión con la base de datos")
29         }
30     }
31 }

```

En la clase interface hay que copiar este código

```

1 package DataAccess;
2
3 import java.util.List;
4
5 public interface IDAO <T> {
6     public boolean create(T entity) throws Exception;
7     public List<T> readAll() throws Exception;
8     public boolean update(T entity) throws Exception;
9     public boolean delete(Integer id) throws Exception;
10
11     public T readBy(Integer id) throws Exception;
12
13 }

```

1. En el código podemos ver al lado de la clase este símbolo este se le llama genérico y nos ayuda a que cuando usemos los métodos en otras clases puedas ir cambiando ese nombre en los métodos y no tengas que perder tu tiempo cambiando en cada una de las clases persona, sexo, etc. En la siguiente imagen puedes ver donde dice `Object` y cambias el nombre que tu quieras y listo te ahorras tiempo.

```

DataAccess > SexoDAO.java > Language Support for Java(TM) by Red Hat > SexoDAO
package DataAccess;

public class SexoDAO extends SQLiteOpenHelper implements IDAO<Object>{
}

```

2. `throws Exception`; nos va ayudar por si ocurre un error

En las clases DTO tenemos que poner los datos de las tablas que creamos de la base de datos

3. En nuestro package DTO creamos nuestra clase `SexoDTO.java` y copiamos los siguiente

```

package DataAccess.DTO;

public class SexoDTO {
    private Integer idSexo;
    private String Nombre;
    private String Estado;
    private String FechaCrea;
    private String FechaModifica;

    public SexoDTO(){}

    public SexoDTO(String nombre){
        this.Nombre = nombre;
    }

    public SexoDTO(int idSexo, String nombre, String estado, String fechacrea, String fechaModifica){
        this.IdSexo = idSexo;
        this.Nombre = nombre;
        this.Estado = estado;
        this.FechaCrea = fechacrea;
        this.FechaModifica = fechaModifica;
    }

    public Integer getIdSexo() {
        return IdSexo;
    }

    public void setIdSexo(Integer idSexo) {
        IdSexo = idSexo;
    }

    public String getNombre() {
        return Nombre;
    }

    public void setNombre(String nombre) {
        Nombre = nombre;
    }

    public String getEstado() {
        return Estado;
    }

    public void setEstado(String estado) {
        Estado = estado;
    }
}

```

- Todo lo que encontramos es lo que esta en nuestra tabla Sexo de nuestra base de datos
4. También vamos a escribir nuestra interface IDAO que va hacer el que tenga nuestros métodos que van a repartirse en las clases que vayamos creando que se van a conectar a nuestra base de datos

```

package DataAccess;

import java.util.List;

public interface IDAO<T> {
    public T readBy(Integer id) throws Exception;
    public boolean create(T entity) throws Exception;
    public List<T> readAll() throws Exception;
    public boolean update(T entity) throws Exception;
    public boolean delete(Integer id) throws Exception;
}

```

5. Por último nuestra clase sexo:

```

You, 9 hours ago | 1 author (You)
public class SexoDAO extends SQLiteOpenHelper implements IDAO<SexoDTO>{

    @Override
    public SexoDTO readBy(Integer id) throws Exception {
        SexoDTO oS = new SexoDTO();
        String query = "SELECT IdSexo1"
            + " ,Nombre"
            + " ,Estado"
            + " ,FechaCrea"
            + " ,FechaModifica"
            + " From Sexo"
            + " WHERE ESTADO = 'A' AND IdSexo= " + id.toString();

        try {
            Connection conn = openConnection(); //conectar a BD
            Statement stmt = conn.createStatement(); //CRUD: Select +
            ResultSet rs = stmt.executeQuery(query); //ejecutar la
            while (rs.next()) {
                oS = new SexoDTO(
                    rs.getInt(columnIndex:1) //IdSexo
                    ,rs.getString(columnIndex:2) //Nombre
                    ,rs.getString(columnIndex:3) //Estado
                    ,rs.getString(columnIndex:4) //FechaCrea
                    ,rs.getString(columnIndex:5)); //FechaModifica
            }

        } catch (SQLException e) {
            throw e;
            // throw new PatException(e.getMessage(), getClass().getName(), "readBy()");
        }
        return oS;
    }

    @Override
    public List<SexoDTO> readAll() throws Exception { ...

    @Override
    public boolean create(SexoDTO entity) throws Exception { ...

    @Override
    public boolean update(SexoDTO entity) throws Exception { ...

    @Override
    public boolean delete(Integer id) throws Exception { ...

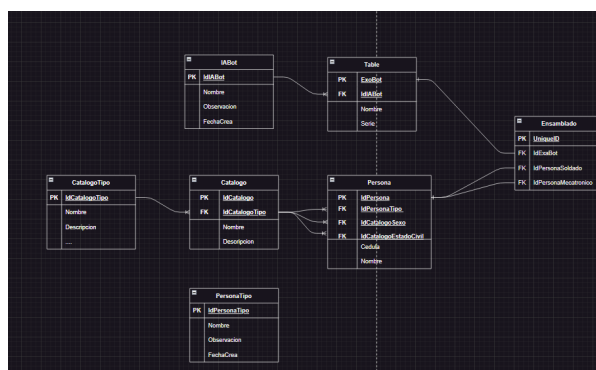
    public Integer getMaxRow() throws Exception{ You, 9 hours ago • First ...

```

Clase nro 12

DAC + DataBase

En esta clase empezamos por crear un catalogo en nuestro MER



y así se vería en nuestro DDL, también modificamos la clase persona:

```

> Execute (Shift + Enter)
DROP TABLE IF EXISTS CatalogoTipo;
> Execute (Shift + Enter)
DROP TABLE IF EXISTS Catalogo;
> Execute (Shift + Enter)
DROP TABLE IF EXISTS IABot;
> Execute (Shift + Enter)
DROP TABLE IF EXISTS ExaBot;
> Execute (Shift + Enter)
DROP TABLE IF EXISTS Persona;
> Execute (Shift + Enter)
DROP TABLE IF EXISTS PersonaTipo;
> Execute (Shift + Enter)
DROP TABLE IF EXISTS Sexo;

> Execute (Shift + Enter)
CREATE TABLE CatalogoTipo(
    IdCatalogoTipo    INTEGER NOT NULL PRIMARY KEY autoincrement
    ,Nombre            VARCHAR(18) NOT NULL UNIQUE
    ,Descripcion       VARCHAR(98) NOT NULL UNIQUE
    ,Estado            VARCHAR(1) NOT NULL DEFAULT('A')
    ,FechaCreacion     DATETIME DEFAULT(datetime('now','localtime'))
    ,FechaModifica     DATETIME
);

> Execute (Shift + Enter)
CREATE TABLE Catalogo(
    IdCatalogo        INTEGER NOT NULL PRIMARY KEY autoincrement
    ,IdCatalogoTipo    INTEGER NOT NULL REFERENCES CatalogoTipo(IdCatalogoTipo)
    ,Nombre            VARCHAR(18) NOT NULL UNIQUE
    ,Descripcion       VARCHAR(98) NOT NULL UNIQUE
    ,Estado            VARCHAR(1) NOT NULL DEFAULT('A')
    ,FechaCreacion     DATETIME DEFAULT(datetime('now','localtime'))
    ,FechaModifica     DATETIME
);

CREATE TABLE Persona (
    IdPersona          INTEGER PRIMARY KEY autoincrement
    ,IdCatalogoTipo    INTEGER NOT NULL REFERENCES Catalogo(IdCatalogoTipo)
    ,IdCatalogoSexo    INTEGER NOT NULL REFERENCES Catalogo(IdCatalogoSexo)
    ,IdCatalogoEstadocivil INTEGER NOT NULL REFERENCES Catalogo(IdCatalogoEstadocivil)
    ,Cedula            VARCHAR(10) NOT NULL UNIQUE
    ,Nombre            VARCHAR(50) NOT NULL
    ,Apellido          VARCHAR(50) NOT NULL
    ,Estado            VARCHAR(1) NOT NULL DEFAULT('A')
    ,FechaCreacion     DATETIME DEFAULT(datetime('now','localtime'))
    ,FechaModifica     DATETIME
);

```

Esta es la informacion que vamos a meter en nuestras tablas:

```

INSERT INTO CatalogoTipo
(Nombre,Descripcion)VALUES
('Tipo Persona','Tipo sol,mec...')
('Sexo','mas, feme...')
('Estado Civil','sol,viu,casa...')
('Raza','mestiz,Afro,blan...');

> Execute (Shift + Enter)
INSERT INTO Catalogo
(IdCatalogoTipo,Nombre,Descripcion)VALUES
(1,'Soldado','tipo persona del ejercito')
(1,'Mecanico','tipo persona del ejercito')
(1,'Experto Ing','tipo persona del ejercito')
(1,'Experto Esp','tipo persona del ejercito')

(2,'Maculino','tipo sexualidad')
(2,'Femenino','tipo sexualidad')
(2,'Hibrido','tipo sexualidad')
(2,'Asexual','tipo sexualidad')

(3,'Soltero','tipo de Estado Civil')
(3,'Casado','tipo de Estado Civil')
(3,'Divorciado','tipo de Estado Civil')
(3,'Viudo','tipo de Estado Civil')

(4,'Negro','tipo de Etnia')
(4,'Blanco','tipo de Etnia')
(4,'Mestizo','tipo de Etnia')
(4,'Indigena','tipo de Etnia');

> Execute (Shift + Enter)
INSERT INTO IABot
(Nombre,Observacion)VALUES
('IA-Ruso','Inteligencia artificial');

> Execute (Shift + Enter)
INSERT INTO ExaBot
(IdIABot,Nombre,Serie)VALUES
(1,'exabot1','Serie E1')
(1,'exabot2','Serie E2');

> Execute (Shift + Enter)
INSERT INTO Persona
(IdCatalogoTipo, IdCatalogoSexo, IdCatalogoEstadocivil, Cedula, Nombre, Apellido)VALUES
(1,5,11,47841,'Carlos','Andrade')
(2,6,10,23432,'Maria','Rodriguez')
(3,6,11,32432,'Karla','Llinin')
(4,5,12,98769,'Daniela','Perez');

```

- A continuacion vamos a crear otra clase llamada SexoDAO y la que ya estaba con ese nombre la vamos a cambiar por SexoAntDAO:
- Como vimos en esta clase lo de catalogo estas clases nueva de Sexo van a tener diferentes propiedades que la anterior clase sexo y lo que intentamos ver aqui es que el codigo que usamos en la anterior va a ser identico solo que con unos cambios en la clase SexoDAO y en la clase SexoAntDAO.A continuacion van a ver el codigo final de las nuevas clases:

1. Clase SexoDTO

```

package DataAccess.DTO;

public class SexoDTO {
    private Integer IdCatalogo;
    private Integer IdCatalogoTipo;
    private String Nombre;
    private String Descripcion;
    private String Estado;
    private String FechaCreacion;
    private String FechaModifica;

    public SexoDTO(){}

    public SexoDTO(Integer idCatalogo, Integer idCatalogoTipo, String nombre, String descripcion, String estado,
        String fechaCreacion, String fechaModifica) {
        this.IdCatalogo = idCatalogo;
        this.IdCatalogoTipo = idCatalogoTipo;
        this.Nombre = nombre;
        this.Descripcion = descripcion;
        this.Estado = estado;
        this.FechaCreacion = fechaCreacion;
        this.FechaModifica = fechaModifica;
    }

    public Integer getIdCatalogo() {
        return IdCatalogo;
    }

    public void setIdCatalogo(Integer idCatalogo) {
        IdCatalogo = idCatalogo;
    }
}

```

2. Clase SexoDAO


```

@Override
public boolean create(SexoDTO entity) throws Exception {
    String query = "INSERT INTO Catalogo (IdCatalogoTipo,Nombre,Descripcion) VALUES(?,?,?)";
    try {
        Connection conn = openConnection(); //conectar a BD
        PreparedStatement pstmt = conn.prepareStatement(query);
        pstmt.setInt(parameterIndex:1,x2);
        pstmt.setString(parameterIndex:1, entity.getNombre());
        pstmt.setString(parameterIndex:2,entity.getDescription());
        pstmt.executeUpdate();
        return true;
    }
    catch (SQLException e){
        throw e;
    }
    // throw new PatException(e.getMessage(), getClass().getName(), "create");
}

@Override
public List<SexoDTO> readAll() throws Exception {
    List <SexoDTO> lts = new ArrayList<>();
    String query = "SELECT IdCatalogo"
        + ".IdCatalogoTipo"
        + ".Nombre"
        + ".Descripcion"
        + ".Estado"
        + ".FechaCreacion"
        + ".FechaModifica"
        + " From Catalogo"
        + " WHERE ESTADO = 'A'"
        + " AND IdCatalogoTipo = 2";

    try {
        Connection conn = openConnection(); //conectar a BD
        Statement stat = conn.createStatement(); //CRUD: Select +
        ResultSet rs = stat.executeQuery(query); //ejecutar la
        while (rs.next()) {
            SexoDTO s = new SexoDTO( rs.getInt(columnIndex:1)
                ,rs.getInt(columnIndex:2) //IdCatalogoTipo
                ,rs.getString(columnIndex:3) //Nombre
                ,rs.getString(columnIndex:4) //Descripcion
                ,rs.getString(columnIndex:5) //Estado
                ,rs.getString(columnIndex:6) //FechaCrea
                ,rs.getString(columnIndex:7)); //FechaModifica

            lts.add(s);
        }
    }
}

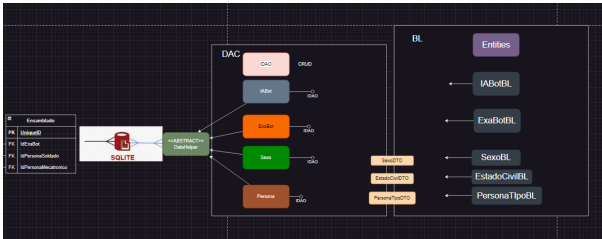
```

```
@Override
public boolean delete(int id) throws Exception {
    String query = "UPDATE Catalogo SET Estado=? WHERE IdCatalogo=?";
    try {
        Connection conn = openConnection(); //conectar a BD
        PreparedStatement pstmt = conn.prepareStatement(query);
        pstmt.setString(parameterIndex:1, x:"X");
        pstmt.setInt(parameterIndex:2, id);
        pstmt.executeUpdate();
        return true;
    } catch (Exception e) {
        throw e;
        // throw new PatException(e.getMessage(), getClass().getName(), "delete()");
    }
}

public Integer getRowCount() throws Exception{
    String query = "SELECT COUNT(*) TotalReg FROM Catalogo"
        + " WHERE Estado = 'A'"
        + " AND IdCatalogoTipo = 2";
    try {
        Connection conn = openConnection(); //conectar a BD
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(query);
        while (rs.next()) {
            return rs.getInt(columnIndex:1);
        }
    } catch (SQLException e){
        throw e;
        // throw new PatException(e.getMessage(), getClass().getName(), "getMaxRow()");
    }
    return 0;
}
```

[illegible]

33 / 34



Y en nuestro package BusinessLogic vamos a crear otro package llamado Entities y ahi vamos a crear nuestra clase SexoBL. Vamos a copiar el siguiente codigo en esa clase:

```
src > BusinessLogic > Entities > SexoBL.java > ...
1 package BusinessLogic.Entities;
2
3 import java.util.List;
4
5 import DataAccess.SexoDAO;
6 import DataAccess.DTO.SexoDTO;
7
8 public class SexoBL {
9
10     private SexoDTO sexo;
11     private SexoDAO sDAO = new SexoDAO();
12
13     public SexoBL(){}
14
15     > public List<SexoDTO> getALL() throws Exception{...
16
17     public SexoDTO getIdSexo(int idSexo) throws Exception{
18         sexo = sDAO.readBy(idSexo);
19         return sexo;
20     }
21
22     public boolean create (SexoDTO sexoDTO) throws Exception{
23         return sDAO.create(sexoDTO);
24     }
25
26     public boolean update (SexoDTO sexoDTO) throws Exception{
27         return sDAO.update(sexoDTO);
28     }
29
30     public boolean delete (int idSexo) throws Exception{
31         return sDAO.delete(idSexo);
32     }
33
34     public Integer getRowCount() throws Exception{
35         return sDAO.getRowCount();
36     }
37
38 }
39
```

Despues vamos a testear que todo este correcto y que no haya errores

```
try {
    SexoBL sBL = new SexoBL();
    for(SexoDTO s : sBL.getALL())
        System.out.println(s.toString());
} catch (Exception e) {
    System.out.println(e.toString());
}
```

```
Terminal
DataAccess.DTO.SexoDTO
IdCatalogo: 6
IdCatalogoTipo: 2
Nombre: femenino
Descripcion: tipo sexualidad
Estado: A
FechaCreacion: 2024-08-02 00:49:56
FechaModifica: null
DataAccess.DTO.SexoDTO
IdCatalogo: 6
IdCatalogoTipo: 2
Nombre: femenino
Descripcion: tipo sexualidad
Estado: A
FechaCreacion: 2024-08-02 00:49:56
FechaModifica: null
DataAccess.DTO.SexoDTO
IdCatalogo: 7
IdCatalogoTipo: 2
Nombre: hibrido
Descripcion: tipo sexualidad
Estado: A
FechaCreacion: 2024-08-02 00:49:56
FechaModifica: null
DataAccess.DTO.SexoDTO
IdCatalogo: 6
IdCatalogoTipo: 2
Nombre: femenino
Descripcion: tipo sexualidad
Estado: A
FechaCreacion: 2024-08-02 00:49:56
FechaModifica: null
DataAccess.DTO.SexoDTO
IdCatalogo: 7
IdCatalogoTipo: 2
Nombre: hibrido
Descripcion: tipo sexualidad
```