

Apuntes Semana 2

Clase 1

*Presentación de la Materia: Programación II

1. Presentación del Profesor

- El curso comenzó con la introducción del profesor, quien detalló su experiencia, expectativas para la clase, y cómo se desarrollaría el contenido a lo largo del semestre.

2. Presentación de los Estudiantes

- Los estudiantes tuvieron la oportunidad de presentarse, compartir sus antecedentes, intereses en la programación, y expectativas para el curso. Esto ayudó a establecer un ambiente colaborativo y conocer a los compañeros.

3. Recursos Necesarios

- Se proporcionó información sobre los recursos que serán necesarios para los talleres, incluyendo software, documentación, y otras herramientas relevantes para el curso. Estos recursos serán fundamentales para completar las tareas y proyectos asignados.

4. Primer Taller en Grupo

- El primer ejercicio práctico consistió en un taller grupal donde los estudiantes intentaron construir una torre lo más alta posible utilizando la menor cantidad de material. Este ejercicio, aunque no tuvo los mejores resultados, sirvió como una actividad de integración y como una introducción a la resolución de problemas en equipo.

5. Evaluaciones

- Se explicó cómo se realizarían las evaluaciones del curso. Las pruebas estarán diseñadas para durar dos horas, evaluando conceptos clave vistos en clase. En contraste, los exámenes serán más extensos, con una duración de ocho horas, y abarcarán aplicaciones prácticas más complejas y desafíos de programación.

6. Proyecto

- El proyecto final se entregará en dos partes: la primera parte se debe completar y entregar al final del primer bimestre, mientras que la segunda parte se entregará al final del segundo bimestre. Este proyecto servirá como una aplicación práctica y comprensiva de los conocimientos adquiridos durante el curso.

- Cuadro de evaluaciones y sus porcentajes:

Evaluación	Puntaje	Bonos Extras	Puntaje
-Prueba	25%	-Actuación	+0.1 punto

Evaluación	Puntaje	Bonos Extras	Puntaje
-Examen	25%	-Retos	+1.0 punto
-Workshop	10%		
-Homework	10%		
-Proyecto	30%		

Explicación Detallada del Curso: Programación II

1. Explicación General del Curso

- Se dio una visión general de cómo se estructurará el curso, destacando la importancia de la práctica constante y el uso de herramientas específicas que los estudiantes deberán instalar y utilizar a lo largo del semestre.

2. Aplicaciones Requeridas y Optimización

- Se enumeraron las aplicaciones que deben instalarse para realizar los trabajos prácticos. Se ofrecieron consejos sobre cómo optimizarlas para mejorar el rendimiento, así como opciones de personalización para hacerlas más cómodas y eficientes, lo que permitirá a los estudiantes maximizar su productividad y obtener mejores resultados en sus proyectos.

3. Presentación de Apuntes en Markdown

- Al final de cada bimestre, los estudiantes deberán presentar sus apuntes utilizando el formato Markdown. Esto no solo fomenta una organización clara y profesional de las notas, sino que también ayuda a desarrollar habilidades en el uso de este formato ampliamente utilizado en la documentación técnica.

4. Talleres en Grupo

- Durante el curso, se realizarán varios talleres en grupo que permitirán a los estudiantes aplicar lo aprendido de manera colaborativa. Estos talleres estarán diseñados para fomentar el trabajo en equipo y la resolución conjunta de problemas de programación.

5. Proyecto en Grupo

- El proyecto final del curso se desarrollará en grupo y deberá incluir un aparato externo como parte de su implementación. Esto podría involucrar la integración de hardware con software, lo que representa un desafío adicional y una oportunidad para explorar áreas como el desarrollo de sistemas embebidos o la automatización.

6. Presentación de la Normativa

- Se presentó la normativa del curso, que incluye reglas sobre la asistencia, la entrega de trabajos, y el comportamiento esperado en clase y durante las actividades grupales. Cumplir con estas normativas es esencial para mantener un ambiente de aprendizaje efectivo y respetuoso.

7. Personalización de Aplicaciones de Trabajo

- Se proporcionaron instrucciones sobre cómo personalizar las aplicaciones de trabajo, como editores de código y entornos de desarrollo, para adaptarlas a las necesidades individuales de cada estudiante. La personalización puede incluir la configuración de temas, atajos de teclado, y extensiones que mejoren la eficiencia durante las sesiones de codificación.

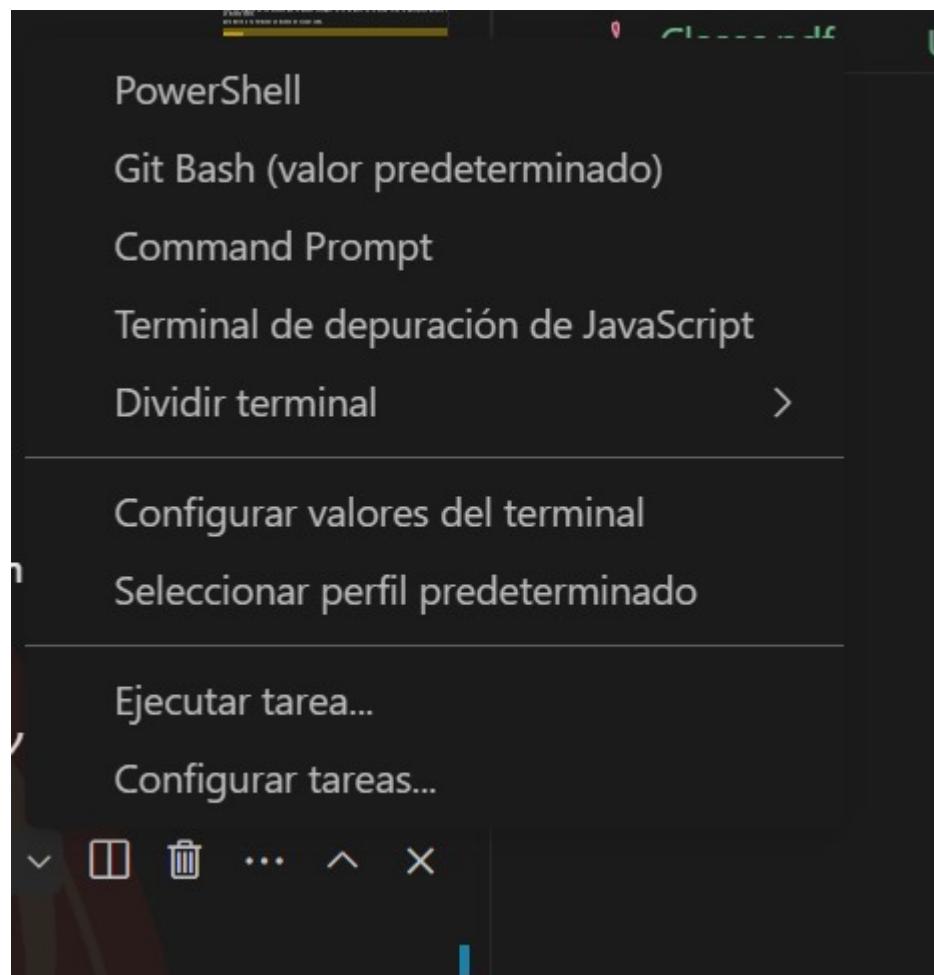
8. Aprendizaje de Comandos

- Se destacó la importancia de aprender y dominar los comandos necesarios para trabajar eficientemente con las aplicaciones y herramientas mencionadas. Esto incluye comandos para la línea de comandos, así como comandos específicos dentro de los entornos de desarrollo, que son fundamentales para acelerar el flujo de trabajo y evitar errores comunes.

Clase 2

Personalización del Visual Code y de Git bash

- Para la personalización del Git Bash y del visual code, se puede personalizar su terminal para que esta sea la determinada en Visual Code y salga por defecto automáticamente, cada vez que se le llame con el comando CTRL + Ñ o manualmente.



Ademas se puede personalizar el git bash en si con la ayuda de otras aplicaciones externas, como por ejemplo: *Oh my posh* que es una de varias que pueden ayudar a darle un toque mas personal a la terminal y la consola de Git bash, con varios diseños y fuentes de se pueden instalar facilmente.

```
$ exec bash
Home ~ ✓ nano .bashrc
Home ~ ✓ exec bash
Celeste ~ 0ms 4:02 PM
Home >> nano .bashrc
Celeste ~ 36.749s 4:03 PM
Home >> exec bash
[ Home@Celeste ~ 0ms
$ nano .bashrc
[ Home@Celeste ~ 20.864s
$ exec bash

37 RAM: 11/31GB 0ms
bash 4:05:43 PM | Saturday nano .bashrc

37 RAM: 11/31GB 16s 267ms
bash 4:06:38 PM | Saturday exec bash
~ 0ms Home bash

> nano .bashrc
~ 14.703s Home bash

> exec bash
Home ~ 16:07 exec bash
Home ~ 16:07 nano .bashrc
Home ~ 16:08 exec bash
Home ~ 16:08 nano .bashrc
Home ~ 16:09 exec bash
[ Home@Celeste ~ nano .bashrc
[ Home@Celeste ~ exec bash
Home ~ nano .bashrc
Home ~ exec bash
Home ~ 16:12

16:11:14 0ms
```

Esos son algunos de los diseños que se pueden conseguir en Oh my posh, de la misma forma la aplicación permite crear un diseño nuevo, al final de la personalización se puede obtener un resultado así en la terminal.

```
terminal session on Celeste
1. touch nota1.java
2. git config user.name
3. git config user.email inu-kun_2005@hotmail.com
4. git config user.name
5. Home >>
```

- De la misma forma se puede personalizar el color de visual estudio con ayuda de los temas que vienen en la aplicación, pero tambien se pueden instalar otros, así mismo con los iconos de las aplicaciones o archivos que se abran dentro del visual.
- Tambien es recomendable cambiar la letra del visual code y de git bash para que sea mas comoda al momento de trabajar, pero eligiendo una que sea entendible y que facilite el trabajo.

Con todo eso listo, se puede tener una buena herramienta, que sea comoda y al gusto del usuario para trabajar adecuadamente.

Comandos 😊

• Windows

- CTRL + SHIFT + P: para abrir la barra de comandos donde estan las aplicaciones y funciones.
- CTRL + P: para ver los documentos que se han abierto recientemente o buscar algun archivo para abrirlo.
- CTRL + B: para aparecer y desaparecer la barra donde se observan las carpetas y documentos que estaban abiertos o almacenados.
- CTRL + D: para seleccionar varias lineas.
- CTRL + S: para guardar.
- CTRL + Ñ: para aparecer y desaparecer la terminal.
- SHIFT + ALT + UP/DOWN: para copiar y pegar una linea del codigo.
- ALT + ->: Para moverse entre los archivos que se encuentran abiertos en la barra de arriba.
- ALT + ->(arriba): Para mover una linea del codigo ya sea arriba o abajo.
- SHIFT + ALT + A: para poner una linea de comentario.
- SHIFT + ALT: para seleccionar la linea o palabra.

• Linux

- pwd: para saber en que carpeta actual te encuentras.

- cd: para cambiar de carpeta.
- ls: para ver los archivos y carpetas que se encuentran en la carpeta actual.
- mkdir: para crear una nueva carpeta.
- rmdir: para eliminar una carpeta.
- rm: para eliminar un archivo.
- cp: para copiar un archivo.
- mv: para mover un archivo.
- cat: para ver el contenido de un archivo.
- nano: para editar un archivo.
- clear: para limpiar la terminal.
- history: para ver los comandos que se han ejecutado recientemente.
- :wq: para guardar y salir de un archivo.
- :q: para salir de un archivo sin guardar.
- :q!: para salir de un archivo sin guardar y sin confirmar.
- :set nu: para mostrar los numeros de las lineas.
- :set nonu: para ocultar los numeros de las lineas.
- :set list: para mostrar los caracteres no impresos.
- :set nolist: para ocultar los caracteres no impresos.
- :set ic: para hacer la busqueda insensible a mayusculas y minusculas.
- :set noic: para hacer la busqueda sensible a mayusculas y minusculas.
- :set hlsearch: para mostrar los resultados de la busqueda.
- :set nohlsearch: para ocultar los resultados de la busqueda.
- touch: para crear un nuevo archivo de cualquier tipo.
- code: para abrir el documento creado.

Clase 3

Uso de Markdown y Manejo de Git/GitHub

Uso de Markdown

Markdown es una herramienta poderosa para tomar apuntes y crear documentos sin salir del entorno de desarrollo, como Visual Studio Code. Una de las ventajas de Markdown es que permite generar archivos en

formato PDF para su distribución y lectura, lo que lo hace ideal para documentar código y elaborar informes.

Elementos Básicos de Markdown

- **Títulos:**

- Se crean utilizando el símbolo `#`. Cuantos más `#` se utilicen, más bajo será el nivel del título.
- Ejemplo:

```
# Título Principal  
## Subtítulo  
### Subtítulo Nivel 3
```

- **Cursiva:**

- Se aplica rodeando la palabra o frase con un asterisco `*` a cada lado.
- Ejemplo: `*Cursiva*` se ve como *Cursiva*.

- **Negrillas:**

- Se aplica rodeando la palabra o frase con dos asteriscos `**` a cada lado.
- Ejemplo: `**Negrilla**` se ve como **Negrilla**.

- **Negrilla y Cursiva:**

- Se aplican simultáneamente rodeando el texto con tres asteriscos `***` a cada lado.
- Ejemplo: `***Negrilla y Cursiva***` se ve como **Negrilla y Cursiva**.

- **Enlaces:**

- Se crean usando el formato `[Texto del enlace](URL)`.
- Ejemplo: `[YouTube](https://www.youtube.com/)` enlaza a [YouTube](https://www.youtube.com/).

- **Imágenes:**

- Se insertan con el formato `![Texto alternativo](Ruta o URL de la imagen)`.
- Ejemplo: `![Texto alternativo](image4.jpg)` inserta una imagen.

- **Bloques de Código:**

- Se rodea el código con tres acentos graves ````` y se indica el lenguaje de programación para resaltar la sintaxis.
- Ejemplo:

```
```c++
#include <iostream>
using namespace std;
int main() {
 cout << "Hola, Mundo!";
 return 0;
}
```

- **Tablas:**

- Se crean utilizando | para separar columnas y --- para definir los encabezados.
- Ejemplo:

Columna 1	Columna 2
Dato 1	Dato 2
Dato 3	Dato 4

- **Comandos Útiles:**

- **CTRL + SHIFT + V:** Permite ver la vista previa del documento Markdown en Visual Studio Code.

## Manejo de Git/GitHub

Git y GitHub son esenciales para el control de versiones y la colaboración en proyectos de programación. Aquí se resumen los comandos básicos y su aplicación.

### Comandos de Git

- **Inicialización del Repositorio:**

- **git init:** Crea un nuevo repositorio de Git en el directorio actual.

- **Estado del Repositorio:**

- **git status:** Muestra el estado de los archivos, indicando cuáles han sido modificados o agregados.

- **Añadir Archivos:**

- **git add .:** Añade todos los archivos y cambios al área de preparación.

- **Guardar Cambios (Commit):**

- `git commit -m "Mensaje"`: Guarda los cambios realizados en el repositorio con un mensaje descriptivo.

- **Subir Cambios:**

- `git push origin main`: Envía los cambios locales al repositorio remoto en la rama principal.

- **Actualizar Cambios Locales:**

- `git pull`: Descarga los cambios más recientes del repositorio remoto y los integra en la rama local.

- **Clonar Repositorios:**

- `git clone [URL del repositorio]`: Crea una copia local de un repositorio remoto.

- **Comprobar Repositorio Remoto:**

- `git remote -v`: Verifica la URL del repositorio remoto con el que está sincronizado el repositorio local.

- **Ver Log de Cambios:**

- `git log`: Muestra el historial de commits del repositorio.

## Subida de Archivos a GitHub

### 1. Clonar un Repositorio:

- Crear una carpeta en la que se clonará el repositorio de GitHub.
- Utilizar Git Bash y ejecutar `git clone [URL del repositorio]` en la terminal dentro de la carpeta creada.

### 2. Realizar Cambios y Subir a GitHub:

- Abrir Visual Studio Code y utilizar los siguientes comandos:
  - `git status`: Verifica los cambios realizados.
  - `git add .`: Agrega todos los archivos al área de preparación.
  - `git commit -m "Mensaje descriptivo"`: Realiza un commit con los cambios.
  - `git push origin main`: Sube los cambios al repositorio en GitHub.
  - `git log`: Comprueba el historial de commits para asegurarse de que los cambios se han subido correctamente.

### 3. Resumen del Proceso de Trabajo:

- Clonar el repositorio, realizar los cambios necesarios en la carpeta clonada, y luego utilizar los comandos de Git para gestionar y subir esos cambios al repositorio en GitHub.

Con esta combinación de Markdown para la documentación y Git/GitHub para la gestión de código, se puede mantener un flujo de trabajo organizado y eficiente, permitiendo a los desarrolladores colaborar y compartir su progreso de manera efectiva.

## Apuntes Semana 3

---

### Clase 4

*Programacion orientada a objetos*

- Java
  - Nace por problemas con el compilador, por las diferencias entre un sistema y otro.
  - Ayuda a la distribución del código complementando los códigos del compilador.
- Funcionamiento:
  - código
  - compilación
  - bytecode
  - JVM
  - Multiplataforma

*Tipos de lenguaje*

- Compilador:

No ejecuta el programa si tiene el mínimo error, proceso de compilación que se presenta en el "trabajo" solo se tiene consola no el F5.
- Interpretador:

Es más permisivo y lee el código, se detiene al encontrar un error.



*Tipos de Java*

Se tienen varios.



### Estructura del lenguaje

- *package*: conjunto de librerias.
- *import*: escoger una libreria.
- *\*\*\**: para traer, pero es mejor traer de forma mas especifica para no cargar demaciado el sistema.
- *class*: usa clases para ejecutar el codigo, debe estar en minusculas.
- *string*: tipos
- *public static void main*: aprender debe estar alli.

### Ejemplo

```

1 package team.ed.course;
2 import java.lang.*;
3 public class Person {
4 private String name;
5 public static void main(String args[]){
6 Person friend = new Person();
7 friend.name = "Peter";
8 System.out.println("Hola " +
9 friend.name);
10 }
11 }

```



### Diferencias

Existen varias diferencias entre las estructuras y Programación orientada a objetos (O.O) de como funcionaba en C++ y como funciona ahora en Java.

Estructura	O.O
include (libreria)	import (paquetes)
funciones	metodos
struct	clases
variables	propiedades

- Propiedades:
  - Ambito: public(+), private(-), protec, friendly. (si se olvida de colocar esto el programa solo se pone en privado, en pocos casos en publico)
    - en URL tienen simbolos.
  - Tipo de dato: int, char.
  - Nombre
- Metodos: Las variables si existen solo que se encuentran dentro de los métodos, no se puede tener fuera de estos *si se tiene fuera se convierten en propiedad.*

- Método que retorne valor.
- Método sin retorno (void).

*Diferencia entre punteros En almacenamiento es lo mismo, pero al referirse en C es mas complicado que en Java, en c se puede darña facilmente y se debe gestionar "manualmente" los punteros, mientras que en Java esto se realiza de forma automatica, ya no es necesario todo el procedimiento que se da en C, basicamente Java facilita el proceso.*

Se pueden utilizar diagramas de flujo y el UML.

NO OLVIDAR EL PUNTO Y COMA (😉 :v

### Aplicaciones

Java gana en aplicaciones industriales.

### Diagrama de flujo

Es importante la ilustración de diagramas de flujo para ver de forma gráfica el código.

### Diagrama de flujo

Símbolo	Nombre	Función
	Inicio / Final	Representa el inicio y el final de un proceso
	Línea de Flujo	Indica el orden de la ejecución de las operaciones. La flecha indica la siguiente instrucción.
	Entrada / Salida	Representa la lectura de datos en la entrada y la impresión de datos en la salida
	Proceso	Representa cualquier tipo de operación
	Decisión	Nos permite analizar una situación, con base en los valores verdadero y falso

**TRACE**

Pseudocódigo. Pruebas de escritorio, seguimiento a la corrida de un programa. Automatizar el TRACE.

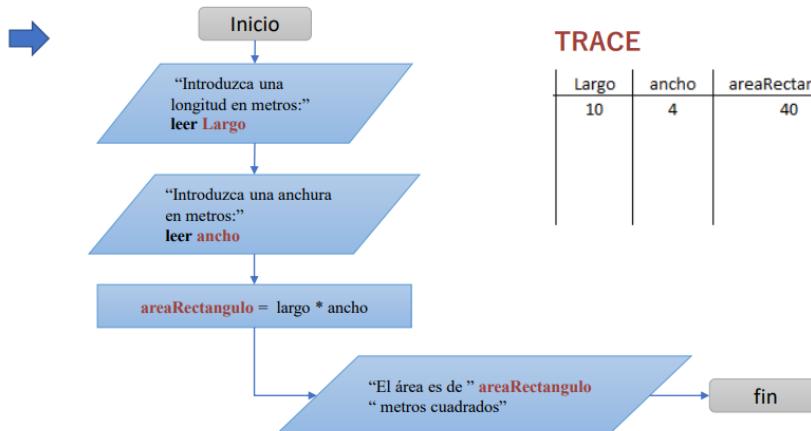
- Errores: el como esta escrito, mayusculas, minusculas, etc. Todo eso afecta.

- Forma correcta:

```

imprimir "Introduzca una longitud en metros:"
leer longitud
imprimir "Introduzca una anchura en metros:"
leer ancho
asignar a areaRectangulo = largo * ancho
imprimir "El área es de" areaRectangulo "metros cuadrados"

```

**TRACE**

Largo	ancho	areaRectangulo	Salida -> Terminal
10	4	40	Introduzca una longitud en metros 10 Introduzca una anchura en metros 4 El área es de 40 metros cuadrados

**ISSUE <> ERROR <> BUG**

- For incorrecta:

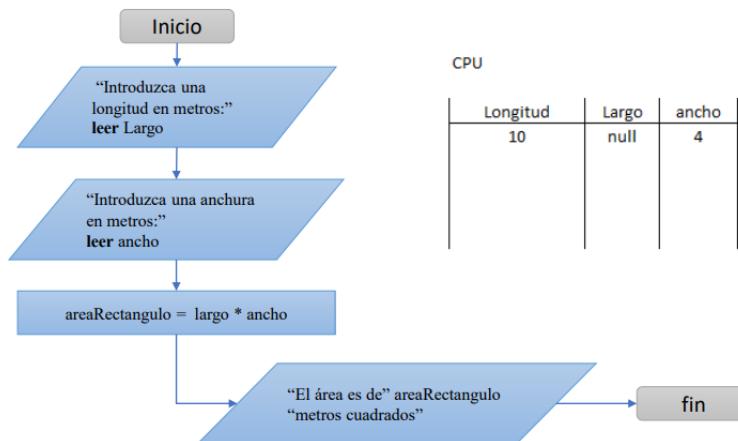
```

imprimir "Introduzca una longitud en metros:"
leer longitud
imprimir "Introduzca una anchura en metros:"
leer ancho
asignar a areaRectangulo = largo * ancho X
imprimir "El área es de" areaRectangulo "metros cuadrados"

```

**INTERPRETE <> COPILADOR**

.exe  
.class

**CPU**

Longitud	Largo	ancho	areaRectangulo	Salida -> Terminal
10	null	4	#VALUE!	Introduzca una longitud en metros 10 Introduzca una anchura en metros 4 El área es de <null> metros cuadrados

**Primer Código**

SHIFT + CNTRL + P

Poner un entorno base, en blanco, nuevo.

- pide que guardes en una carpeta.

Nombre: Proyecto 1.

- src: código fuente.

```
public class Hi {
 //propiedades aquí afuera.
 public static void main(String[] args){
 //métodos, funciones.
 int n = 10;
 for(int i = 0; i < n; i++){
 System.out.println(i);
 }
 }
}
```

- Compilar y Ejecutar desde consola: pwd: donde se encuentra. compilar: javac *Nombre del archivo*  
Ejecutar: *Nombre del archivo*

COMO FUNCIONA UN *for*

```
int n = 10;
for(int i = 0; i < n; i++)
 System.out.println(i);
```

(int i = 0): se puede poner fuera, se pone dentro para ahorrar una línea. (i < n): se puede cambiar a true or false TRUE: acepta el código, condición. FALSE: NO FUNCIONA, es como decir que no es real o falso lo que dice, entonces el código dice: "para que pones si es falso? :v". (i++/i--): para que sume o reste.

*for para que comience en 2 y se aumente de 2 en 2*

```
int n = 10;
for(int i = 2; i < n; i+=2)
 System.out.println(i);
```

(i+=2): forma abreviada para aumentar ya no solo de 1 en 1, sino más. (i=i+2): forma no abreviada.

CONDICIONAL (*if*)

```
int n = 10;
for(int i = 0; i < n; i++)
 if (i == 6)
 System.out.println("hay un seis");
 else
 System.out.println(i);
```

cuando toque en el número 6 en la ejecución, en lugar de salir "6" saldrá "hay un seis".

(else): es parte de la misma instrucción del if.

```
public class Hi {
 //propiedades aqui afuera.
 public static void main(String[] args){
 //metodos, funciones.
 int n = 10;
 for(int i = 2; i < n; i+=2){
 if (i == 6)
 System.out.println("hay un seis");
 else
 System.out.println(i);
 }
 }
}
```

codigo mas compacto con **OPERADOR TERNARIO**

```
public class Hi {
 //propiedades aqui afuera.
 public static void main(String[] args){
 //metodos, funciones.
 int n = 10;
 for(int i = 2; i < n; i+=2){
 System.out.println("hay un seis");
 }
 }
}
```

(terminar de poner el codigo compacto xd)

## Clase 5

### Bucles

Los bucles son estructuras de control fundamentales en programación, que permiten ejecutar repetidamente un bloque de código hasta que se cumpla una condición específica. Uno de los bucles más utilizados es el **for**, el cual es particularmente útil cuando se conoce de antemano cuántas veces se debe repetir el ciclo.

#### Bucle **for**

- **Función Principal:**
  - El bucle **for** permite definir un punto de inicio, una condición para continuar iterando, y una actualización de la variable de control después de cada iteración.
  - Es ideal para recorrer rangos de números, iterar sobre elementos en una colección, o repetir una tarea un número específico de veces.

- **Sintaxis Básica:**

- En la mayoría de los lenguajes de programación, la estructura básica de un `for` se ve así:

```
for (int i = 0; i < 10; i++) {
 // Código a ejecutar en cada iteración
 System.out.println("Iteración: " + i);
}
```

- Aquí, `i` se inicializa en 0, el bucle continúa mientras `i` sea menor que 10, y `i` se incrementa en 1 después de cada iteración.

## Creación de Documentos

- Al trabajar con bucles, especialmente cuando se generan documentos o archivos en cada iteración, es crucial estructurar correctamente el código para asegurar que cada documento se cree y guarde sin errores.
- **Ejemplo de Creación de Documentos:**

- Al generar múltiples informes o archivos en un bucle, se debe asegurar que cada archivo tenga un nombre único y que los recursos (como flujos de archivo) se cierren correctamente al final de cada iteración.

```
for (int i = 0; i < 5; i++) {
 FileWriter fileWriter = new FileWriter("documento_" + i + ".txt");
 fileWriter.write("Este es el documento número " + i);
 fileWriter.close(); // Cierre del archivo
}
```

## Pretty Format

- **Descripción:**

- La extensión "Pretty Format" es una herramienta de formateo de código que asegura que el código fuente esté bien estructurado y sea legible. Es especialmente útil para mantener la consistencia en la indentación y el cierre correcto de bloques de código, como corchetes `{}` en bucles.

- **Instalación:**

- Para instalar "Pretty Format", se puede buscar en el marketplace de extensiones del editor de código (como Visual Studio Code). Una vez instalada, puede configurarse para formatear automáticamente el código al guardar.

- **Uso:**

- Ayuda a evitar errores visuales y de sintaxis, asegurando que los corchetes y otros delimitadores estén correctamente alineados y cerrados.

## Instanciación de Clases

- **Instanciación:**

- La instancia es el proceso de crear un objeto a partir de una clase. Esto se hace utilizando la palabra clave **new**, que asigna memoria y prepara el objeto para su uso.
- Ejemplo:

```
BucleFor bf = new BucleFor();
```

- En este ejemplo, **BucleFor** es la clase y **bf** es la instancia del objeto, lo que "da vida" al bucle o estructura definida en esa clase.

## Extensión Power Mode

- **Descripción:**

- Power Mode es una extensión visual que añade efectos dinámicos y llamativos al editor de código, como partículas y vibraciones de pantalla al escribir. Aunque principalmente estética, puede hacer que el trabajo en el código sea más entretenido y motivador.

- **Instalación:**

- Similar a otras extensiones, se instala desde el marketplace del editor. Aunque su función es principalmente estética, es compatible con la mayoría de los entornos de desarrollo y no interfiere con la funcionalidad principal del código.

## Revisión de Errores

- **Errores Comunes en Bucles:**

- **Condiciones de Salida Incorrectas:** Asegurarse de que el bucle tiene una condición de salida válida, de lo contrario, puede causar bucles infinitos que congelan el programa.
- **Errores de Sintaxis:** Falta de cierre de corchetes **{}**, paréntesis **()**, o puntos y comas **;** son errores comunes que pueden impedir que el código compile.
- **Acceso Fuera de Rango:** Al iterar sobre arrays o listas, asegurarse de que no se intenta acceder a un índice fuera del rango, lo que puede causar excepciones.

- **Herramientas de Depuración:**

- Utilizar las herramientas de depuración del IDE para revisar y solucionar errores antes de que se compile el código. Estas herramientas pueden detener la ejecución en puntos clave y permitir examinar el estado de las variables durante la iteración de un bucle.

- **Imagen Referencial:**

```
public class App {
 Run | Debug
 public static void main(String[] args) throws Exception {
 int a;
 Buclefor buclefor;

 //instancias new
 buclefor = new Buclefor();
 buclefor.SignoAlterno();
 }
}
```

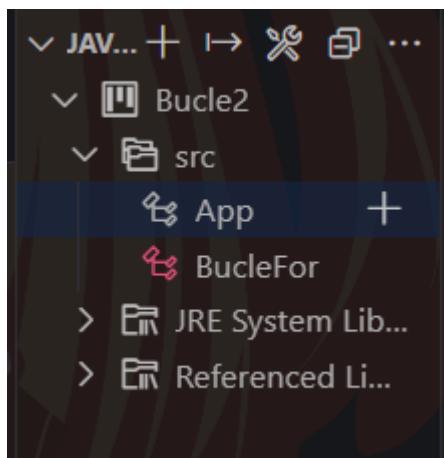
## Clase 6

### Bucles For - Parte 2

#### Orden del Directorio

Cuando se trabaja en proyectos de programación, es fundamental mantener un orden adecuado en los directorios para facilitar la organización y el acceso a los archivos.

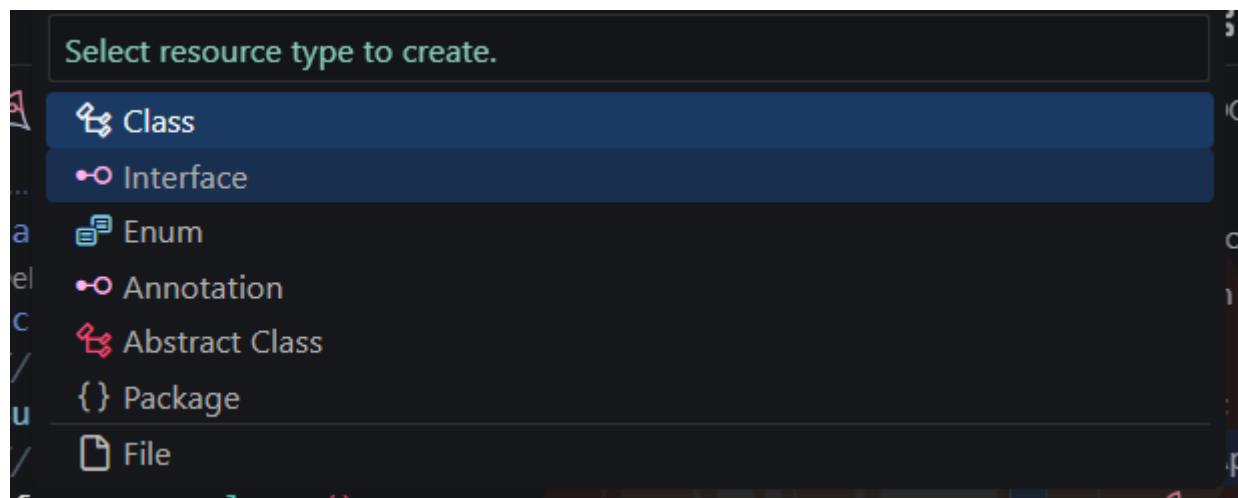
- **Proyectos en Distintas Carpetas:**
  - Es recomendable organizar cada proyecto en su propia carpeta. Esto no solo mantiene el trabajo ordenado, sino que también ayuda a evitar confusiones entre archivos y configuraciones.
- **Crear un Nuevo Proyecto:**
  - Al iniciar un nuevo proyecto en Java, se crea un directorio con una estructura predefinida que incluye un directorio **src** (source) y un archivo **App.java**.
  - El archivo **App.java** actúa como el punto de entrada del proyecto, desde donde se puede compilar y ejecutar el programa.
- **Estructura del Proyecto:**
  - **src**: Contiene el código fuente del proyecto.
  - **App.java**: Archivo principal que puede contener el método **main**, que es el punto de inicio del programa.



- **Creación de Archivos Dentro del Proyecto:**

- Se puede crear un nuevo archivo en la carpeta `src`, que puede ser de diferentes tipos, como `Class`, `Interface`, o `Enum`.
- Para este ejemplo, se seleccionó `Class` y se nombró `BucleFor`.

Es importante recordar que en Java, las clases deben comenzar con una letra mayúscula, lo cual es una convención estándar en la programación orientada a objetos.



### Compilación del Código For

- **Trabajo con la Clase `BucleFor`:**

- Después de crear la clase `BucleFor`, se comenzó a trabajar en el código dentro de ella, implementando un bucle `for` básico para ilustrar su funcionamiento.

```
App.java src 2
src > App.java > App > main(String[])
Comment Code | Run | Debug
1 public class App {
2 public static void main(String[] args) throws Exception {
3 BucleFor
4 }
5 }
6 I

BucleFor.java src
src > BucleFor.java > BucleFor
Comment Code |
1
2
3
4
5
6
7
8
9
10
11
12
13
public class BucleFor {
 // método : + - + - + - + -
 public void signoAlternado(){
 for (int i = 1; i < 10; i++) {
 if (i%2==0)
 System.out.println("-");
 else
 System.out.println("+");
 }
 }
}
```

- **Ejemplo de Código Compilado:**

- Una vez implementado el código, se compila para verificar que no haya errores de sintaxis y que el bucle funcione como se espera.

```
App.java X BucleFor.java
src > App.java > ...
1 public class App {
2 Run | Debug
3 public static void main(String[] args) throws Exception {
4 //declarar
5 BucleFor bf;
6 //instanciar
7 bf = new BucleFor();
8 //llamar el metodo
9 bf.signoAlterno();
10 }
11 }
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL ... Run: App +

```
Celeste ➜ \OneDrive\SAKI\progra\proyec2\Bucle2 ➔ (main)
Home ➜ /usr/bin/env C:\\Program\\ Files\\Java\\jre-1.8\\bin\\java.exe
\OneDrive\\SAKI\\progra\\proyec2\\Bucle2\\bin App
● FOR:
+ - + - + - + - +
```

## Errores y Depuración

- **Identificación de Errores:**

- Durante la compilación, es posible que se encuentren errores. Es esencial aprender a leer y entender los mensajes de error que proporciona el compilador para corregirlos rápidamente.

```
Comment code | 1 public class BucleFor { 2 // método : + - + - + - + - 3 public void signoAlterno(){ 4 System.out.println("FOR: "); 5 for (int i = 1; i < 10; i++) { 6 if(i%2==0) 7 System.out.print("- "); 8 else 9 System.out.print("+ "); 10 } 11 } 12 }
```

TERMINAL PROBLEMS 4 OUTPUT DEBUG CONSOLE PORTS ... Run: App + - X

```
pat_mic
prjBucle02 12:11 /usr/bin/env C:\Program\Files\Java\jdk-21\bin\java.exe -XX:+ShowCodeDetailsInExceptionMessages -cp C:\pat_mic\pat_dev\PRG_II_CC\prjBucle02\bin App
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
Syntax error on token "1", invalid VariableDeclaratorId
i cannot be resolved to a variable
i cannot be resolved to a variable
i cannot be resolved to a variable

at BucleFor.signoAlterno(BucleFor.java:6)
at App.main(App.java:8)
```

## Siguiente Ejemplo - Secuencia Matemática

- **Entendiendo el Patrón:**

- En algunos ejercicios, es necesario identificar patrones matemáticos para escribir un bucle eficiente.
- Ejemplo: En una secuencia de símbolos como ++++++....., es importante notar que:
  - Los números impares están representados por +.
  - Los números pares están representados por -.
  - La cantidad de símbolos aumenta con cada iteración.

- **Compactación del Código:**

- Una vez entendido el patrón, se puede simplificar y compactar el código para que sea más eficiente y legible.

The screenshot shows a Java development environment with two open files:

- App.java:** A simple application that creates an instance of `BucleFor` and calls its `signoAlternoGenerativo()` and `signoAlternoGenerativoSecond()` methods.
- BucleFor.java:** A class containing two methods for printing alternating signs. The `signoAlternoGenerativo()` method prints a sequence of dashes and pluses, while the `signoAlternoGenerativoSecond()` method prints a sequence of pluses and dashes.

The terminal window at the bottom shows the execution of the program and its output:

```
⚡ Celeste ➜ /usr/bin/env C:\\Program Files\\Java\\jre-1.8\\bin\\java.exe -cp C:\\Users\\Home\\OneDrive\\SAKI\\progra\\proyec2\\Bucle2\\bin App
● FOR:
+ - + - + - + -
FOR:
- ++ --- ++++ ----- ++++++ -----
FOR:
- ++ --- ++++ ----- ++++++ +++++++
```

## Ejercicio: La Escalera

### • Construyendo una Escalera con Bucles:

- Este ejercicio consiste en usar bucles anidados para construir una escalera en la consola, donde cada nivel de la escalera es representado por líneas horizontales y verticales.

### • Niveles (Variable **n**):

- for (n):** Controla cuántos niveles tendrá la escalera. A medida que aumenta **n**, se agrega un nuevo nivel a la escalera.

- Ejemplo:
  - Nivel 1: |\_
  - Nivel 2: |\_|
  - Nivel 3: |||

- Nivel 1: |\_
- Nivel 2: |\_|
- Nivel 3: |||

### • Ancho de la Escalera (Variable **h**):

- for (h):** Controla el espacio entre los niveles de la escalera, aumentando la distancia a medida que se sube de nivel.

The screenshot shows a Java development environment with two code editors. The left editor contains `App.java` with the following code:

```

src > App.java > App > main(String[])
You, hace 4 minutos | 1 author (You)
1 public class App {
 Run | Debug
 public static void main(String[] args) throws Exception {
 //declarar
 BucleFor bf;
 //instanciar
 bf = new BucleFor();
 //Llamar el metodo
 bf.signoAlterno();
 bf.signoAlternoGenerativo();
 bf.signoAlternoGenerativoSecond();
 bf.escalera();
 }
}

```

The right editor contains `BucleFor.java` with the following code:

```

src > BucleFor.java > BucleFor > escalera()
1 public class BucleFor {
 public void signoAlternoGenerativoSecond(){
 }
 //metodo: escalera.
 public void escalera(){
 System.out.println("Escalera:");
 int nivel = 10;
 String sacalon= "|_";
 for (int i = 1; i < nivel; i++){
 for (int e = 1; e < i; e++){
 System.out.print(" ");
 }
 System.out.println(sacalon);
 }
 }
}

```

Below the code editors, there is a navigation bar with tabs: PROBLEMAS, SALIDA, CONSOLA DE DEPURACIÓN, TERMINAL, PUERTOS, and GITLENS. The 'CONSOLA DE DEPURACIÓN' tab is currently selected.

## Bucle While

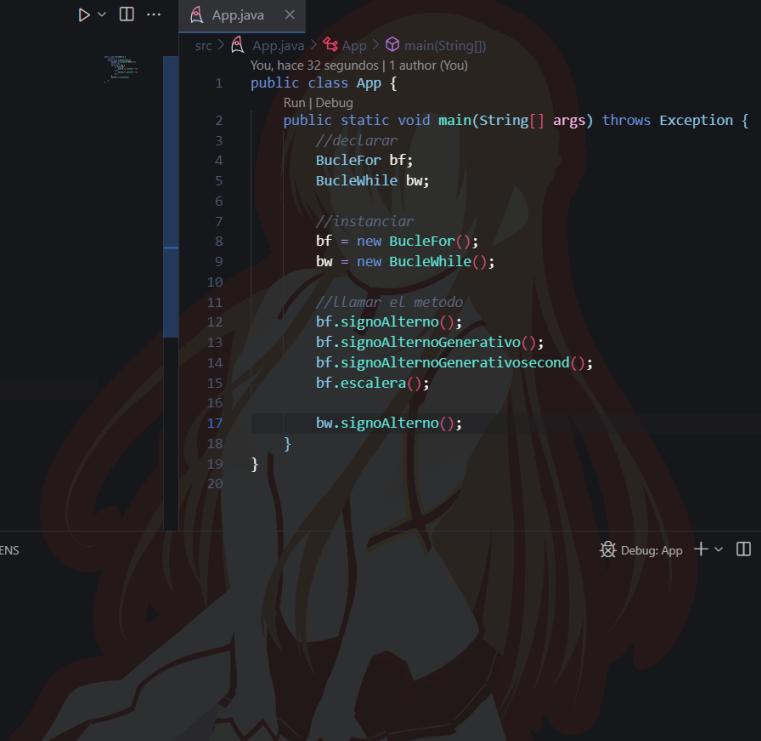
- **Conversión de `for` a `while`:**

- El bucle `while` se utiliza cuando no se sabe de antemano cuántas veces se debe repetir un bloque de código. A diferencia del `for`, en `while` la condición se evalúa antes de ejecutar el bloque de código.
- Ejemplo de conversión:

```

int i = 0;
while (i < 10) {
 System.out.println("Iteración: " + i);
 i++;
}

```



BucleWhile.java

```

src > BucleWhile.java > ...
1 public class BucleWhile {
2 //metodo: + - + - + - +
3 public void signoAlterno(){
4 System.out.println("While:");
5 int i = 1;
6 while (i < 10) {
7 if (i%2==0)
8 System.out.print("- ");
9 else
10 System.out.print("+ ");
11 i++;
12 }
13 System.out.println();
14 }
15 }
16
17

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS GITLENS

while:  
+ - + - + - + - +

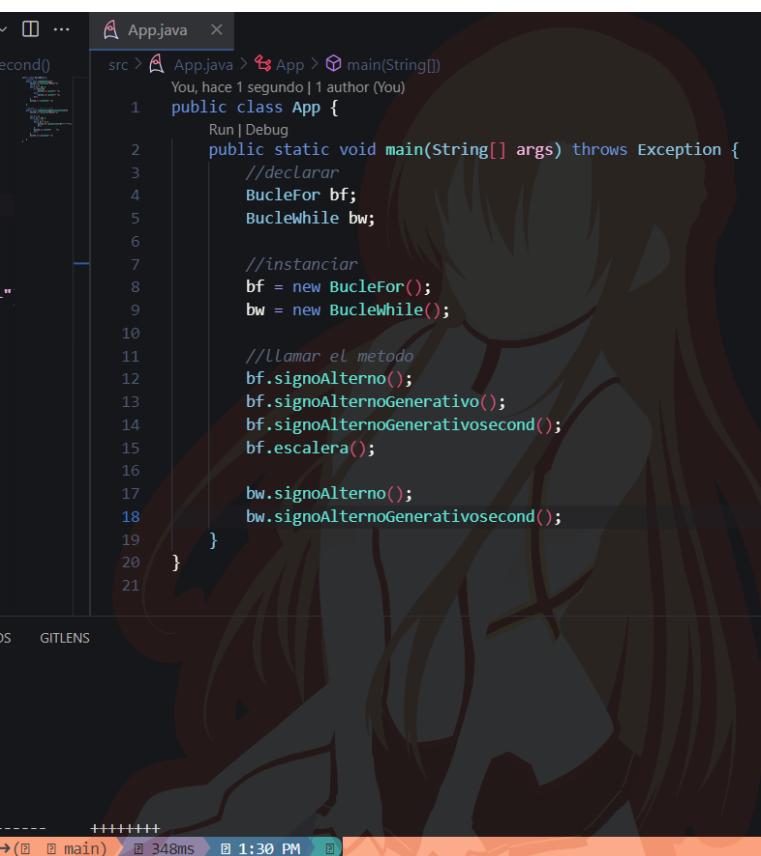
App.java

```

src > App.java > App > main(String[])
You, hace 32 segundos | 1 author (You)
1 public class App {
2 Run | Debug
3 public static void main(String[] args) throws Exception {
4 //declarar
5 BucleFor bf;
6 BucleWhile bw;
7
8 //instanciar
9 bf = new BucleFor();
10 bw = new BucleWhile();
11
12 //Llamar el metodo
13 bf.signoAlterno();
14 bf.signoAlternoGenerativo();
15 bf.signoAlternoGenerativoSecond();
16
17 bw.signoAlterno();
18 }
19
20

```

Debug: App



BucleWhile.java

```

src > BucleWhile.java > BucleWhile > signoAlternoGenerativoSecond()
1 public class BucleWhile {
2 //metodo: + - + + - - + + + -
3 public void signoAlternoGenerativoSecond(){
4 System.out.println("While:");
5
6 int i = 1;
7 while (i < 10) {
8 int s = 1;
9 while (s < i) {
10 System.out.print((i%2==0)? "- ":"+");
11 s++;
12 }
13 System.out.print(" ");
14 i++;
15 }
16 System.out.println(" ");
17 }
18
19
20

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS GITLENS

while:  
+ - + - + - + - +  
While:  
- + + - - + + + - - + + + + -

App.java

```

src > App.java > App > main(String[])
You, hace 1 segundo | 1 author (You)
1 public class App {
2 Run | Debug
3 public static void main(String[] args) throws Exception {
4 //declarar
5 BucleFor bf;
6 BucleWhile bw;
7
8 //instanciar
9 bf = new BucleFor();
10 bw = new BucleWhile();
11
12 //Llamar el metodo
13 bf.signoAlterno();
14 bf.signoAlternoGenerativo();
15 bf.signoAlternoGenerativoSecond();
16 bf.escalera();
17
18 bw.signoAlterno();
19 bw.signoAlternoGenerativoSecond();
20
21

```

Celeste

OneDrive\SAKI\progra\proyec2\Bucle2

main

348ms

1:30 PM

## Bucle Do-While

- Características:**

- El bucle **do-while** es similar al **while**, pero la principal diferencia es que el bloque de código se ejecuta al menos una vez antes de evaluar la condición.
- Esto es útil cuando se necesita ejecutar el código y luego decidir si se debe repetir.

- Sintaxis:**

```
int i = 0;
do {
 System.out.println("Iteración: " + i);
 i++;
} while (i < 10);
```

En resumen, dominar los bucles y sus variaciones (`for`, `while`, `do-while`) es esencial para escribir código eficiente y flexible. Además, la correcta organización del directorio del proyecto y la capacidad de depurar errores contribuyen a un flujo de trabajo más productivo y menos propenso a errores.

## Apuntes semana 4

---

### Clase 7

#### Paradigma

- Solventar problemas.

##### 1. **Conceptualizacion:**

- Idea de lo que se quiere realizar, se puede dibujar, diseñar, etc. Tener una idea del resultado.
- Como realizarlo.
- ¿Qué tengo? Ponerle nombre, concretar un significado.
- Solo los objetos se pueden hacer dos cosas: Características, acciones.
  - **Características:** propiedades. Almacenamiento de la información, cantidad, medir. (descripción clara del objeto). Recomendación: dibujarlo.
  - **Acciones:** métodos. Una acción es un verbo (jugar, correr, caminar...), que tenga sentido y vaya de acuerdo con las características mencionadas y con la conceptualización que se quiere, debe tener parámetros.
- No salir del tema, definir un límite, una parte de la información.
- No se invente.
- Solo lo necesario al momento de describir.
- Los parámetros podrían interpretarse como un límite.
  - **Parámetros:** dentro de las acciones Ej: Caminar(tiempo, lugar...) información extra que da las instrucciones, para que el programa haga lo que tu quieras. Se parece a las propiedades. Pero cambian.
- Definir si es público o privado/protector del objeto.
  - PUBLIC (+)
  - PRIVATE (-)

- PROTEC ()
  - FRENDLY (#)
- Definir si las características y los métodos es alguna de esas, eso depende del programador, tiene que tener una razón para saber cómo se va a comportar eso.

## 2. **UML:**

- *Significado:* Lenguaje de modelaje unificado.
- Es el ¿Qué?
- Colocar en una clase. (Diagrama de clase)
- Ejemplo:

### + CLASE

---

Edad: float

---

tieneOjos: bool

---

tipoCabello: String

---

..

- Colocar las ideas, para que se identifiquen, para que estas empiezan a tener una forma más sólida, después de colocar las características se escriben los métodos.
- Ejemplo N.-2:

### + CLASE

---

Edad: float

---

tieneOjos: bool

---

tipoCabello: String

---

..

bailar(cancion: String; tiempoMin: int; ritmo: String): void/String/boolean

---

tocar(cosaobjeto: String; tiempoMin: int): booleana

---

saltar(altura: int; cantidad: int): void

---

..

- Void: no interesa
- String: que da un resultado
- Boolean: saber solo si cumplió o no cumplió.

- *Interaccion con otros objetos:* se necesita un evento (algo que pasa por algo externo, interaccion/comunicacion entre dos objetos) esto se puede añadir como un cuerpo u objeto mas, esos son los eventos.

### + CLASE

---

Edad: float

---

tieneOjos: bool

---

tipoCabello: String

---

..

---

bailar(cancion: String; tiempoMin: int; ritmo: String): void/String/boolean

---

tocar(cosaobjeto: String; tiempoMin: int): booleana

---

saltar(altura: int; cantidad: int): void

---

..

---

sentir (nombre: Hombre)

- Se pueden poner en lineas extra o alli mismo, depende del diseñador.
- se debe utlizar un herramienta para Diagrama de clases.

### 3. Código (Java)

- Diagrama y codigo deben estar iguales.

- *Ejemplo:* Propiedades.

```
public class Mujer {
 private float edad;
 public boolean tieneOjos;
 private String tipoCabello;
 ...
}
```

- *Ejemplo N.- 2:* metodos.

```
protected String bailar (String cancion; int tiempoMin; String ritmo) {
 ...
 return "sjjsjmskdk..."
}
```

- *Deber*

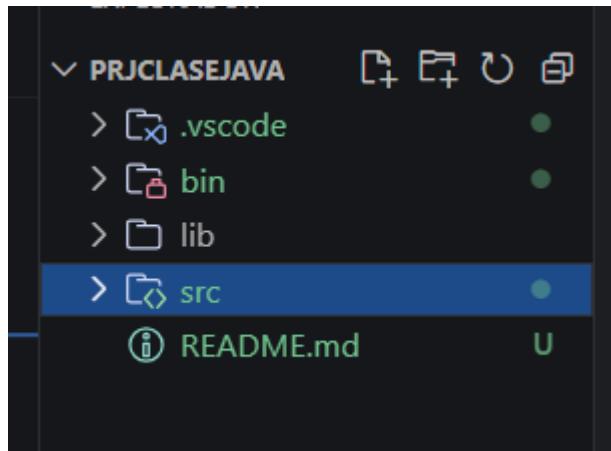
- Concretualizar un animal salvaje, a mano en una hoja.
- Debe tener 3 propiedades y 3 metodos.

## Clase 9

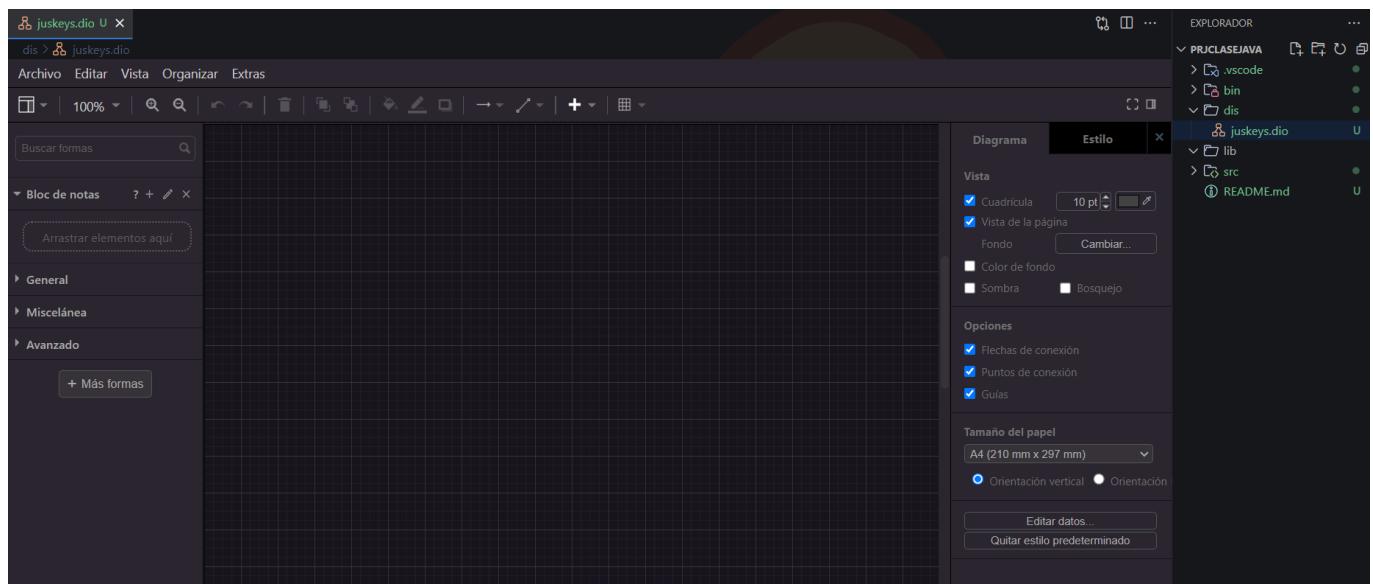
Comprobar que se tienen las siguientes extensiones:

- Herramientas:
  - drawn.io
  - excalidraw
  - etc.

Abrir un nuevo proyecto, abrir solo la carpeta DEL PROYECTO, no mas solo la carpeta, se tiene que ver asi:



abri una pestaña de drawn.io

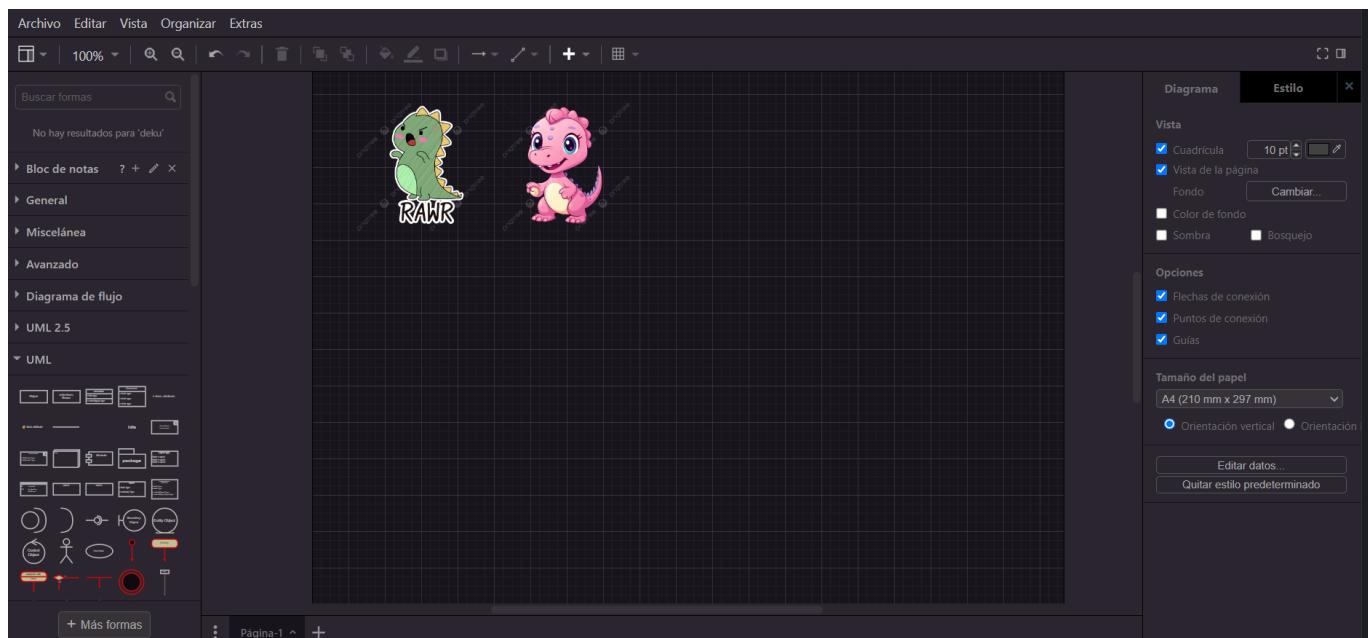


para los diseños abrir, en una nueva carpeta, nombre.dio (el "dio" es lo que hace que se transforme en ese formato)

### diagramas Caso de Uso

Es para definir lo que se quiere realizar, tener una imagen concreta

En este ejemplo se usaron dos imagenes:



Se tiene la imagen, (use) Donde se tienen varias ramas.

- relacion
- include
- externo

Pasa del diagrama de uso, interpretandolo, y pasandole a:

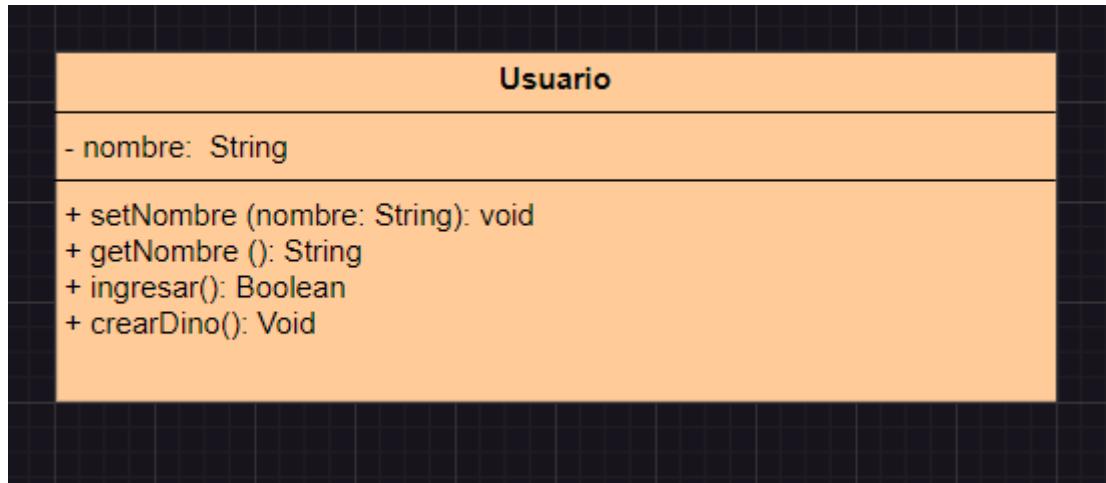
### Diagramas de Clase

- relacion
- asociacion
- herencia

(PARA EL PROYECTO SE DEBE TENER DIAGRAMA DE USO Y DIGRAMA DE CLASE)

Comprobar que Java compile que la todo funcione

crear la imagen y pasar al UML



Tras eso pasarlo al codigo

```
▼ Click here to ask Blackbox to help you code faster
public class Usuario {
 private String nombre;

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

 public String getNombre() {
 return nombre;
 }

 public Boolean ingresar() {
 System.out.println("listo para el ingreso");
 return true;
 }

 public void crearDino() {
 System.out.println("crear un dino");
 }
}
```

y ver que eso compile

The screenshot shows an IDE interface with the following details:

- File:** Usuario.java
- Project:** src > Usuario.java > Usuario > setNombre(String)
- Code:**

```
1 public class Usuario {
2 private String nombre;
3
4 public void setNombre(String nombre){
5 this.nombre = nombre.toUpperCase();
6 }
7
8 public String getNombre() {
9 return nombre;
10 }
11
12 public Boolean ingresar() {
13 System.out.println("listo para el ingreso");
14 return true;
15 }
16
17 public void crearDino() {
18 System.out.println("crear un dino");
19 }
20}
21}
```
- Terminal:** Shows the command line path and the output of the program.

```
Celeste ➤ ② ③ ④ \OneDrive\SAKI\progra\prjClaseJava ➤ → (② ③ mai
⚡Home ➤ /usr/bin/env C:\\Program\\ Files\\Java\\jre-1.8\\bin\\j
e\\OneDrive\\SAKI\\progra\\prjClaseJava\\bin App
```

listo para el ingreso  
crear un dino

## Apuntes semana 5

### Clase 10

#### Ejemplo de Cómo Trabaja TUTI

**Actor:** En el contexto de TUTI, un "actor" es cualquier entidad que interactúa con el sistema. Esto puede incluir usuarios, clientes, o cualquier agente externo que realice una acción dentro del sistema.

- **Ejemplo:** En TUTI, un cliente (actor) puede **seleccionar un producto de los estantes**. Este es un ejemplo de cómo un actor utiliza el sistema para realizar una acción específica.
- **Ejemplo 2:** El cliente también puede **pagar los productos seleccionados**. En este caso, el **cajero** es quien recibe el pago. El sistema de caja asiste al cajero en el proceso de pago, pero su uso inadecuado puede causar dificultades.

**App:** Refleja el sistema en sí mismo, que en este caso es TUTI.

## Identificación de Procesos y Situaciones

### 1. Procesos de Devoluciones:

- Permiten a los clientes devolver productos. Este proceso puede incluir la verificación de la razón de la devolución y el ajuste del inventario.

### 2. Procesos de Cancelación:

- Requiere **autorización de un supervisor**. La cancelación puede involucrar la anulación de una transacción y la actualización de registros.

### 3. Proceso de Enviar Comprobante Electrónico:

- Implica enviar un comprobante de compra al cliente de manera electrónica, a menudo por correo electrónico.

## Nuevas Funcionalidades

### 1. Productos en Línea para Envío a Casa:

- Los clientes pueden seleccionar productos para ser enviados a su hogar.
- **Pago de Productos Seleccionados:** Obligatorio para completar la compra.
- **Pagar con Tarjeta de Crédito/Débito:** Opcional, proporcionando flexibilidad en los métodos de pago.

## Herencia en Programación

### Aspectos Comunes:

- Permite la reutilización de código, optimiza la gestión y protege la información al encapsular datos y comportamientos comunes en una clase base.

### 1. Conceptualización:

- Identificar y definir las características comunes entre diferentes clases que pueden ser agrupadas en una clase base.

### 2. UML (Unified Modeling Language):

- Usado para representar visualmente la estructura de clases y las relaciones entre ellas. Se emplean diagramas de clases para modelar la herencia y otros aspectos de la programación orientada a objetos.

- **Privacidad:**

- Las propiedades y métodos en una clase base pueden ser definidos como **private**, **protected**, o **public**, controlando su accesibilidad desde otras clases.

- **Ejemplo:**

- Una clase **Producto** puede ser una clase base para **ProductoElectrónico** y **ProductoAlimenticio**, cada una con atributos y comportamientos específicos.

## Generalización

### Palabras Clave:

- **ES UN:** Utilizada para describir la relación de herencia, donde una clase "es un" tipo específico de otra clase. Ejemplo: **Coche es un Vehículo**.
- **PUEDE SER:** Utilizada para describir la relación de generalización, donde una clase "puede ser" uno de varios tipos. Ejemplo: **Transacción puede ser Compra o Venta**.

## Aspectos a Considerar

- **Valor Numérico:**

- Cuando se trabaja con valores numéricos, es importante definir su tipo de dato (**int**, **float**, **double**, etc.) y considerar cómo se utilizarán (operaciones aritméticas, comparaciones).

- **Diseño:**

- Crear un diseño conceptual antes de implementar el código. Esto puede incluir diagramas y descripciones detalladas del sistema y sus componentes.

En resumen, la programación en TUTI y otros sistemas involucra el uso de actores y procesos para gestionar la interacción del usuario con el sistema. La herencia y la generalización son conceptos clave en la programación orientada a objetos que permiten una estructura más organizada y eficiente del código.

## Clase 11

### Constructor y Sobrecarga en Programación

#### Constructor

- **Definición:** Un constructor es un método especial en una clase que se utiliza para inicializar objetos de esa clase. Su nombre es el mismo que el de la clase y no tiene un tipo de retorno.
- **Visibilidad:**
  - **Público:** El constructor puede ser accedido desde cualquier otra clase.
  - **Protegido:** El constructor solo puede ser accedido dentro de la misma clase y por clases derivadas.
- **Características:**

- **Por Defecto:** Si no se define un constructor en una clase, Java proporciona un constructor por defecto sin argumentos. Este constructor inicializa las variables de instancia a sus valores predeterminados.
  - **Inicialización:** Se utiliza para establecer valores iniciales para las variables de instancia del objeto.
  - **Ejecución Automática:** Se llama automáticamente cuando se crea un nuevo objeto de la clase. No es necesario invocar el constructor explícitamente.
- **Ejemplo:**

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

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

 // Método para mostrar información del coche
 public void mostrarInfo() {
 System.out.println("Modelo: " + modelo + ", Año: " + año);
 }
}

public class Main {
 public static void main(String[] args) {
 // Creación de un objeto Coche, constructor se llama automáticamente
 Coche miCoche = new Coche("Toyota", 2020);
 miCoche.mostrarInfo();
 }
}
```

## Sobrecarga de Constructores

- **Definición:** La sobrecarga de constructores ocurre cuando una clase tiene más de un constructor con diferentes listas de parámetros. Esto permite crear objetos de una clase de diferentes maneras, dependiendo de los argumentos proporcionados.
  - **Características:**
    - **Diferentes Parámetros:** Cada constructor sobrecargado debe tener una lista de parámetros diferente (número, tipo, o ambos).
    - **Flexibilidad:** Proporciona flexibilidad en la creación de objetos, permitiendo diferentes formas de inicialización.
- **Ejemplo:**

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

 // Constructor con dos parámetros
 public Coche(String modelo, int año) {
 this.modelo = modelo;
 this.año = año;
 this.color = "No especificado"; // Valor por defecto
 }

 // Constructor con tres parámetros (sobrecargado)
 public Coche(String modelo, int año, String color) {
 this.modelo = modelo;
 this.año = año;
 this.color = color;
 }

 // Método para mostrar información del coche
 public void mostrarInfo() {
 System.out.println("Modelo: " + modelo + ", Año: " + año + ", Color:
" + color);
 }
}

public class Main {
 public static void main(String[] args) {
 // Usando el primer constructor
 Coche miCoche1 = new Coche("Honda", 2019);
 miCoche1.mostrarInfo();

 // Usando el segundo constructor
 Coche miCoche2 = new Coche("Ford", 2021, "Rojo");
 miCoche2.mostrarInfo();
 }
}
```

## Resumen

- **Constructor:** Método especial con el mismo nombre que la clase, usado para inicializar objetos. Siempre es público o protegido, y se ejecuta automáticamente cuando se crea un objeto.
- **Sobrecarga:** Permite tener múltiples constructores en una clase con diferentes parámetros, proporcionando varias formas de inicializar objetos de la clase.

Estos conceptos permiten un control más preciso sobre cómo se crean y configuran los objetos en la programación orientada a objetos, facilitando la flexibilidad y reutilización del código.

## Clase 12

## Variables, String, Scanner, Arrays y Operadores en Java

### Variables

- **Definición:** Las variables son contenedores para almacenar datos que pueden ser modificados durante la ejecución del programa.
- **Reglas para Nombres de Variables:**
  - **Letras:** Los nombres de variables pueden contener letras (A-Z, a-z).
  - **Inicio en Minúsculas:** Los nombres de variables suelen comenzar con una letra minúscula y usan notación camelCase para separar palabras (ej. `miVariable`).
  - **Sin Espacios:** No pueden contener espacios en blanco.
  - **Guion Bajo:** Se permite el uso de guiones bajos (\_) para separar palabras (ej. `mi_variable`), aunque no es común en notación camelCase.
  - **Números:** Pueden incluir números, pero no deben comenzar con ellos (ej. `variable1`, no `1variable`).
  - **Símbolo de Dólar:** También se permite el uso del símbolo de dólar (\$) (ej. `variable$`).
  - **Tipos:** Las variables pueden ser de tipo primitivo (int, char, etc.) o de tipo no primitivo (String, arrays, etc.).
- **Ejemplo:**

```
int edad = 25;
double salario = 50000.50;
String nombreCompleto = "Juan Pérez";
```

### String

- **Definición:** `String` es una clase en Java que representa una secuencia de caracteres.
- **Operaciones Comunes:**
  - **Obtener un Carácter:** Utiliza el método `charAt(int index)` para obtener el carácter en una posición específica.
  - `String texto = "Hola";
char letra = texto.charAt(2); // 'l'`
  - **Longitud:** Usa el método `length()` para obtener la longitud de la cadena.
  - `int longitud = texto.length(); // 4`

- **Comparación:** Usa `equals(String otroString)` para comparar dos cadenas.

```
boolean esIgual = texto.equals("Hola"); // true
```

## Scanner

- **Definición:** `Scanner` es una clase en Java utilizada para leer entradas de datos.
- **Usos:**

- **Leer un Valor:** Puedes leer diferentes tipos de datos (`int`, `double`, `String`) desde la entrada estándar (teclado) u otras fuentes como archivos.

```
Scanner scanner = new Scanner(System.in);
int numero = scanner.nextInt(); // Lee un entero
String texto = scanner.nextLine(); // Lee una línea de texto
```

- **Entradas Secuenciales:** Permite leer datos de manera secuencial desde diferentes fuentes.
- **Librería:** Importar la clase con `import java.util.Scanner;`.

## Arrays

- **Definición:** Un array es una estructura de datos que almacena múltiples valores del mismo tipo en una sola variable.
- **Modos de Creación:**

- **Declaración y Inicialización:**

```
int[] numeros = new int[5]; // Array de enteros con 5 elementos
```

- **Inicialización Directa:**

```
int[] edades = {25, 30, 35}; // Array con valores iniciales
```

- **EOF (End Of File):** Cuando se lee datos hasta el final del archivo, se utiliza para saber que no hay más datos disponibles.
- **Ejemplo:**

```
String[] nombres = {"Ana", "Luis", "Carlos"};
for (int i = 0; i < nombres.length; i++) {
```

```
 System.out.println(nombres[i]);
 }
```

## Operadores

- **Tipos de Operadores:**

- **Aritméticos:** +, -, \*, /, %.
- **Relacionales:** ==, !=, >, <, >=, <=.
- **Lógicos:** &&, ||, !.
- **Asignación:** =, +=, -=, \*=, /=, %=.
- **Incremento/Decremento:** ++, --.

- **Ejemplo:**

```
int a = 10;
int b = 20;
int suma = a + b; // 30
boolean esMayor = a > b; // false
```

## Ejercicio

- **Propósito:** Los ejercicios ayudan a validar el código y garantizar que el programa funcione correctamente.

- **Ejemplo de Código para Validación:**

```
public class EjemploValidacion {
 public static void main(String[] args) {
 Scanner scanner = new Scanner(System.in);
 System.out.print("Introduce tu edad: ");
 int edad = scanner.nextInt();
 if (edad >= 18) {
 System.out.println("Eres mayor de edad.");
 } else {
 System.out.println("Eres menor de edad.");
 }
 }
}
```

- **Objetivo:** Asegurarse de que el código compile y ejecute correctamente para validar su funcionalidad.

Estos conceptos son fundamentales para el desarrollo de programas en Java, y comprender cómo funcionan te permitirá escribir código más eficiente y efectivo.

# Apuntes semana 6

## Clase 13

### Interfaz y UML en Programación

#### Interfaz

- **Definición:** Una interfaz en Java es una abstracción que define un conjunto de métodos que una clase debe implementar. No puede contener implementación de métodos (a menos que se trate de métodos estáticos o predeterminados en interfaces).
- **Características:**
  - **Abstracción:** Permite definir métodos que deben ser implementados por las clases que heredan la interfaz.
  - **Implementación:** Una clase que implementa una interfaz debe proporcionar la implementación de todos los métodos declarados en la interfaz.
  - **Uso de Interfaces:** Facilita el diseño de aplicaciones basadas en el comportamiento en lugar de en la estructura de las clases.
- **Ejemplo:**

```
interface Vehiculo {
 void acelerar();
 void frenar();
}

class Coche implements Vehiculo {
 @Override
 public void acelerar() {
 System.out.println("Coche acelerando...");
 }

 @Override
 public void frenar() {
 System.out.println("Coche frenando...");
 }
}

public class Main {
 public static void main(String[] args) {
 Vehiculo miCoche = new Coche();
 miCoche.acelerar();
 miCoche.frenar();
 }
}
```

- **Definición:** UML es un lenguaje de modelado estándar utilizado para especificar, visualizar, construir y documentar artefactos de sistemas de software.
- **Tipos de Diagramas:**

- **Estáticos:**

- **Diagrama de Clases:** Representa la estructura del sistema en términos de clases, atributos y relaciones.

- **Ejemplo:**

```
+-----+
| Coche |
+-----+
| - modelo: String |
| - año: int |
+-----+
| + acelerar() |
| + frenar() |
+-----+
```

- **Diagrama de Objetos:** Muestra instancias específicas de clases y sus relaciones.

- **Dinámicos:**

- **Diagrama de Secuencia:** Muestra cómo los objetos interactúan entre sí en un flujo de eventos a lo largo del tiempo.
    - **Diagrama de Actividades:** Representa el flujo de actividades y operaciones en el sistema.

- **Proceso de Modelado:**

- **Bosquejar Datos:** Crear diagramas que representen la estructura y el comportamiento del sistema.
  - **Pruebas:** Validar los diagramas para asegurar que representan correctamente los requisitos del sistema.

- **Ejemplo:**

- **Actor "Bibliotecario":** Define un actor en un diagrama de casos de uso y sus funciones hacia el sistema (ej. agregar libros, prestar libros).
    - **Actor "Estudiante":** Define las acciones y variables asociadas a un estudiante (ej. solicitar libros, devolver libros).

## Herencia

- **Definición:** La herencia es un mecanismo en programación orientada a objetos que permite a una clase (subclase) heredar atributos y métodos de otra clase (superclase).
- **Características:**

- **Especialización:** Permite a una subclase especializarse en la funcionalidad de la superclase.
  - **Extensión:** Las subclases pueden extender la funcionalidad de la superclase añadiendo nuevos métodos o atributos.
- **Uso:**
    - **extends:** La palabra clave utilizada para heredar de una superclase.

```
class Animal {
 void comer() {
 System.out.println("Animal comiendo...");
 }
}

class Perro extends Animal {
 void ladrar() {
 System.out.println("Perro ladrando...");
 }
}
```

- **implements:** La palabra clave utilizada para implementar una interfaz.

```
interface Volador {
 void volar();
}

class Pajaro implements Volador {
 @Override
 public void volar() {
 System.out.println("Pájaro volando...");
 }
}
```

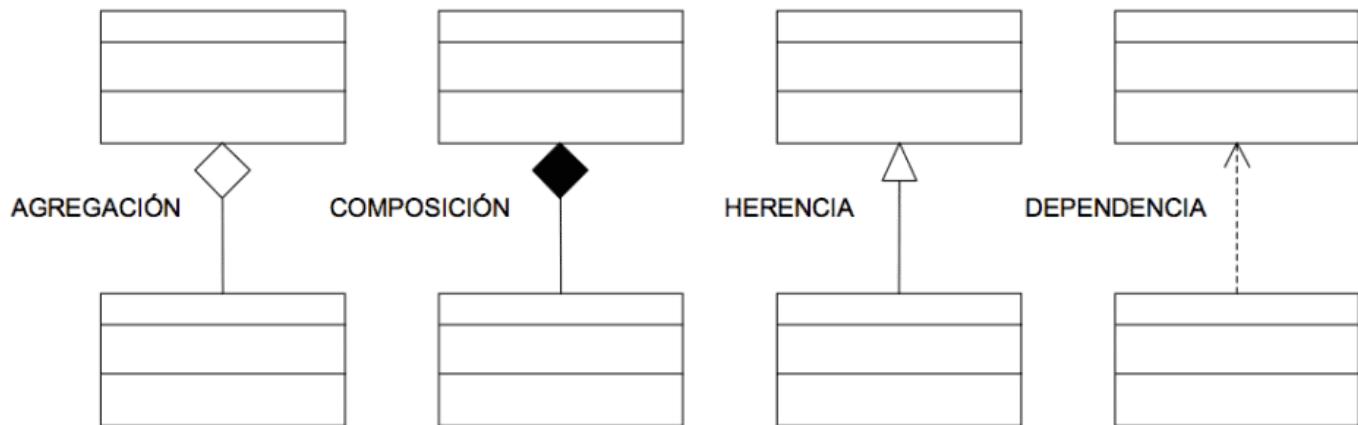
- **Relaciones entre Clases:**

- **Especialización:** Representa la relación donde una subclase es un tipo más específico de una superclase.
- **Generalización:** Representa la relación donde una superclase es un tipo más general que una subclase.

- **Inclusión y Exclusión:**

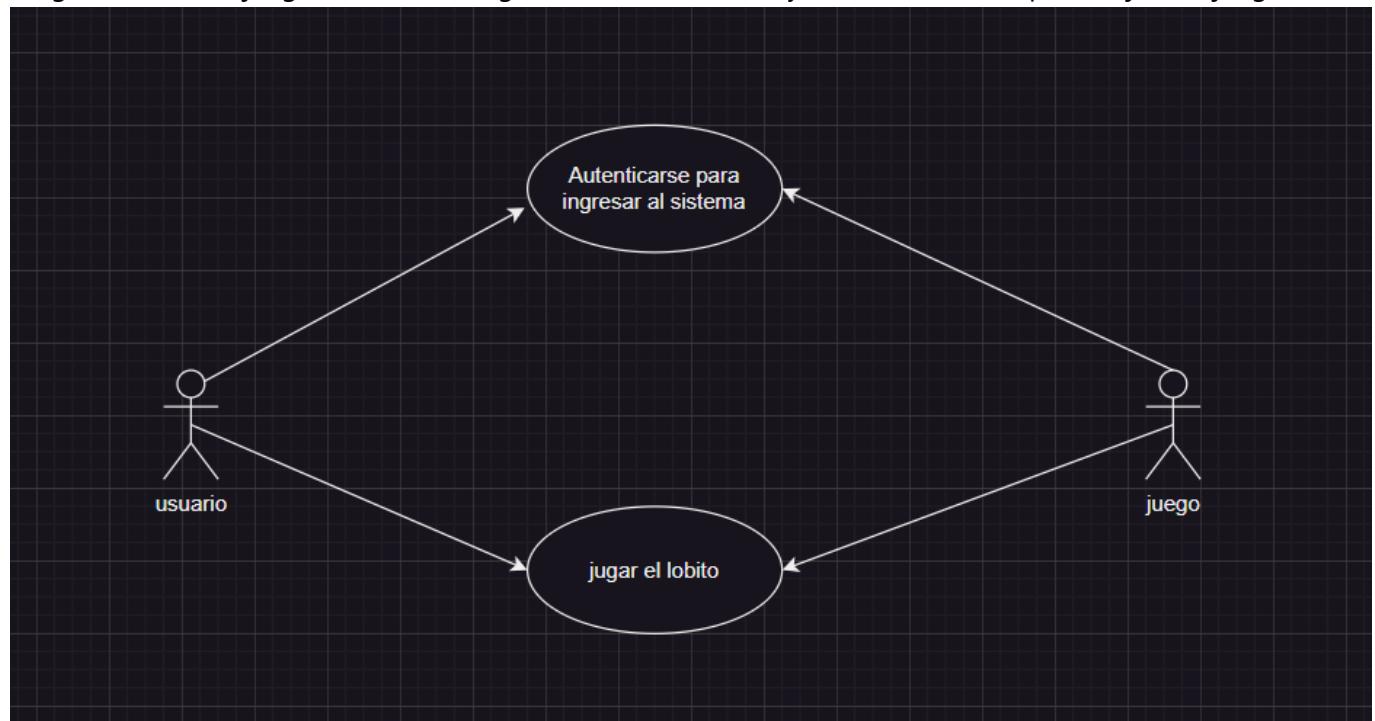
- **include:** Relación entre casos de uso donde un caso de uso siempre incluye el comportamiento de otro.
- **extend:** Relación opcional entre casos de uso donde un caso de uso puede extender el comportamiento de otro.

Estos conceptos son fundamentales para diseñar y entender sistemas orientados a objetos, ayudando a crear aplicaciones modulares y reutilizables.



## Clase 14

Diagrama de clase juego del lobo En el grafico se ve al usuario y al sistema (no los personajes del juego)



## Apuntes semana 7

---

### Clase 15

#### Scanner en Java: Estático vs. Dinámico

##### Scanner Estático

- **Definición:** Un **Scanner** estático en Java es una instancia única de la clase **Scanner** que se usa a lo largo del programa para leer datos de una fuente específica, como el teclado.
- **Características:**
  - **Instancia Única:** No se puede clonar; hay una sola instancia que se utiliza en todo el programa.

- **Uso de Recursos:** Ideal para situaciones en las que solo necesitas una fuente de entrada (ej. leer datos del teclado).
- **Ejemplo:**

```
import java.util.Scanner;

public class Main {
 private static final Scanner scanner = new Scanner(System.in);

 public static void main(String[] args) {
 System.out.print("Introduce tu nombre: ");
 String nombre = scanner.nextLine();
 System.out.println("Hola, " + nombre);
 }
}
```

- **Ventajas:**

- **Eficiencia:** Utiliza menos recursos ya que solo hay una instancia del **Scanner**.
- **Simplicidad:** Menos propenso a errores relacionados con múltiples instancias.

## Scanner Dinámico

- **Definición:** Un **Scanner** dinámico puede ser creado y destruido en diferentes partes del programa, permitiendo múltiples instancias que pueden leer datos desde diferentes fuentes.
- **Características:**

- **Instancias Múltiples:** Se pueden crear varias instancias de **Scanner**, cada una con su propia fuente de entrada.
- **Flexibilidad:** Permite leer datos de diferentes fuentes o leer datos en diferentes momentos.
- **Ejemplo:**

```
import java.util.Scanner;

public class Main {
 public static void main(String[] args) {
 Scanner scanner1 = new Scanner(System.in);
 Scanner scanner2 = new Scanner(System.in);

 System.out.print("Introduce tu edad: ");
 int edad = scanner1.nextInt();
 scanner2.nextLine(); // Consumir el salto de línea pendiente

 System.out.print("Introduce tu ciudad: ");
 String ciudad = scanner2.nextLine();

 System.out.println("Edad: " + edad);
 System.out.println("Ciudad: " + ciudad);
 }
}
```

```
}
```

- **Ventajas:**

- **Versatilidad:** Permite la creación de múltiples instancias para diferentes propósitos.
- **Adaptabilidad:** Puede ser útil para leer datos desde diferentes fuentes de entrada en paralelo.

## Comandos de Escape

- **Definición:** Los comandos de escape son secuencias de caracteres especiales que se usan en las cadenas de texto para representar caracteres que no se pueden ingresar directamente.

- **Comandos Comunes:**

- **\r (Retorno de Carro):** Mueve el cursor al principio de la línea actual. Se usa para sobrescribir el contenido en la misma línea.

```
System.out.print("Progreso: 50%\rProgreso: 75%");
```

- **\n (Salto de Línea):** Mueve el cursor a la siguiente línea, creando un salto de línea.

```
System.out.println("Primera línea\nSegunda línea");
```

## Refactorización

- **Definición:** Refactorizar es el proceso de reorganizar el código existente para mejorar su estructura y legibilidad sin alterar su funcionalidad.

- **Objetivos:**

- **Legibilidad:** Hacer el código más fácil de entender.
- **Eficiencia:** Mejorar el rendimiento y reducir la complejidad.
- **Mantenimiento:** Facilitar el mantenimiento y la expansión futura del código.

- **Proceso:**

- **Identificar Áreas de Mejora:** Buscar partes del código que sean difíciles de leer o de mantener.
- **Aplicar Cambios:** Reorganizar, simplificar, y limpiar el código.
- **Pruebas:** Asegurarse de que los cambios no afectan la funcionalidad del programa.

- **Ejemplo:**

```
// Antes de refactorizar
public class Ejemplo {
 public void procesarDatos(int a, int b) {
```

```
if (a > b) {
 System.out.println("A es mayor que B");
} else {
 System.out.println("A no es mayor que B");
}
}

// Después de refactorizar
public class Ejemplo {
 public void procesarDatos(int a, int b) {
 String mensaje = a > b ? "A es mayor que B" : "A no es mayor que B";
 System.out.println(mensaje);
 }
}
```

- **Resultado:**

- **Código más limpio y entendible.**
- **Facilita futuras modificaciones y mantenimiento.**

Estos conceptos te ayudarán a gestionar la entrada de datos, a mejorar la calidad del código y a estructurar tu aplicación de manera eficiente.

## Clase 17

### Revisión del Proyecto

#### Aspectos del Proyecto

##### 1. Presentación

- **Carátula:**
  - **Contenido:** Debe incluir nombres de los integrantes del equipo, el tema del proyecto, la idea principal y cómo funciona.
  - **Formato:** La carátula debe ser clara y profesional, proporcionando una visión general concisa.
  - **Extensión:** No debe exceder las 10 líneas de texto que expliquen el objetivo y el alcance del proyecto.
- **Prototipo del Proyecto:**
  - **Descripción:** Incluir una representación visual o conceptual del proyecto. Esto puede ser un esquema, boceto o un prototipo funcional.
  - **Dispositivo Externo:** Debe identificar y describir el dispositivo externo utilizado en el proyecto. ¿Cómo se integra y qué papel juega en el funcionamiento del sistema?
- **Páginas de Presentación:**
  - **Número de Páginas:** La presentación debe abarcar un total de 5 páginas.
  - **Contenido:** Deberá incluir la introducción, objetivos, alcance, metodología, y resultados esperados o obtenidos.

##### 2. Diagrama de Casos de Uso

- **Tamaño y Complejidad:**

- **Consideración:** Debe ser lo suficientemente claro para entender los principales casos de uso sin ser excesivamente complejo.
- **Enfoque:** Representar las interacciones entre actores y el sistema de manera que se puedan identificar fácilmente los requisitos funcionales del proyecto.

### 3. Diagrama de Clases

- **Diseño del Software:**

- **Creación de Clases:** Definir y crear las clases necesarias para el proyecto.
- **Representación:** El diagrama debe reflejar las relaciones entre clases, atributos, métodos y asociaciones.
- **Propósito:** Facilitar la comprensión de la estructura del software y cómo las diferentes partes del sistema interactúan entre sí.

### 4. Código

- **Bosquejo de los Diagramas de Clase:**

- **Descripción:** Presentar un bosquejo del código basado en los diagramas de clase. No es necesario que el código esté completamente funcional en esta etapa.
- **Navegabilidad:** El código debe estar organizado de manera que sea navegable y comprensible, con comentarios y estructuras que reflejen el diseño conceptual.

### Entrega

- **Fecha de Entrega:** Miércoles, 12/06/2024.

- **Preparación:** Asegurarse de que todos los documentos y el código estén listos para la entrega en la fecha estipulada. Revisar y validar que todos los requisitos se hayan cumplido.

Este resumen te proporciona una guía clara sobre lo que debe incluirse en cada sección del proyecto y cómo estructurar la presentación y los diagramas de manera efectiva.

## Apuntes semana 8

---

### Clase 18

#### Paquetes y Clases Abstractas

##### Paquetes

- **Definición:** Los paquetes en Java son una forma de organizar y agrupar clases relacionadas para una mejor gestión del código y evitar conflictos de nombres.
- **Propósito:** Facilitan la identificación y el manejo de temas relacionados dentro del proyecto.
- **Ejemplo:**

```
package com.ejemplo.proyecto;

public class MiClase {
```

```
// Código de la clase
}
```

## Clases Abstractas

- **Definición:** Las clases abstractas son clases que no se pueden instanciar directamente. Se utilizan como base para otras clases y pueden contener métodos abstractos (sin implementación) que las subclases deben implementar.
- **Características:**
  - **Métodos Abstractos:** Métodos sin cuerpo, que deben ser implementados por las subclases.
  - **Métodos Concretos:** Pueden tener métodos con implementación que las subclases heredan.
- **Ejemplo:**

```
public abstract class Animal {
 public abstract void hacerSonido();

 public void dormir() {
 System.out.println("El animal está durmiendo.");
 }
}

public class Perro extends Animal {
 @Override
 public void hacerSonido() {
 System.out.println("El perro hace guau.");
 }
}
```

## Examen

### Conceptos Claves

#### 1. Paquetes

- Organización del código.
- Agrupación de clases relacionadas.

#### 2. Clases Abstractas

- No se pueden instanciar.
- Contienen métodos abstractos y concretos.

#### 3. Interfaces

- Definen un contrato que las clases deben seguir.
- Los métodos en una interfaz son públicos y abstractos por defecto.
- **Ejemplo:**

```
public interface IComportamientoNatural {
 void comer();
 void dormir();
}
```

#### 4. Herencia

- Una clase hija hereda características (atributos y métodos) de una clase padre.
- **Ejemplo:**

```
public class Animal {
 public void respirar() {
 System.out.println("Respirando...");
 }
}

public class Perro extends Animal {
 public void ladrar() {
 System.out.println("El perro está ladrando.");
 }
}
```

### Relaciones entre Clases

#### 1. Asociación

- Relación entre dos clases en la que una clase utiliza o tiene una referencia a la otra.
- **Ejemplo:** Un propietario tiene una mascota, pero una mascota no puede existir sin un propietario.

#### 2. Composición

- Relación en la que una clase contiene varias instancias de otra clase. La vida de la clase contenida está ligada a la clase contenedora.
- **Ejemplo:** Un coche tiene varias ruedas. Las ruedas no pueden existir independientemente del coche.

#### 3. Agregación

- Relación en la que una clase puede tener varias instancias de otra clase, pero la existencia de estas instancias no está ligada a la existencia de la clase contenedora.
- **Ejemplo:** Un coche puede tener varias ruedas, pero las ruedas pueden existir independientemente del coche.

#### 4. Dependencia

- Relación en la que una clase utiliza otra clase, pero la segunda clase no es necesaria para la existencia de la primera.

- **Ejemplo:** Un coche puede utilizar un GPS, pero el GPS no es esencial para el funcionamiento del coche.

## Interfaz

- **Definición:** Una interfaz define un conjunto de métodos que una clase debe implementar. No se puede instanciar y se utiliza para definir acciones comunes entre clases.
- **Características:**
  - Todos los métodos son públicos y abstractos por defecto.
  - **Ejemplo:**

```
public interface IComportamientoNatural {
 void comer();
 void dormir();
}

public class Gato implements IComportamientoNatural {
 @Override
 public void comer() {
 System.out.println("El gato está comiendo.");
 }

 @Override
 public void dormir() {
 System.out.println("El gato está durmiendo.");
 }
}
```

Estos conceptos te ayudarán a estructurar y organizar tu código de manera efectiva, facilitando la reutilización y el mantenimiento.

## Clase 19

Claro, aquí está el resumen con las imágenes incluidas:

### Realización e Implementación de Interfaces

#### Realización e Implementación

- **Definición:**
  - **Realización e Implementación** son términos usados para describir la forma en que una clase cumple con un contrato definido por una interfaz.
  - **Realización y Implementación** son sinónimos en el contexto de interfaces. Ambos se refieren a la implementación de métodos definidos en una interfaz.
- **Características:**

- **Abstracción:** Las interfaces definen métodos sin implementar, proporcionando un contrato para las clases que las implementan.
- **Visibilidad Pública:** Todos los métodos en una interfaz son públicos y abstractos por defecto.  
No se puede crear una instancia directa de una interfaz.
- **Ejemplo:**

```

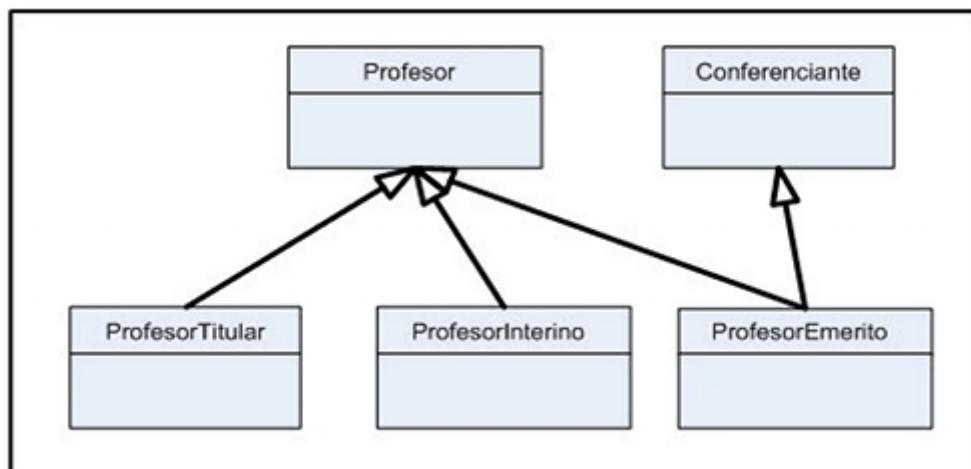
public interface IAnimal {
 void comer();
 void dormir();
}

public class Perro implements IAnimal {
 @Override
 public void comer() {
 System.out.println("El perro está comiendo.");
 }

 @Override
 public void dormir() {
 System.out.println("El perro está durmiendo.");
 }
}

```

- **Imagen:**



## Restricciones de Herencia

- **Herencia Simple:**

- En Java, una clase solo puede heredar de una sola clase padre. Esto se conoce como herencia simple.
- **Ejemplo:**

```

public class Profesor {
 // Atributos y métodos del Profesor
}

```

```

 }

 public class ProfesorEmerito extends Profesor {
 // Atributos y métodos adicionales del ProfesorEmerito
 }
}

```

- **Limitación:** No se puede heredar de múltiples clases debido a la ambigüedad y complejidad que eso generaría.

- **Interfaces y Herencia Múltiple:**

- Las interfaces permiten una forma de herencia múltiple en Java. Una clase puede implementar múltiples interfaces.
- **Ejemplo:**

```

public interface IComportamiento {
 void actuar();
}

public interface ISalud {
 void revisarSalud();
}

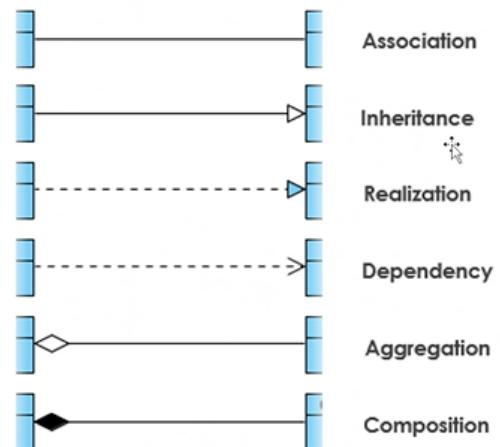
public class Persona implements IComportamiento, ISalud {
 @Override
 public void actuar() {
 System.out.println("La persona está actuando.");
 }

 @Override
 public void revisarSalud() {
 System.out.println("La persona está revisando su salud.");
 }
}

```

- **Imagen:**

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



## Convenciones y Ejemplos

- **Convención de Nombres:**

- Las interfaces deben comenzar con la letra "I" para indicar claramente que son interfaces.
- **Ejemplo:**

```

public interface IVehiculo {
 void conducir();
}

public class Coche implements IVehiculo {
 @Override
 public void conducir() {
 System.out.println("El coche está conduciendo.");
 }
}

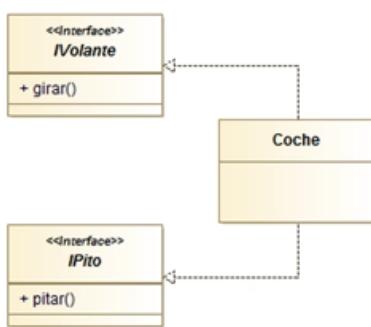
```

- **Uso de Números y Asteriscos:**

- Los números o asteriscos en los diagramas o líneas de código suelen indicar la cantidad de variables o métodos que se deben declarar o implementar.
- **Ejemplo:**
  - En una interfaz con un número específico de métodos, cada método representado puede tener un asterisco o número para indicar su cantidad.

- **Imagen:**

- **Implementación de Interfaces:**



```

1 public interface IVolante {
2 public void girar();
3 }
4 public interface IPito {
5 public void pitar();
6 }
7 class Coche implements IVolante, IPito {
8 public void girar() {
9 System.out.println("¡Girando, girando!");
10 }
11 public void pitar() {
12 System.out.println("¡Pitando, pitando!");
13 }
14 }

```

## Material de Lectura

- **Recomendación:** Asegúrate de leer y comprender el material de referencia proporcionado para una comprensión más profunda y para practicar la implementación de interfaces y la herencia en Java.

Este resumen ofrece una visión clara de cómo trabajar con interfaces y sus restricciones en Java, incluyendo convenciones y prácticas recomendadas para su implementación. Las imágenes proporcionan ejemplos

visuales que ayudan a entender mejor los conceptos.

# Apuntes semana 9

---

## Clase 21

Claro, aquí tienes el resumen ampliado con las correcciones y detalles adicionales:

### Corrección del Examen

#### Corrección de la Lista de Animales

- **Lista Descendente:**
  - **Descripción:** La lista debía ser en relación con el animal que te tocaba, creando una lista descendente hasta llegar al animal seleccionado, no abarcando toda la cadena de animales.
  - **Corrección:** En lugar de incluir toda la cadena de animales, enfócate en construir la lista que comienza desde el animal seleccionado y desciende hacia los animales anteriores en la jerarquía.

#### Creación de Animales

- **Clasificación de Animales:**
  - **Descripción:** Solo se debía crear un animal que perteneciera a la clasificación que le correspondía.
  - **Corrección:** Asegúrate de crear únicamente el animal de la clasificación especificada en la pregunta, no más.

#### Polimorfismo

- **Acción de Varias Maneras:**
  - **Descripción:** El polimorfismo permite realizar una acción de varias maneras. En tu caso, se esperaba que se mostrara el nombre del animal y lo que le correspondía (por ejemplo, "comer").
  - **Corrección:** Implementa el polimorfismo correctamente mostrando cómo la acción puede variar según el tipo de animal. Debías incluir la acción correspondiente, como "comer", junto con el nombre del animal.

#### Contraseña y Otros Aspectos

- **Contraseña:**
  - **Descripción:** Asegúrate de incluir los detalles relacionados con la contraseña según las especificaciones dadas en el examen.

#### Diagrama de Clases

- **Paquetes y Clases:**
  - **Descripción:** En el diagrama de clases, debes ubicar un animal en el Paquete 2 y otro en el Paquete 3.

- **Corrección:** Coloca las clases de animales en los paquetes correspondientes como se indicó en el examen.
- **Estructura de Paquetes:**
  - **Descripción:** En total, se tenían tres paquetes: los dos primeros relacionados con la primera pregunta y el tercer paquete con el polimorfismo.
  - **Corrección:** Organiza los paquetes de manera que los dos primeros correspondan a la primera pregunta y el tercero a la segunda pregunta, que trata sobre el polimorfismo.
- **Elementos Fuera de Paquetes:**
  - **Descripción:** El biológico y el sistema deben estar fuera de los otros paquetes.
  - **Corrección:** Coloca el biológico y el sistema en un nivel superior, fuera de los paquetes específicos.

## Puntos de Corrección

- **Evaluación:**
  - **Descripción:** Se asignarán 2 puntos por una corrección completa y bien hecha.
  - **Corrección:** Asegúrate de seguir las indicaciones para obtener la puntuación completa, corrigiendo todos los aspectos mencionados en la revisión.

Este resumen proporciona una guía clara sobre cómo realizar las correcciones y ajustes necesarios en el examen, asegurando que cada aspecto esté cubierto de acuerdo con las instrucciones dadas.

## Clase 22

No se puede instanciar si es que es una clase abstracta.

### Asociacion

#### Asociación

Las asociaciones son relaciones entre clases en un diagrama de clases UML. Están representados por una línea sólida entre las clases. Las asociaciones generalmente se nombran usando un verbo o una frase verbal que refleja el dominio del problema del mundo real.

#### Asociación simple

- Un vínculo estructural entre dos clases de pares.
- Hay una asociación entre Class1 y Class2



Hay una asociación que conecta la clase <> Class1 y la clase <> Class2.

La relación se muestra como una línea sólida que conecta las dos clases.

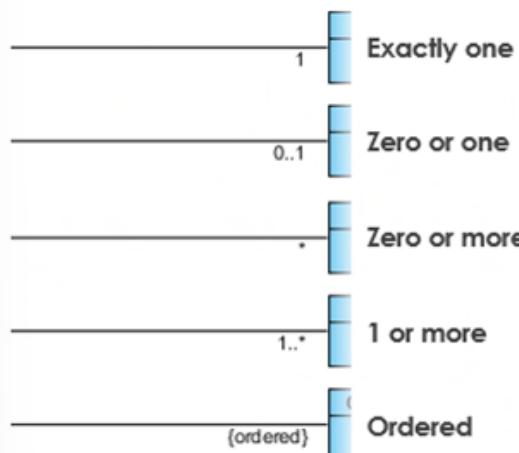


Flechas indican el sentido de lectura. Los numeros indican la cantidad que se le puede asignar, o que puede existir (Cardinalidad {me recuerda al sistema cardinal de SAO, este detectaba y eliminaba errores del sistema}).

si no se pone el numero por defecto se indica que es uno. "1" relaciones uno a uno "+" minimo existe uno o muchas personas. "\*" pueden existir cero o muchos elementos.

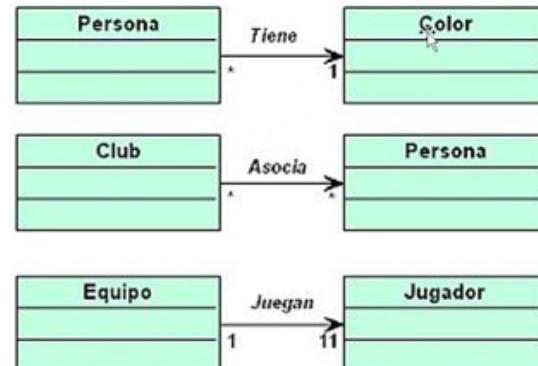
### Cardinalidad

- Uno a uno
- Uno a muchos
- muchos a muchos



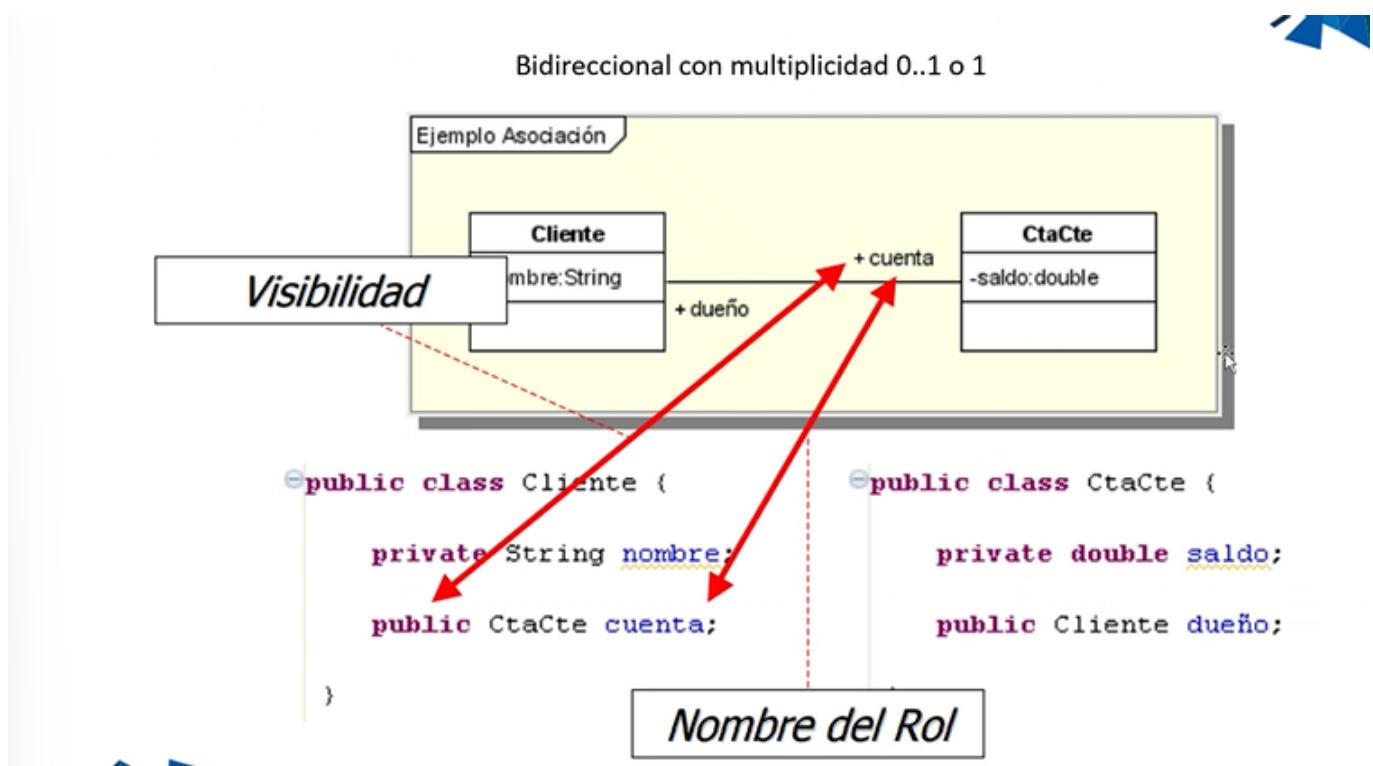
- 1 --> sólo uno
- 0..1 --> cero o uno
- n --> Indica cuántas relaciones pueden haber.
- n..m --> varios a varios
- \* --> cero o más
- 0..\* --> cero o más (lo mismo que el anterior)
- 1..\* --> uno o más

Ejemplos:



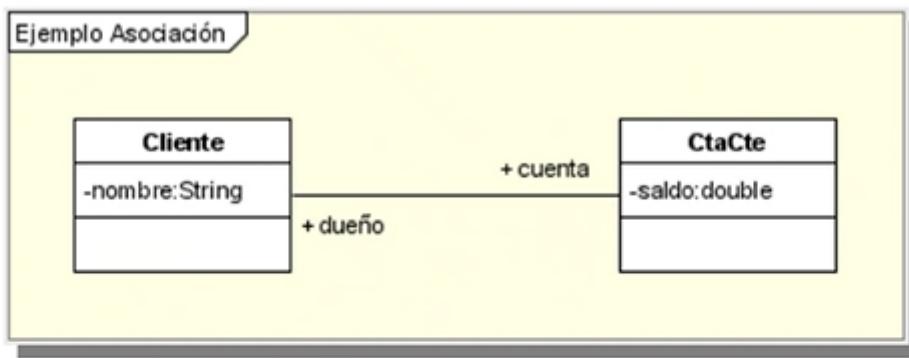
### Ejemplo

Interpretacion de diagrama de clase para escribirlo o pasarlo a codigo



Resultado del ejemplo:

### Bidireccional con multiplicidad 0..1 o 1



```

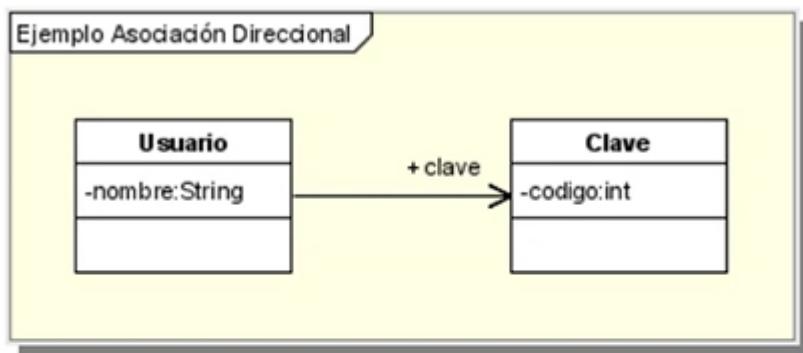
public class Cliente {
 private String nombre;
 public CtaCte cuenta;
}

public class CtaCte {
 private double saldo;
 public Cliente dueño;
}

```

Otro ejemplo pero ahora esta unido por la flecha los dos cuadros del diagrama, señalando la asociacion.

### Direccional con multiplicidad 0..1 o 1



```

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

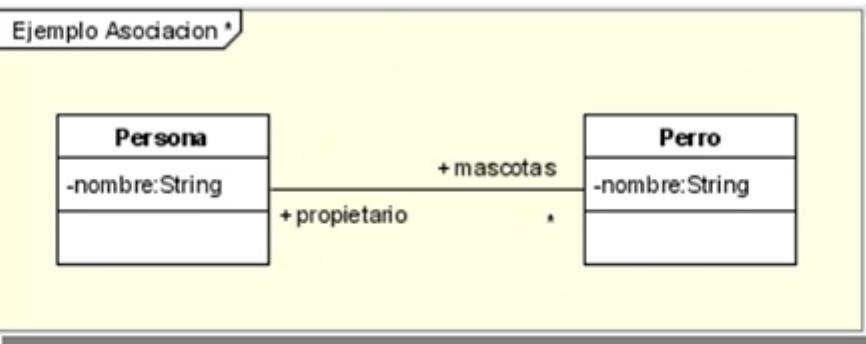
public class Clave {
 private int codigo;
}

```

#### Detalles

Si se tiene un asterisco o cruz: en este ejemplo, el asterisco indicaria que se tendría una colección de esos objetos, tal como se dice, ejemplo general y otro mas exacto.

### Bidireccional con multiplicidad \*



```

public class Persona {

 private String nombre;
 Java.util.List<Perro> mascotas = new ArrayList<Perro>();
 public java.util.Collection mascotas = new java.util.TreeSet();

}

public class Perro {

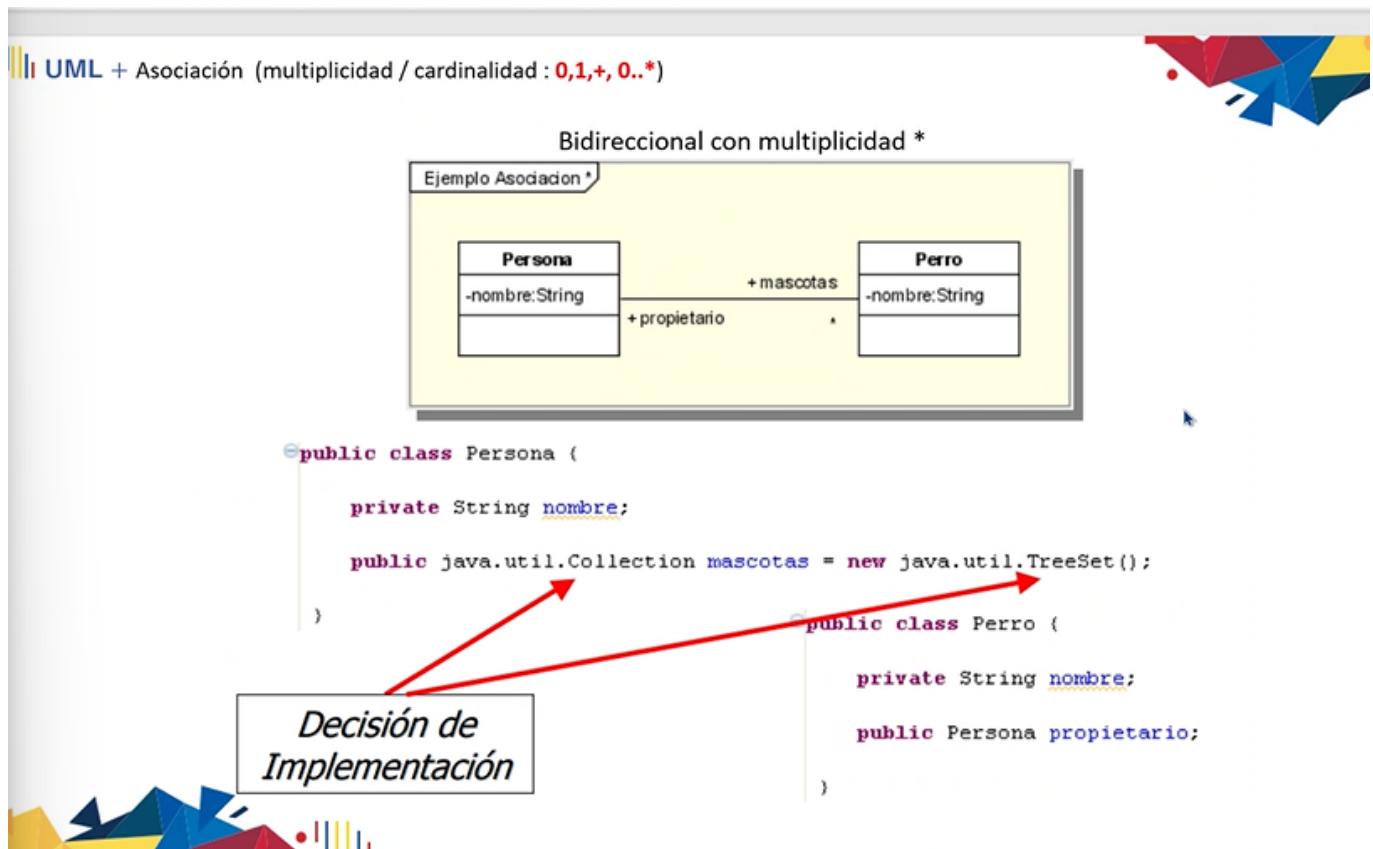
 private String nombre;

 public Persona propietario;

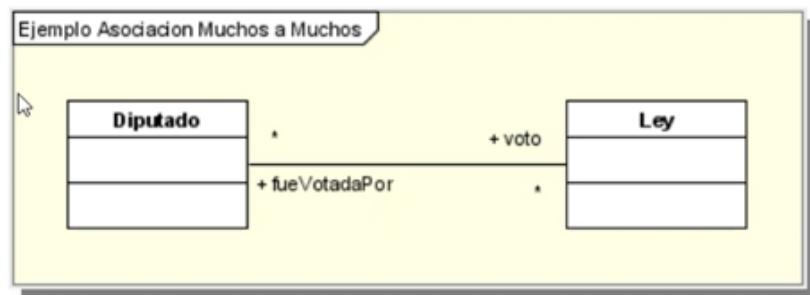
}

```

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



### Bidireccional con multiplicidad \*



```

public class Diputado {

 public java.util.Collection voto = new java.util.TreeSet();

}

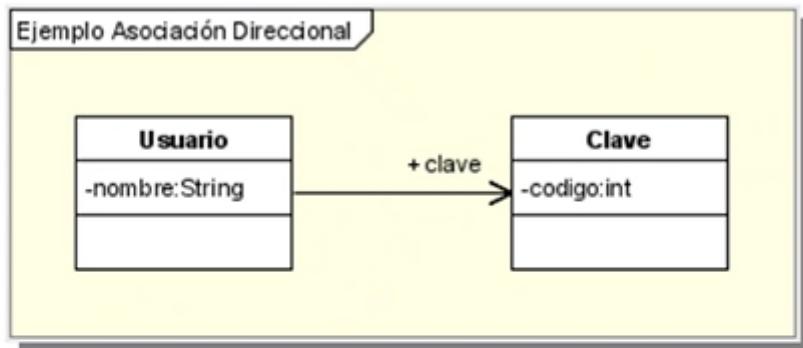
@public class Ley {

 public java.util.Collection fueVotadaPor = new java.util.TreeSet();
}

```

si tiene un 1, +, \* es una lista, si te determina por algun numero, debes hacer un arreglo de la cantidad que se te solicita.

### Direccional con multiplicidad 0..1 o 1



```

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

@public class Clave {
 private int codigo;
}

```

A continuacion un trabajo en clase:

## MISION

Ganancia: 0,5p

Mision: crear animales vertebrados.

Clasificacion de los animales vertebrados:

## Misión → CLASIFICACIÓN DE LOS ANIMALES VERTEBRADOS



Lo que se desea optener:

## Misión → Validar diagrama de clases



Realizar el Diagrama en una hoja

## Clase 23

Continuacion de la mision:

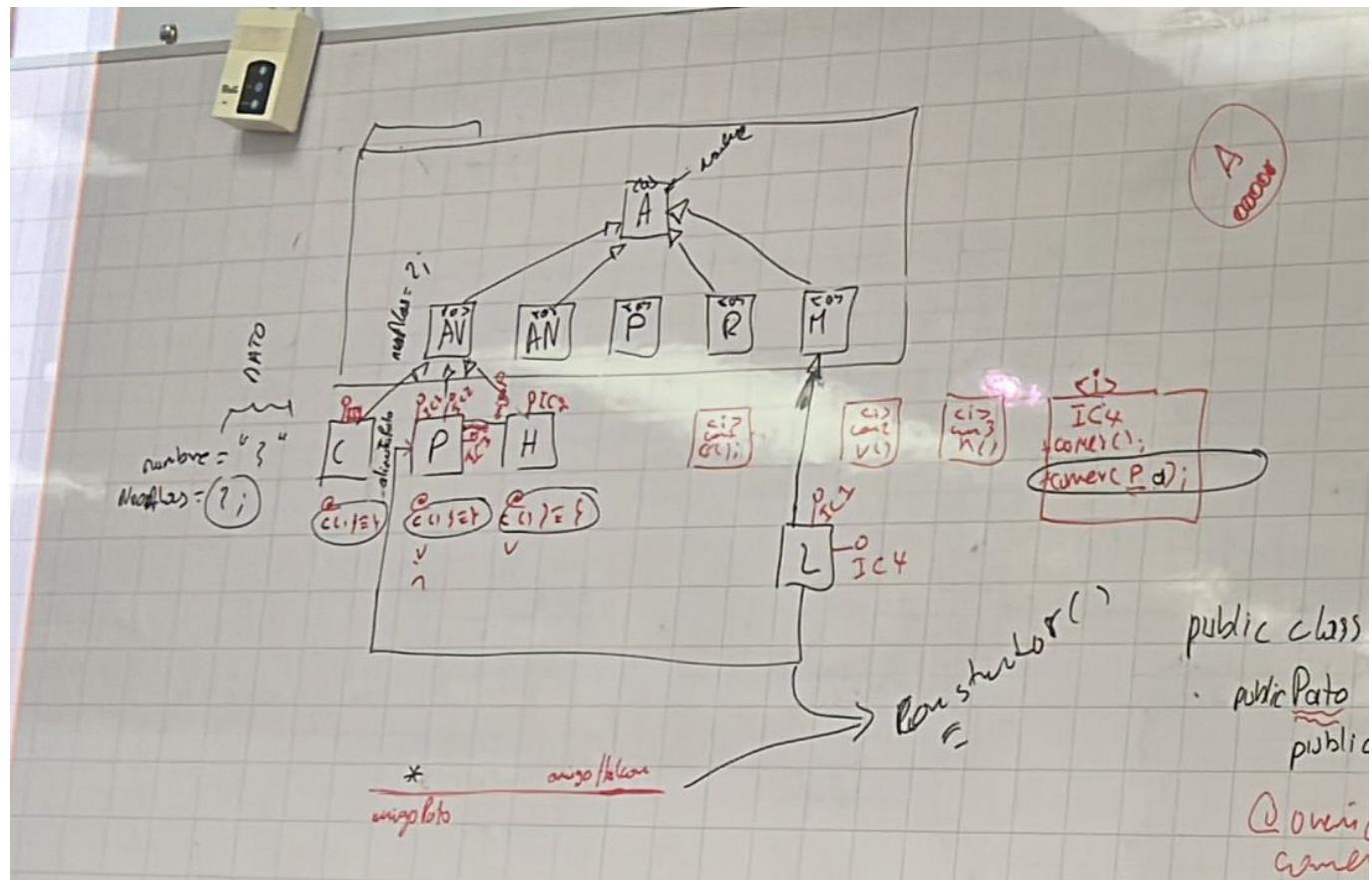
Planificación a mano, para comprender le tema, ordenar las ideas para entender que es lo que se pide y que es lo que se requiere.

Diseño del digrama de clase con herencia se debia de colocar todas las clasificaciones siendo "hijos" del 'reino animal', estas se mantienen el clases abstractas.

Tras eso, se colocan los animales que se necesiten a continuacion, y para las acciones de los animales se pueden utilizar interfaces, para lograr obtener un mundo de acciones para mas animales.

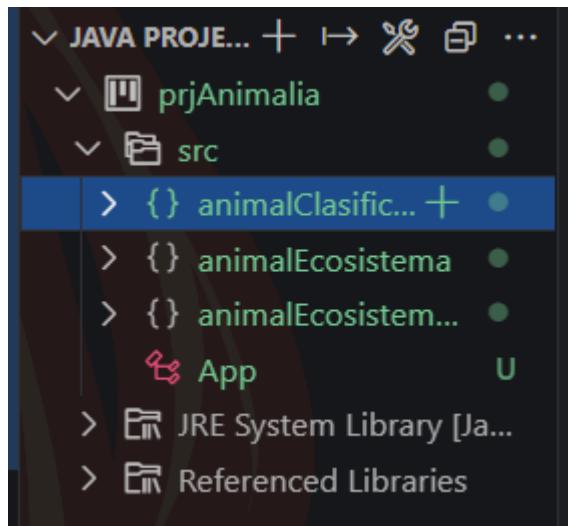
Con las interfaces se pueden ayudar a las demas especies, si es que se crean mas especies, con las interfaces se puede hacer que estas tambien tengan las acciones, con la ayuda de un `@override`, se puede sobreescibir, para adaptarle al nuevo animal.

Un metodo tiene que ser publico, para obligar al objeto a hacer algo.

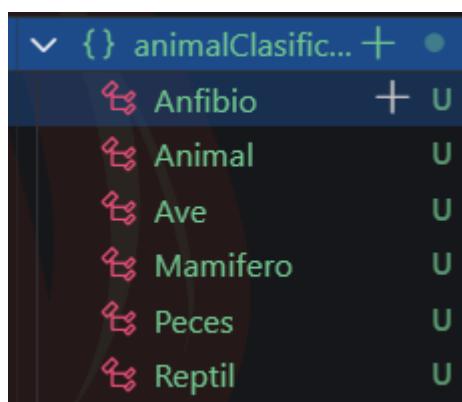


Dependiendo, las interfaces se pueden poner en una misma carpeta.

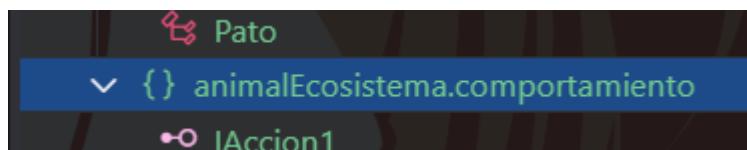
al momento de programar, crear el proyecto y comenzar creando las carpetas.



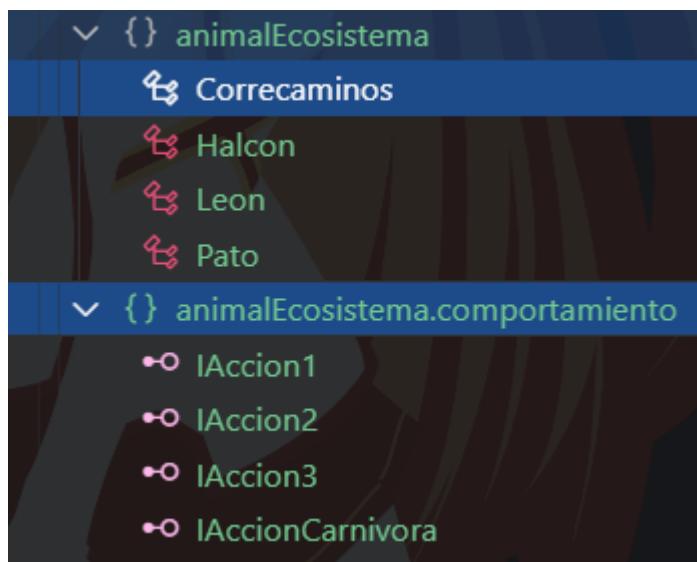
y dentro de cada carpeta poner las clases ya sean abstractas o no.



ademas se puede meter un paquete dentro de otro paquete, y mas.



tendra al principio el nombre del paquete en que estara metido, se parado por un punto para el nombre del nuevo paquete.



ademas de colocar las interfaces, guiararse por la simbología.

Llenar todo de acorde al digrama de clase.

# Apuntes Semana 10

## Clase 24

Si esta mal nombrado, no se puede materializar.

Observar donde se encuentra la lista.

Relacion consigo mismo.

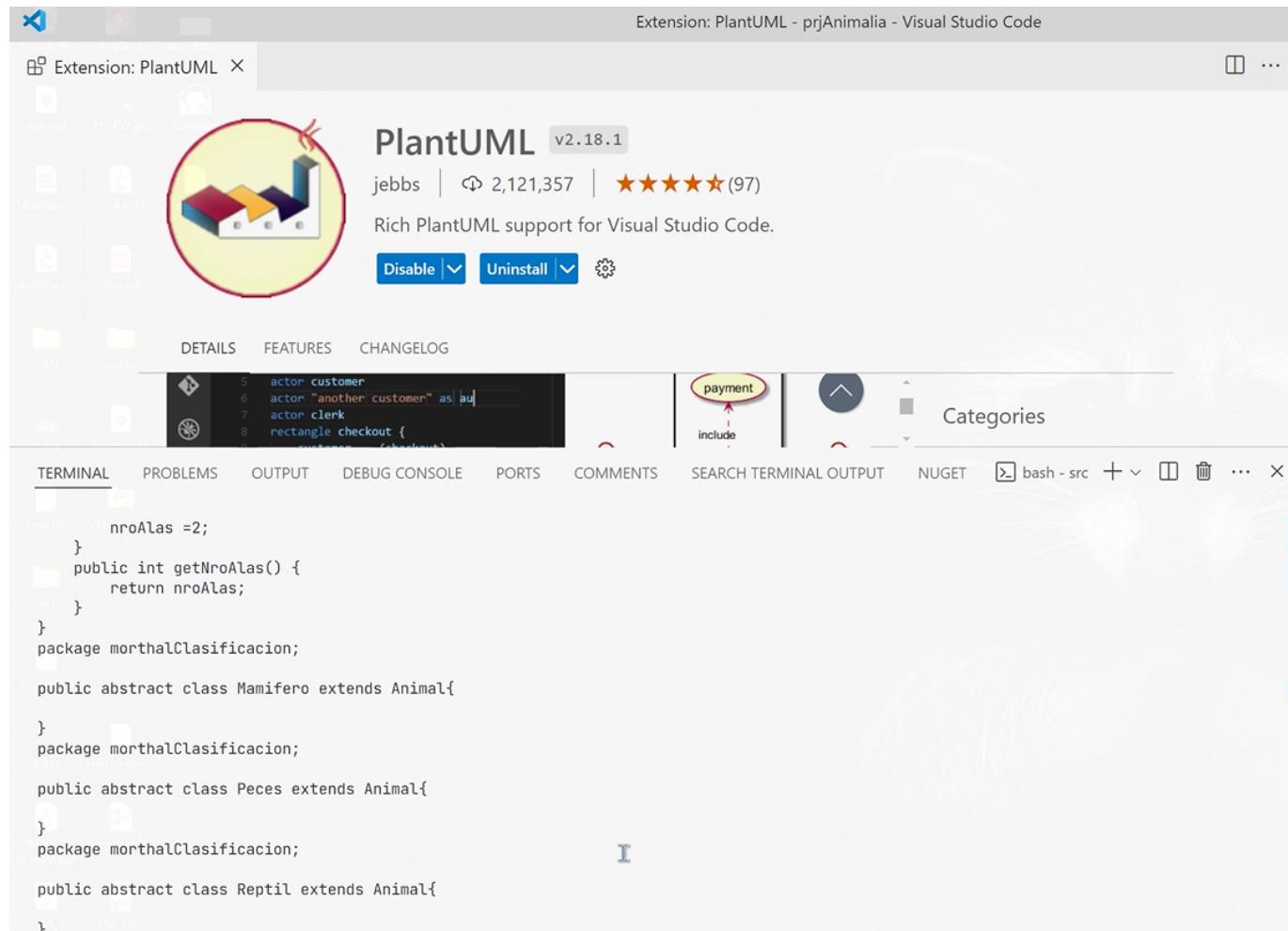
Relacion en el ejercicio: *el leon se pude comer un pato* en este caso, hay mas variedad. *un pato alcanza solo para un leon*

Saber como se utiliza la flecha.

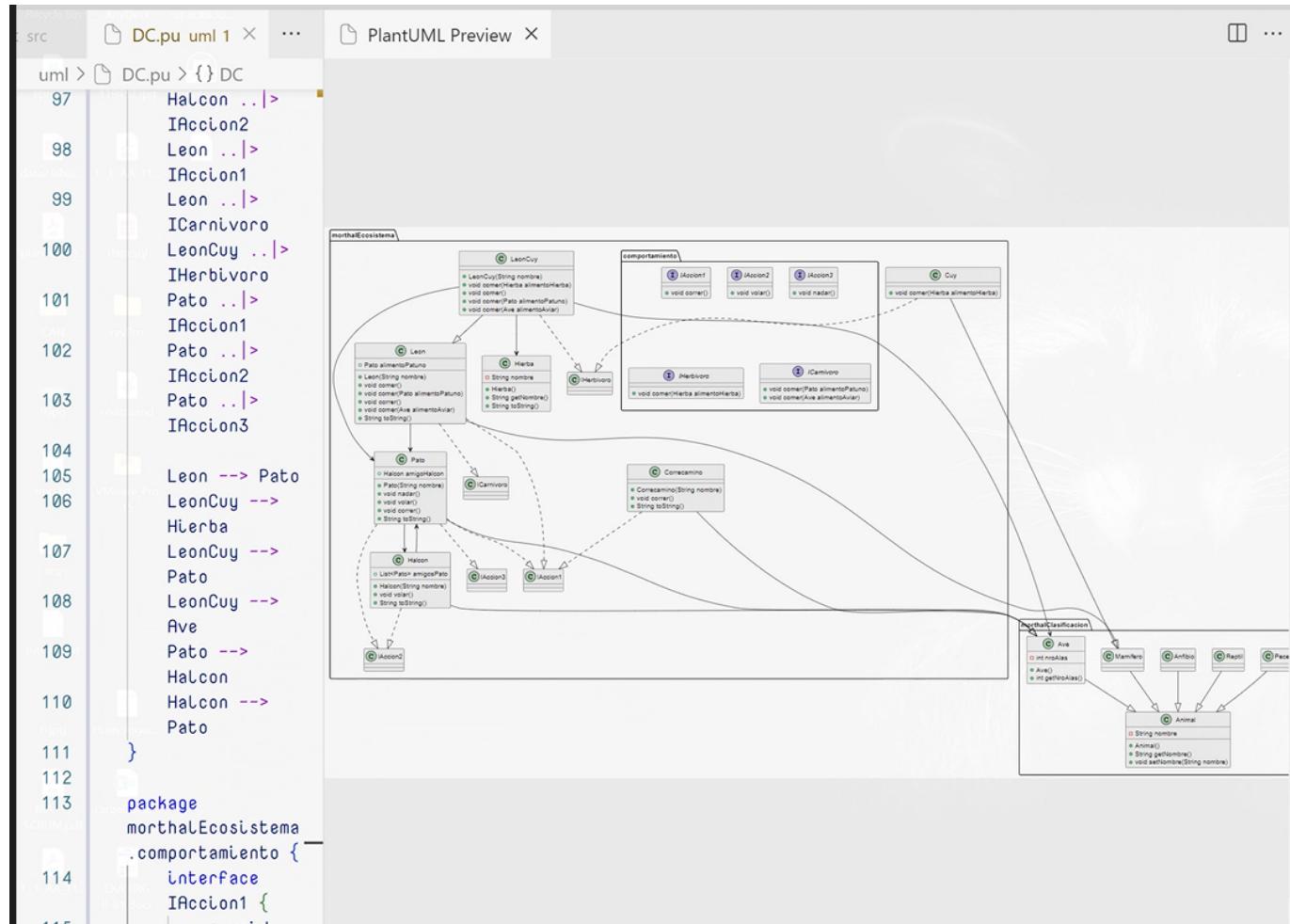
Ayuda de listas, las colecciones si se pueden coleccionar de la misma especie o si es distinta, ayudandose con las herencias.

Ayuda para hacer los diagramas de clase.

1. Extencion.



ALT + d: procesar



## Clase 25

### Base de datos

Repositories para almacenar datos:

```
-- database: ./database/EXOBOT.sqlite
/*
CopyRight
autor: I
Fecha: */
DROP TABLE IF EXISTS ExaBot;
DROP TABLE IF EXISTS IABot;
CREATE TABLE IABot (
 IdIABot Integer primary key autoIncrement,
 Nombre TEXT NOT NULL UNIQUE,
 Observacion TEXT,
 FechaCrea datetime default current_timestamp
);
```

## Clase 26

(laboratorios ocupados)

# Apuntes Semana 11

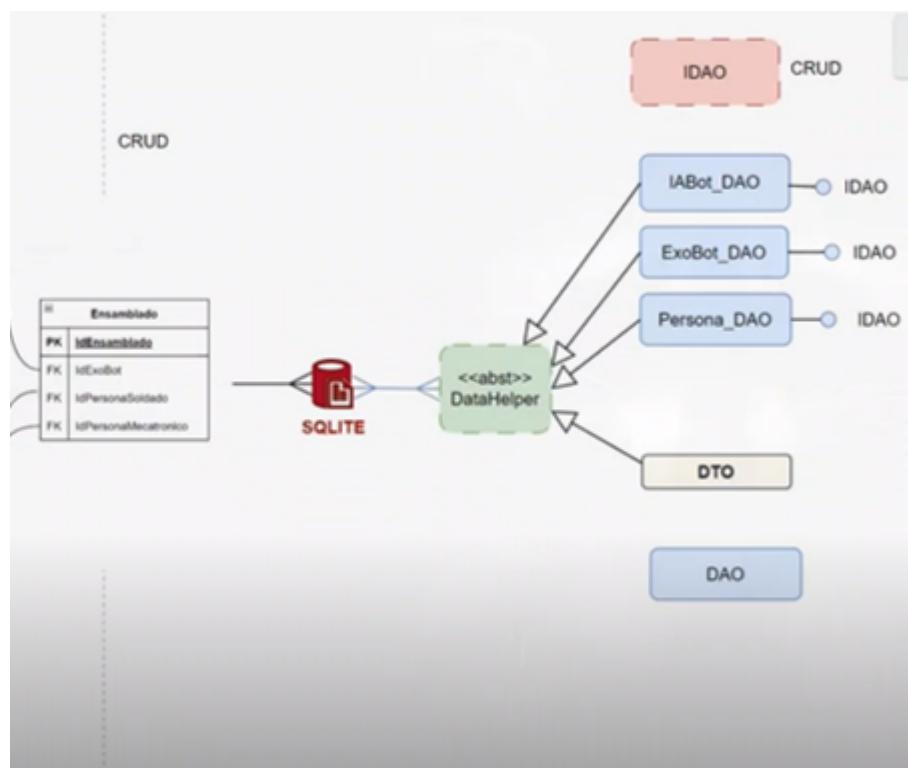
---

## Clase 27

### Arquitectura N-Layer

Para conectar la base de datos, se necesita un DataHelper, que los archivos esten en orden, pues no se puede almacenar todo al mismo tiempo, y no deben ser demasiados, ademas de hacer una DAO cada uno debe ejecutar un CRUD por cada base que se quiera conectar, si se hace todo en general va a ser mas complicado el mantenimiento.

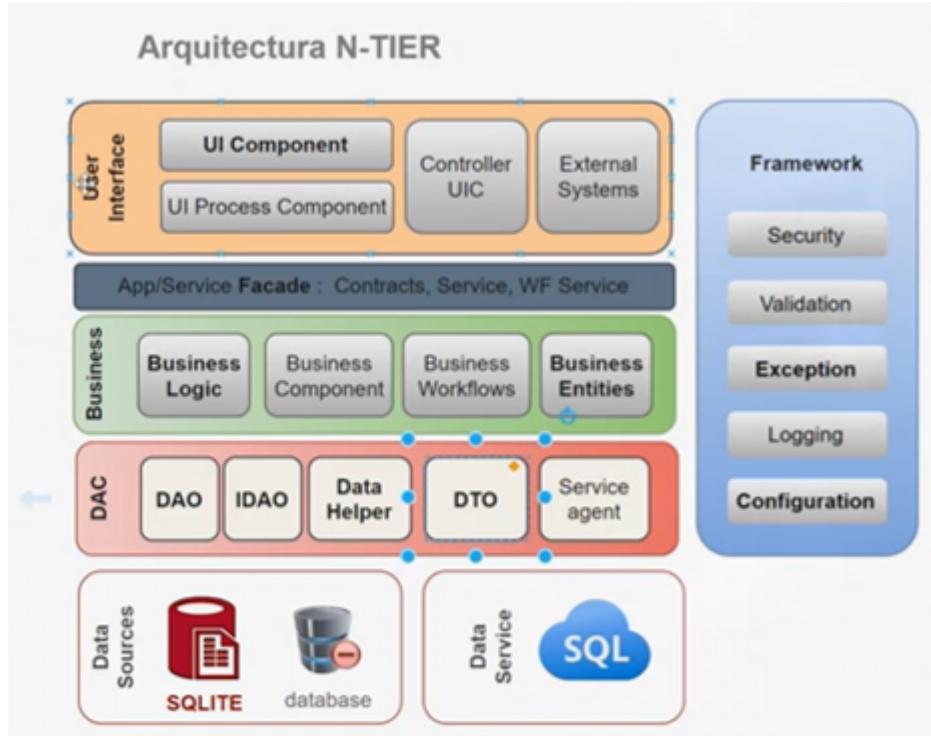
A su vez crear un IDAO, que es una interface de los DAO, misma que debera ejecutar el CRUD para todos los DAO y asi que estos no se desvien o hagan cosas que no son, ademas de crear la carpeta necesaria para los DTO, que son para la transferencia de datos.



Armas toda la estructura de los guardados, eliminados y editar esos datos.

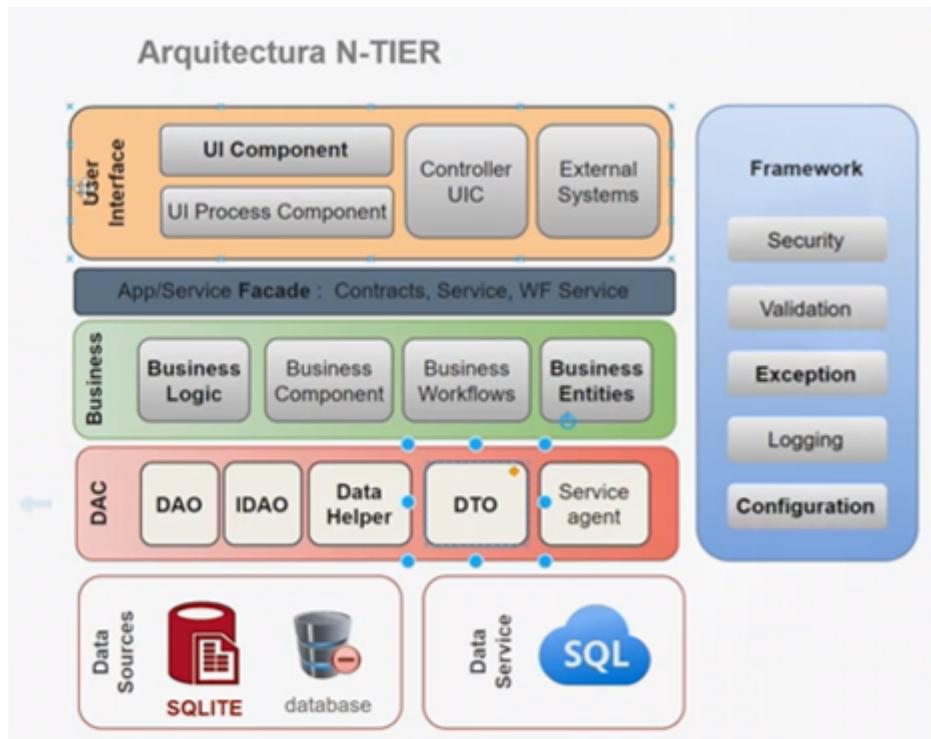
y todos estos archivos se encuentran en la parte de la DAC.

*Nota:* la unica forma de acceder a la base es el DataHelper.

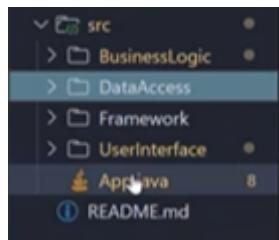


El cuadrado de DAC, el cuadro rojo para ser precisos.

Continuando con las capaz esas capaz son de las que se compone el proyecto para armar la aplicación por así decirlo.



relacionado con esto se deben tener estas carpetas:



primero se trabaja la data access, despues business logic, y por ultimo la user interface, se vera un poco de la framework.

Empezando por la data access:

## Apuntes Semana 12

### Clase 30

**Misión**

Se requiere crear un exoesqueleto cyberbot (**exobot**) que pueda ser usado por los soldados rusos para potencializar las habilidades físicas de los guerreros en la batalla contra Ucrania.

El exobot se deriva de un sistema matriz con inteligencia artificial llamado IABOT que le permite potencializar las acciones que realiza el soldado. El exobot se compone de las mismas extremidades del humano. El brazo derecho puede cargar una metralleta y el izquierdo un láser cuya potencia varía según la energía de la fuente de poder.

El exobot tiene un turbo reactor en la parte de la espalda que le permiten volar usando energía de la fuente de poder.

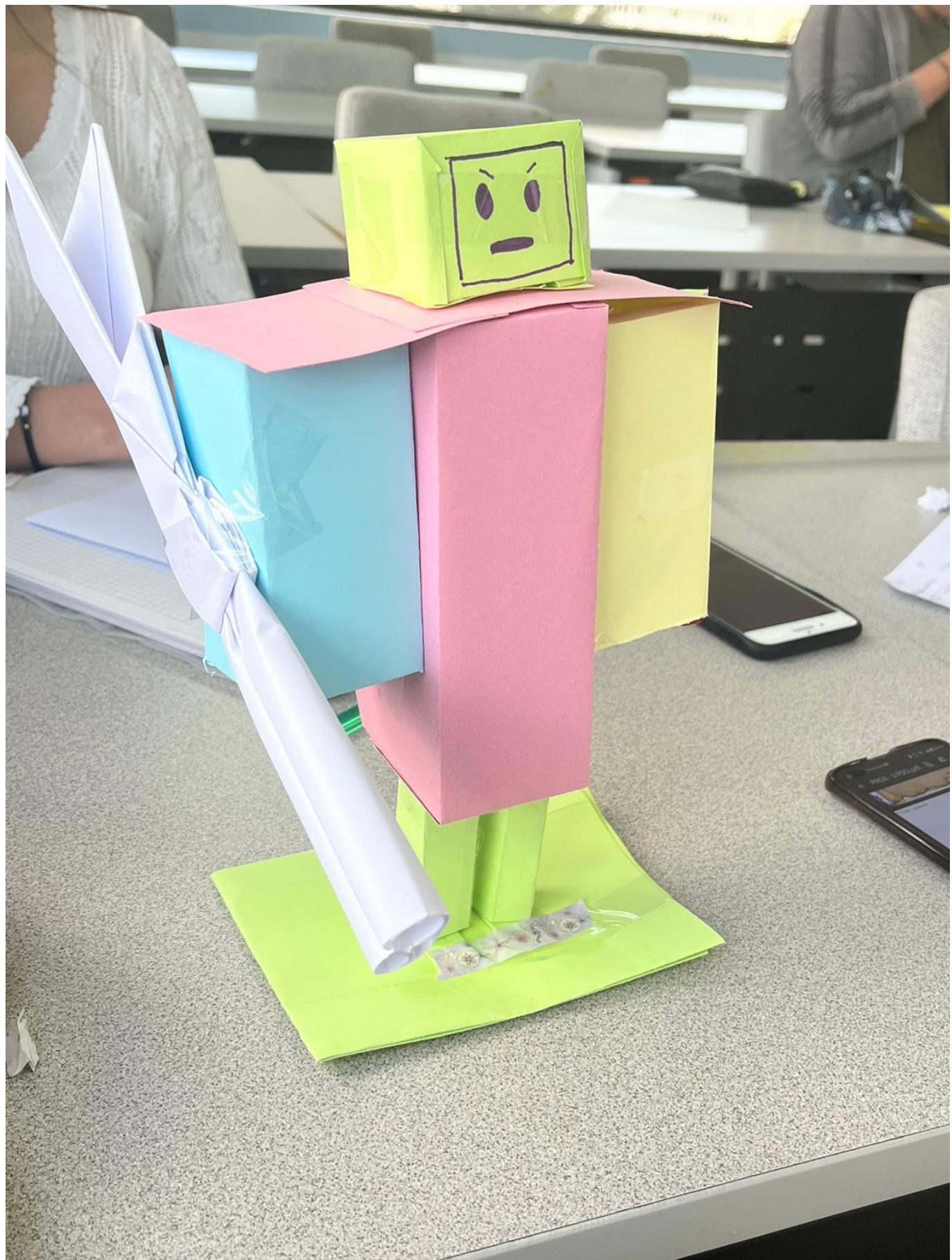
La fuente de poder es recargable y reemplazable bajo la asistencia del IABOT.

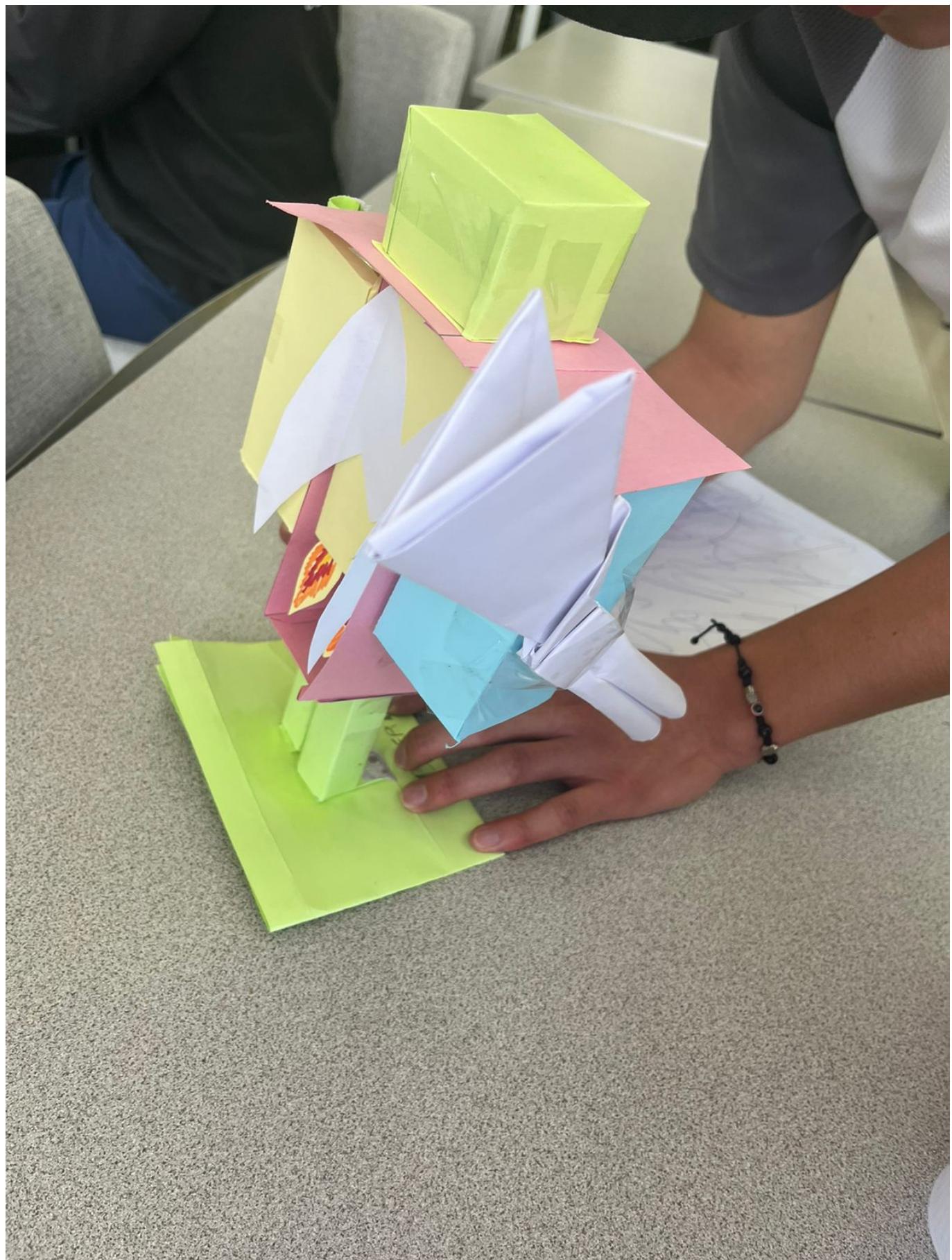
El exobot tiene capacidad para aprender inglés y español y requiere de expertos parlantes de inglés y español quienes son los únicos que deben entrenar el léxico, gramática y fonética.

Patrício Michael Pachá Angamarca  
MASTER EN INGENIERÍA DE SOFTWARE

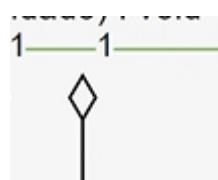
hacer el digrama de clase con todo lo aprendido y un prototipo del exobot.

**Prototipo:**

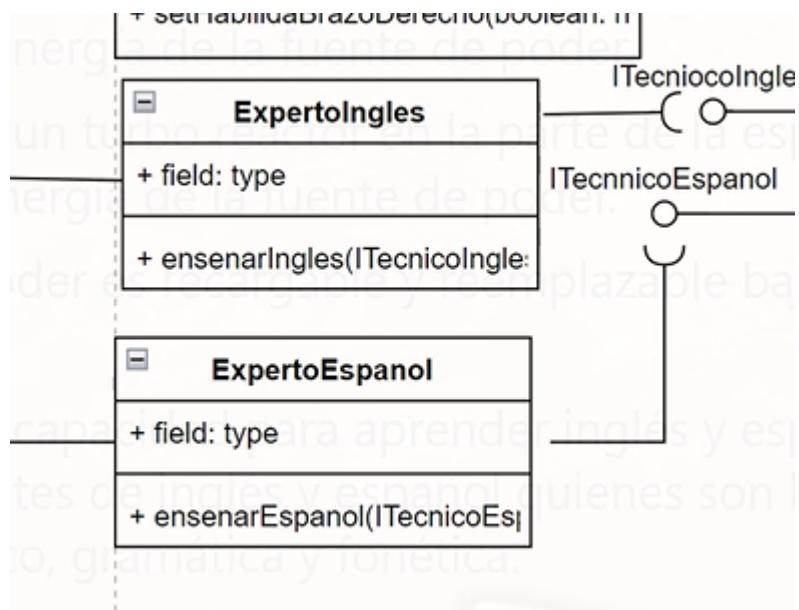




## Clase 31

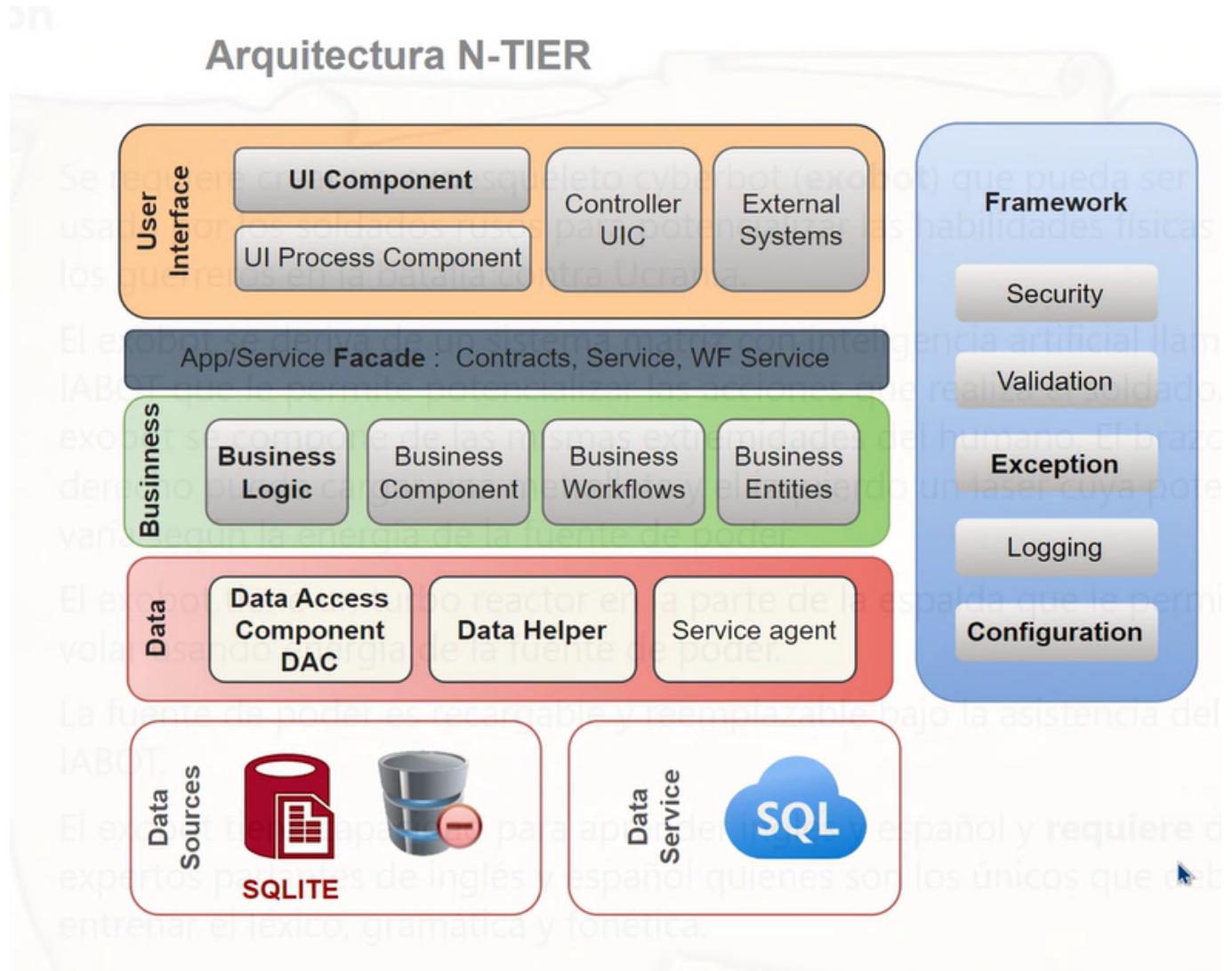


simbolo de que es extraible. (el que esta en blanco)



si se los crea como interfaces, no se los puede instanciar.

por que le exobot tiene la capacidad de aprender, alguien tendra que enseñarle.



trabajar con las arquitecturas empresariales, y aumentar la base de datos con una estructura.

Se necesita de un driver, para conectar.

Se debe instalar el ORACLE para poder acceder a las bases de datos.

se necesita crear una tabla que es como una tabla de excel, para la base de datos, donde se almacenen en orden y como deben ser los datos.

todas las tablas deben tener un primary key.

Primary key: no se pueden cambiar por nada, si se pone mal te metes en un lio.

no poner como primera linea el primary key, sino crear un campo aparte.

todo debe funcionar en función de la atomicidad, osea que los datos deben ser atomicos, osea que son únicos.

## Apuntes Semana 13

### Clase 33

relación 1 a 1

relacion 1 a muchos

relacion de muchos a muchos

cardinalidad: numero de veces

En cada tabla poner un primary key

ejemplo

Cursos en linea "Undemy"

roadmad: un tutor elabora varios cursos (dibujo) un curso puede ser elaborado por un solo tutor un alumno puede tomar muchos cursos un curso puede ser tomado por varios alumnos

- como pagar el curso
- idiomas

### Tutor

---

idTutor int(PK)

---

nombre char(100)

---

apellido char(100)

---

edad int X

---

fechaNacimiento date

---

estatura float

---

---

---

idUsuario int

---

fechaRegistro date

---

fechaModificacion date

- X: edad int (no guardar edad, sino fecha de nacimiento, hacer esto con todas las cosas que se calculen)
- fechaNacimiento date (debe ser de esta forma con el origen de la data)
- fechaRegistro date siempre poner
- fechaModificacion date siempre poner

```
CREATE TABLE GCCatalogo (
 ,IdCatalogo INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT
 ,IdCatalogoTipo INTEGER NOT NULL REFERENCES GCCatalogoTipo (IdCatalogoTipo)
 ,Nombre VARCHAR(10) NOT NULL UNIQUE
 ,Descripcion VARCHAR(90)

 ,Estado VARCHAR(1) NOT NULL DEFAULT('A')
 ,FechaCreacion DATETIME DEFAULT(datetime('now','localtime'))
 ,FechaModifica DATETIME
);

CREATE TABLE GCUbicacion (
 ,IdUbicacion INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT
 ,Pais VARCHAR(10) NOT NULL DEFAULT ('Ecuador')
 ,Region VARCHAR(10)
 ,Provincia VARCHAR(20)
 ,Estado VARCHAR(1) NOT NULL DEFAULT('A')
 ,FechaCreacion DATETIME DEFAULT(datetime('now','localtime'))
 ,FechaModifica DATETIME
);


```

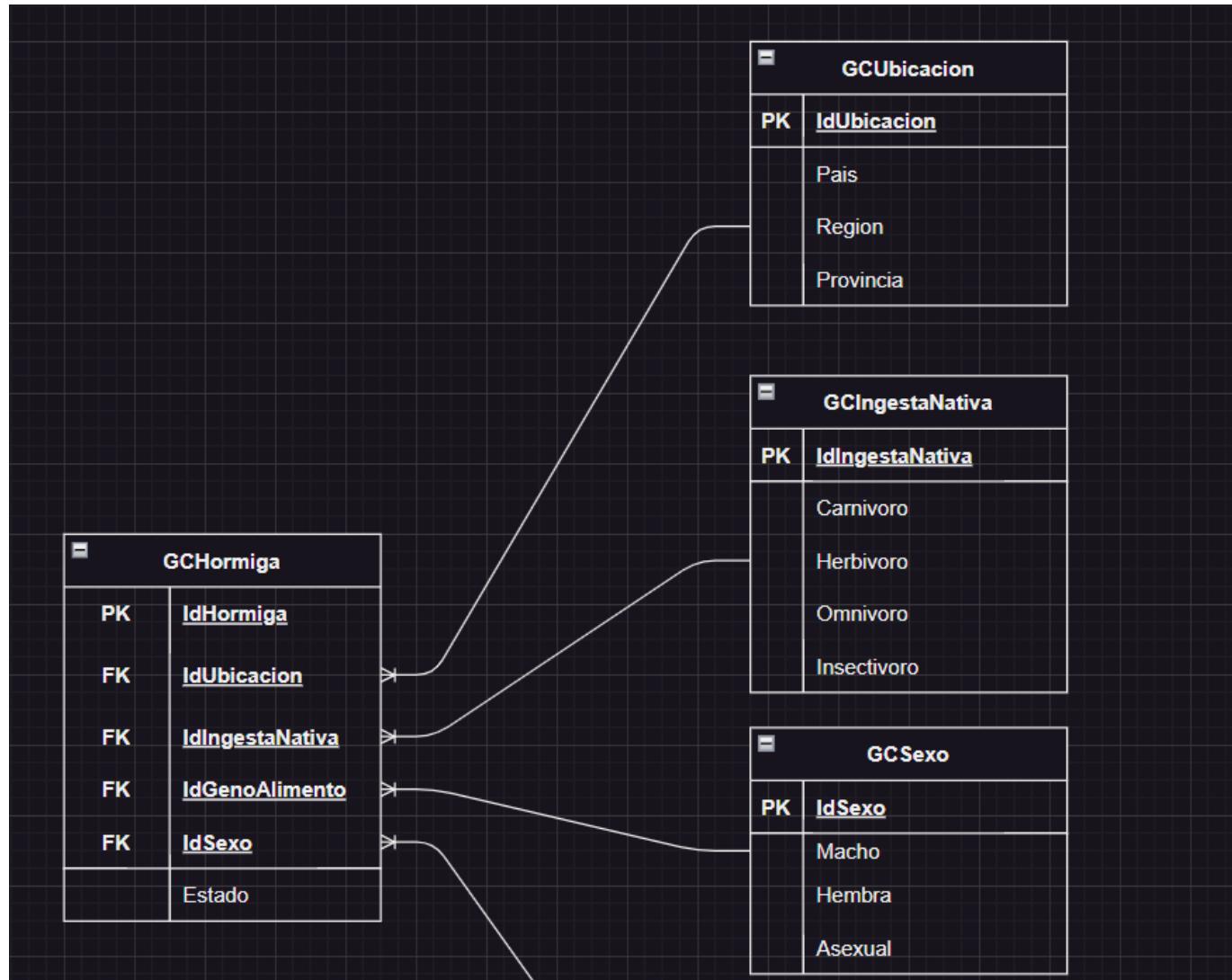
## alumno

---

idAlumno int(PK)

las tablas de tutor y alumno son iguales, por lo que se puede crear una sola tabla y que las tablas que contienen las mismas cosas se simplifiquen al estar enlazadas a esta tabla mas grande.

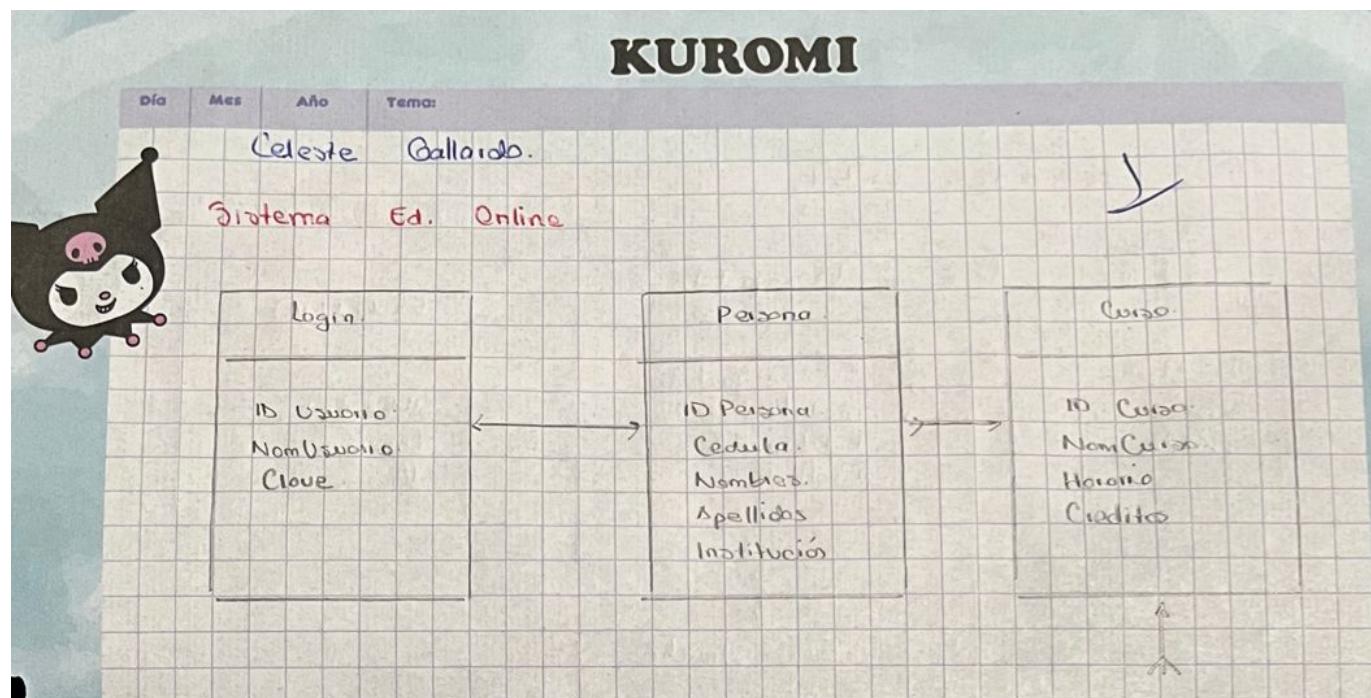
Para los cursos, estos se encuentran por categorias, ejemplo

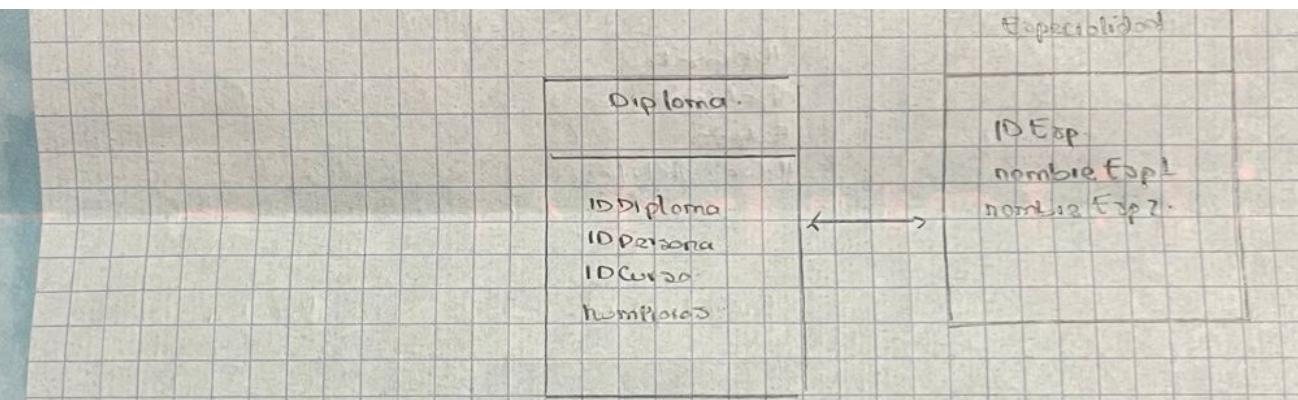


## Clase 34

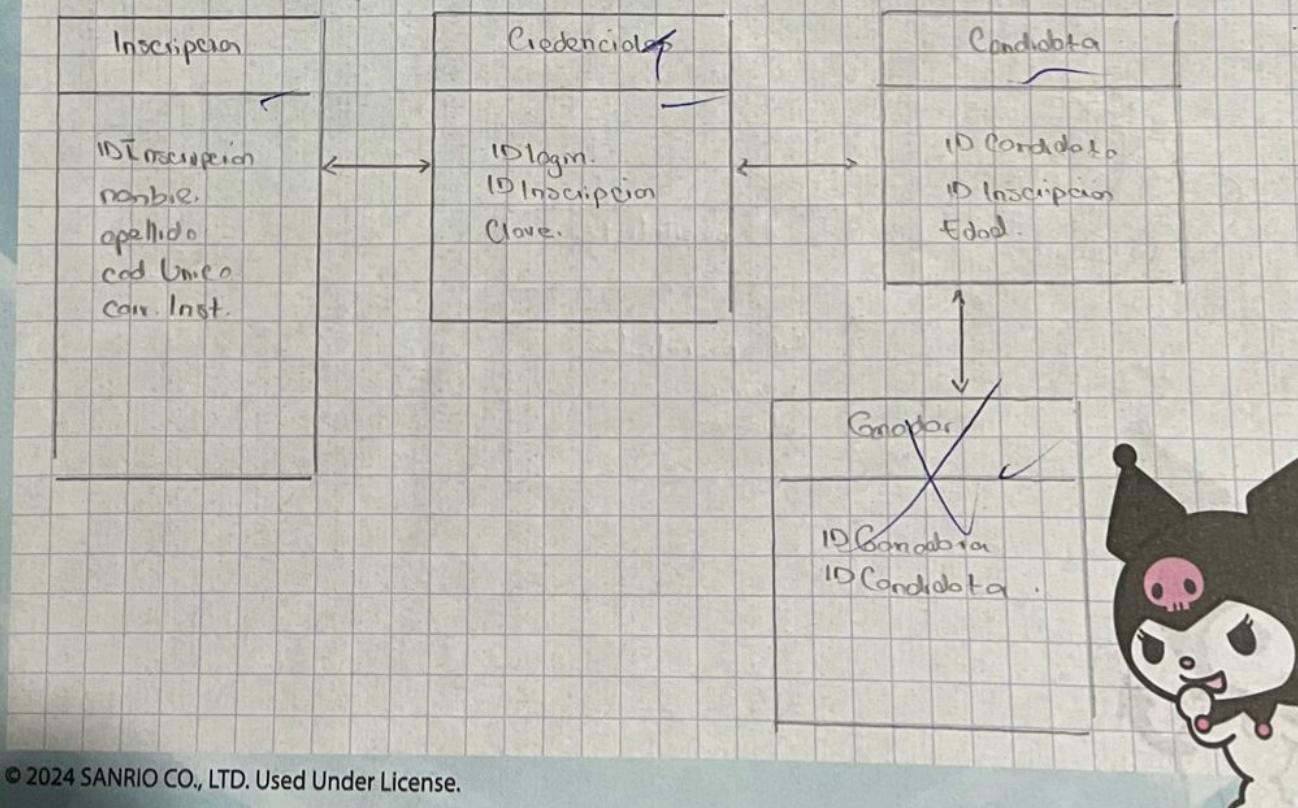
apuntes a mano

### Deber





Sist. Elección de Reino de FIS EPN.

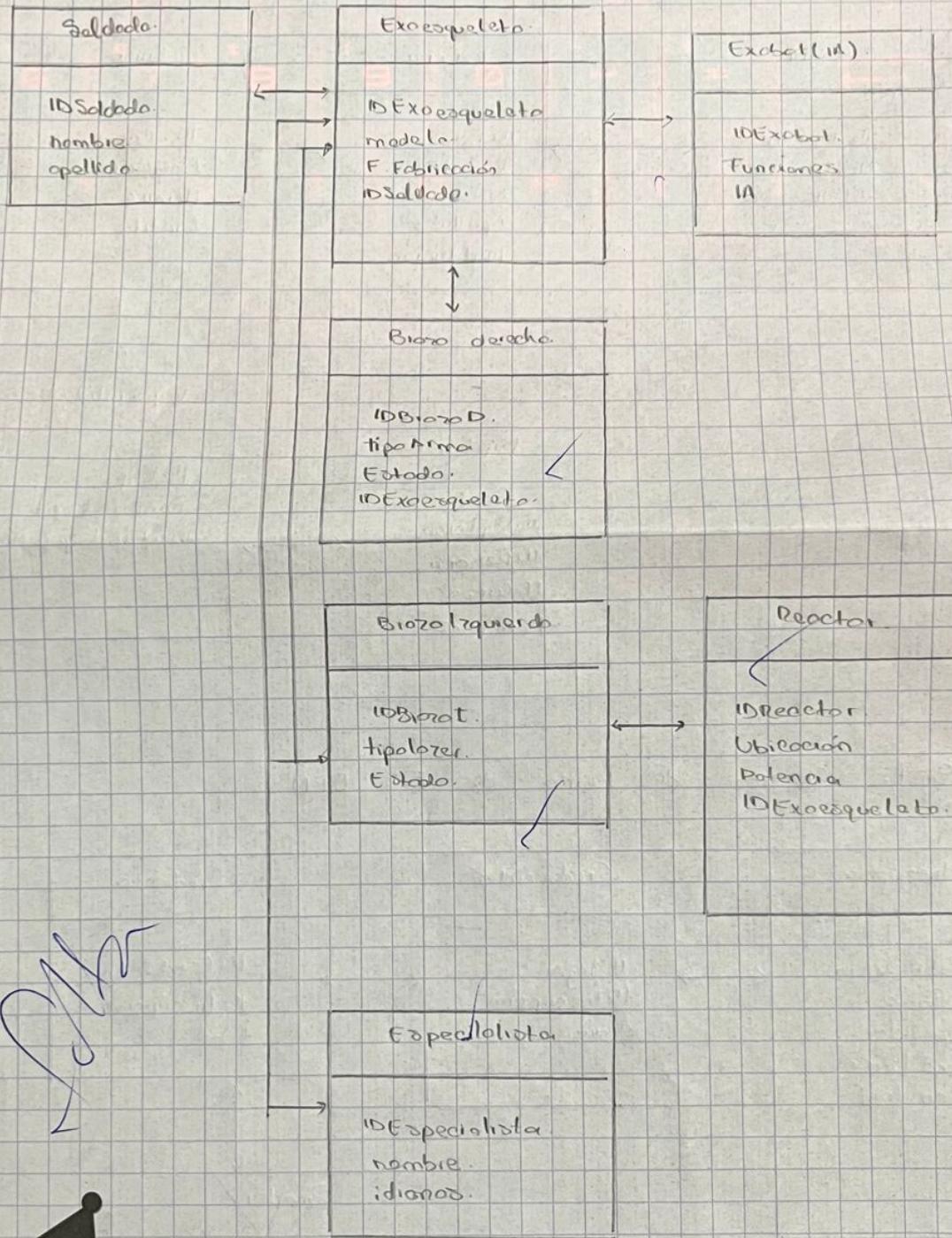


© 2024 SANRIO CO., LTD. Used Under License.

# KUROMI

Día Mes Año Tema:

Base de datos Exobot.



© 2024 SANRIO CO., LTD. Used Under License.

Como armar la base de datos den un archivo sql, mismo que creara la base de datos en un SQLite.

PRIMARY KEY (siempre se debe poner siempre en mayusculas)

Las celdas deben ser unicas: UNIQUE

```

▷ Run | New Tab | Comment Code
DROP TABLE IF EXISTS PersonaTipo;

▷ Run | New Tab | Copy | Comment Code
CREATE TABLE IABot (
 IdIABot INTEGER PRIMARY KEY autoincrement
 ,Nombre TEXT NOT NULL UNIQUE
 ,Observacion TEXT
 ,FechaCrea DATETIME DEFAULT current_timestamp
);
▷ Run | New Tab | Copy | Comment Code
CREATE TABLE ExaBot (
 IdExaBot INTEGER PRIMARY KEY autoincrement
 ,IdIABot INTEGER NOT NULL
 ,Nombre TEXT NOT NULL
 ,Serie TEXT NOT NULL
 ,CONSTRAINT fk_IABot FOREIGN KEY (IdIABot) REFERENCES IA_bot(IdIABot)
);

▷ Run | New Tab | Copy | Comment Code
CREATE TABLE PersonaTipo (
 IdPersonaTipo INTEGER primary key autoincrement
 ,Descripcion TEXT NOT NULL UNIQUE
 ,FechaCrea datetime default current_timestamp
);
▷ Run | New Tab | Copy | Comment Code
CREATE TABLE Persona (

```

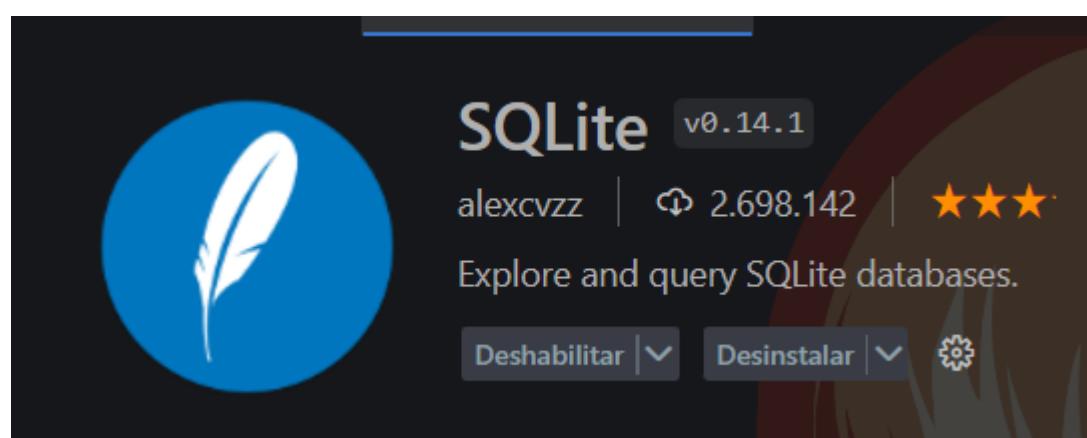
En la parte superior se puede ver donde esta el archivo

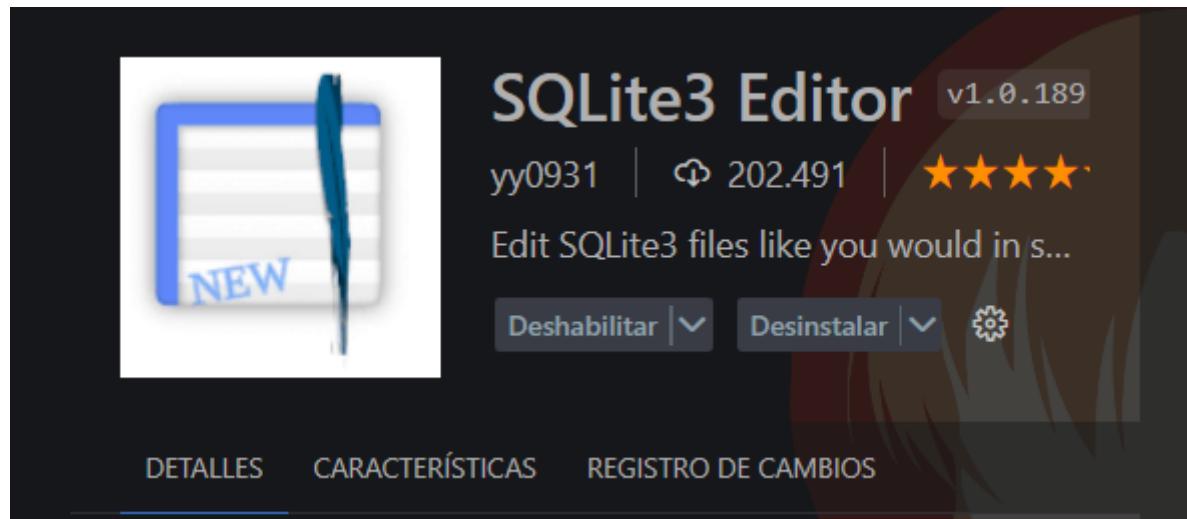
```

Script > DDLsql > ...
 ⚒ Connect | ⚒ Connect and Open Panel | 🔒 Active Connection | Comment Code |
1 -- database: ../../database/EXOBOT.sqlite
Comment Code

```

extensiones necesarias:





## Las estructuras

```

tmp.drawio Desktop DML.sql Script 1 DDL.sql Script 1 × ⌂ ⌃ ⌄ ⌅ ⌆ ⌇ ...
Script > DDL.sql
18 | ,FechaCrea DATETIME DEFAULT current_timestamp
19 |);
20 | CREATE TABLE ExaBot (
21 | IdExaBot INTEGER PRIMARY KEY autoincrement
22 | ,IdIABot INTEGER NOT NULL
23 | ,Nombre TEXT NOT NULL
24 | ,Serie TEXT NOT NULL
25 | ,FechaCrea DATETIME DEFAULT current_timestamp
26 | ,CONSTRAINT fk_IABot FOREIGN KEY (IdIABot) REFERENCES IABot(IdIABot)
27 |);
28 | CREATE TABLE PersonaTipo (
29 | IdPersonaTipo INTEGER PRIMARY KEY autoincrement
30 | ,Nombre TEXT NOT NULL UNIQUE
31 | ,Estado VARCHAR(1) DEFAULT('A') CHECK (Estado IN ('A','X'))
32 | ,FechaCrea DATETIME DEFAULT current_timestamp
33 |);
34 | CREATE TABLE Persona (
35 | IdPersona INTEGER PRIMARY KEY autoincrement
36 | ,IdPersonaTipo INTEGER NOT NULL REFERENCES PersonaTipo(IdPersonaTipo)
37 | ,Cedula TEXT NOT NULL UNIQUE
38 | ,Nombre TEXT NOT NULL
39 | ,FechaCrea datetime default current_timestamp
40 |);
41 |

```

## datos iniciales del archivo

```

Comment Code
/*
CopyRight
autor: pat_mlc
Fecha: 17jul2k14 I
description: insert
*/

```

se debe colocar inicio para poder conectar crear la base de datos allí

```
C:\Users\HomeOneDrive\ExamenIBP\GCEcuAFauna> You hace 4 horas | 1 author (You)
-- database: ./DataBase/GCEcuAFauna.sqlite
```

investigar como funciona el sqlite

```
> Execute (Shift + Enter)
DELETE FROM PersonaTipo

PROBLEMS OUTPUT DEBUG CONSOLE PORTS COMMENTS ...
<
INSERT INTO PersonaTipo
(Nombre) VALUES
('Soldado')
('Mecanico')
('Experto Ingles')
('Experto español');

> Execute (Shift + Enter)
INSERT INTO Persona
(IdPersonaTipo ,Cedula ,Nombre) VALUES
('321654', 'Pepe sanchez')
```

Conexión y creación de la base de datos, ejemplificada anteriormente en un archivo sql.

The screenshot shows a Visual Studio Code interface with several tabs open. The main area displays an SQLite database named 'ExaBot2k24.sqlite'. A 'Query' tab is active, showing the SQL command: 'SELECT \* FROM Persona'. Below the query, the results are displayed in a table:

|   | IdPersona | IdPersonaTipo | Nombre            | Cedula |
|---|-----------|---------------|-------------------|--------|
| 1 | 5         | 1             | 'Soldado'         | 32132  |
| 2 | 6         | 2             | 'Mecanico'        | 32353  |
| 3 | 7         | 3             | 'Experto Ingles'  | 32154  |
| 4 | 8         | 4             | 'Experto español' | 87546  |

On the left, there is a code editor showing an SQL script named 'DML\_ExaBotK24.sql'. The script contains various DDL and DML statements for creating tables and inserting data into them. The code editor has syntax highlighting and code completion features.

## Clases de la Interfaz grafica

Presentaciones de errores, colores, alineamientos, letras, imágenes, etc en un IAStyle que pueden ser llamado a las clases de los botones creados.

```

public static final int ALIGNMENT_LEFT = SwingConstants.LEFT;
public static final int ALIGNMENT_RIGHT = SwingConstants.RIGHT;
public static final int ALIGNMENT_CENTER= SwingConstants.CENTER;

public static final Cursor CURSOR_HAND = new Cursor(Cursor.HAND_CURSOR);
public static final Cursor CURSOR_DEFAULT = new Cursor(Cursor.DEFAULT_CURSOR);

public static final URL URL_MAIN = IASTyle.class.getResource("/UserInterface/Resource/Img/IABot.png");
public static final URL URL_LOGO = IASTyle.class.getResource("/UserInterface/Resource/Img/Logo.png");
public static final URL URL_SPLASH= IASTyle.class.getResource("/UserInterface/Resource/Img/Splash.png");

public static final CompoundBorder createBorderRect(){
 return BorderFactory.createCompoundBorder(new LineBorder(Color.lightGray),
 new EmptyBorder(5, 5, 5, 5));
}

public static final void showMsg(String msg){
 JOptionPane.showMessageDialog(null, msg, "😊 IABot", JOptionPane.INFORMATION_MESSAGE);
}
public static final void showMsgError(String msg){
 JOptionPane.showMessageDialog(null, msg, "💀 IABot", JOptionPane.OK_OPTION);
}
public static final boolean showConfirmYesNo(String msg){
 return (JOptionPane.showConfirmDialog(null, msg, "😢 IABot", JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION);
}
}

```

Clases de los botones creados, que ayudaran a la creacion y materializacion de los botones en un panel, dandole un diseño nuevo y personalizable a los botones.

```

public class GCButton extends JButton implements MouseListener {
 public GCButton(String text){
 customizeComponent(text);
 }
 public GCButton(String text, String iconnew){
 customizeComponent(text, iconnew);
 }

 public void customizeComponent(String text, String iconnew){

 setsize(width:20, height:70);
 addMouseListener(this);
 customizeComponent(text);
 setBounds(x:50, y:30, width:100, height:20);

 setIcon(new ImageIcon(iconnew));
 setFont(GCIAStyle.GCFONT);
 }
 public void customizeComponent(String text) {
 setText(text);
 setOpaque(isOpaque:true);
 setFocusPainted(b:false);
 setBorderPainted(b:true);
 setContentAreaFilled(b:true);
 setForeground(GCIAStyle.GCCOLOR_FONT_DARK);
 setHorizontalTextAlignment(GCIAStyle.GCALIGNMENT_CENTER);
 setFont(GCIAStyle.GCFONT);
 setBackground(GCIAStyle.GCCOLOR_FONT_LIGHT);

 setCursor(new Cursor(Cursor.HAND_CURSOR));
 }
}

```

Clase para el panel de una tabla llamada sexo donde se añaden botnes que cumpliran una funcion especifica que se le coloca a cada boton conforme se va a elaborando la interfaz.

The screenshot shows a Java IDE interface. The left pane displays the code for `SexoPanel.java`, which extends `Form`. The code includes methods for customizing components like `txtIdSexo`, `txtNombre`, and `pnlBtnPage`, and setting up layouts using `GridBagLayout` and `GridBagConstraints`. The right pane, titled "EXPLORER", shows the project structure under "EXOBOT". The "src" folder contains packages for "BusinessLogic", "DataAccess", "Framework", "UserInterface", and "Form". Under "Form", files like `LocalidadPanel.java`, `LoginPanel.java`, `MainForm.java`, `MainPanel.java`, `MenuPanel.java`, `SexoPanel.java`, and `SplashScreenForm.java` are listed. Other files in the project include `GUI.java`, `IAStyle.java`, `Whiterun.java`, `App.java`, and `README.md`.

```

public void customizeComponent() {
 setLayout(new GridBagLayout());
 GridBagConstraints gbc = new GridBagConstraints();

 txtIdSexo.setEnabled(false);
 txtIdSexo.setBorderLine();
 txtNombre.setBorderLine();

 pnlBtnPage.add(btnPageInt);
 pnlBtnPage.add(btnPageAnt);
 pnlBtnPage.add(new PatLabel(" Page: "));
 pnlBtnPage.add(lblTotalReg); //cambiar
 pnlBtnPage.add(new PatLabel(") "));
 pnlBtnPage.add(btnPageSig);
 pnlBtnPage.add(btnPageFin);

 pnlBtnRow.add(btnRowInt);
 pnlBtnRow.add(btnRowAnt);
 pnlBtnRow.add(lblTotalReg);
 pnlBtnRow.add(btnRowSig);
 pnlBtnRow.add(btnRowFin);

 pnlBtnCRUD.add(btnNuevo);
 pnlBtnCRUD.add(btnGuardar);
 pnlBtnCRUD.add(btnCancelar);
 pnlBtnCRUD.add(btnEliminar);
 pnlBtnCRUD.setBorder(IAStyle.createBorderRect());
}

gbc.insets = new Insets(5, 5, 5, 5);

gbc.gridx = 0;
gbc.gridy = 0;
gbc.gridwidth = 2;
}

```

Se pueden generar varios paneles, estos pueden ser metidos dentro de una contendor asignandole un espacio, ya sea arriba, medio o abajo.

The screenshot shows a Java IDE interface. The left pane displays the code for `GCMainForm.java`, which extends `JFrame`. The code initializes panels `GCCedNomPanel` and `GCPanelCentral`, sets the title, size, and location for the frame, and adds them to the content pane using a `BorderLayout`. The right pane, titled "EXPLORER", shows the project structure under "EXOBOT". The "src" folder contains packages for "Framework", "Interfaz", "Customer", and "Form". Under "Form", files like `GCMainForm.java`, `GCCedNomPanel.java`, `GCPanelCentral.java`, and `GCSplash.java` are listed. Other files in the project include `Image.java`, `IAStyle.java`, `App.java`, `MainForm.java`, and `README.md`. A separate project "GCEcuFauna2K24A" is also visible in the "JAVA PROJECTS" section.

```

public class GCMainForm extends JFrame{
 GCCedNomPanel GCCedNom = new GCCedNomPanel();
 GCPanelCentral GCPcentral = new GCPanelCentral();

 public GCMainForm(String tilteApp){
 customizeComponent(tilteApp);
 }

 private void customizeComponent(String tilteApp) {
 setTitle(tilteApp);
 setSize(width:810, height:570); You, hace 5 horas + examenII
 setResizable(resizable:false);
 setLocationRelativeTo(c:null);
 setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 setBackground(GCIAStyle.GCCOLOR_FONDO);

 Container GContainer = getContentPane();
 GContainer.setLayout(new BorderLayout());
 // Agregar los paneles al contenedor
 GContainer.add(GCCedNom, BorderLayout.NORTH);
 GContainer.add(GCPcentral, BorderLayout.CENTER);
 setVisible(b:true);
 }
}

```

Allí se pueden añadir los paneles, y dentro de un panel tambien se pueden añadir estos mini paneles, donde se les añadiran botones pequeños, logos y demás. como se muestra a continuacion.

```

A GCPanelCentral.java 7 X A GCButton.java | A GCDAOHormiga.java 1 A GCMainForm.java | A App.java
src > Interfaz > Form > A GCPanelCentral.java > GCPanelCentral() GCPanelCentral()
41 public class GCPanelCentral extends JPanel{
42 public GCPanelCentral() {
43 cargarProvinciasDesdeDB();
44 //configurarAcciones();
45
46 // Configuración del panel superior
47 JPanel GCTopPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
48 try {
49 Image GClogo = ImageIO.read(GCIAStyle.GCURL_LOGOSF);
50 GClogo = GClogo.getScaledInstance(width:100, height:100, Image.SCALE_SMOOTH);
51 GCTopPanel.add(new JLabel(new ImageIcon(GClogo)));
52 } catch (IOException e) {
53 e.printStackTrace();
54 }
55 GCTopPanel.add(new GCLabel3(text:"Hormiguero Virtual
56 "));
56 GCTopPanel.add(GCbtnCrear, FlowLayout.RIGHT);
57 GCbtnCrear.addActionListener(e -> {
58 int respuesta = JOptionPane.showConfirmDialog(
59 parentComponent:null,
60 message:"¿Estás seguro de crear una Hormiga Larva?",
61 title:"Confirmación",
62 JOptionPane.YES_NO_OPTION,
63 JOptionPane.QUESTION_MESSAGE
64);
65
66 if (respuesta == JOptionPane.YES_OPTION) {
67 int numeroRegistro = GCmodel.getRowCount() + 1;
68 if (!provincias.isEmpty()) {
69 int randomIndex = (int) (Math.random() * provincias.size());
70 String provinciaAleatoria = provincias.get(randomIndex);
71 GCmodel.addRow(new Object[]{numeroRegistro, "Larva", provinciaAleatoria, "Asexual", null, null, "VIVA",
72 });
73 }
74 });
75 add(GCTopPanel, BorderLayout.NORTH);
76 }
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109

```

```

// Configuración del panel central para mostrar la tabla
JPanel GCgridPanel = new JPanel(new BorderLayout());
JTable table = new JTable(GCmodel);
JScrollPane GCscrollPane = new JScrollPane(table);
GCscrollPane.setPreferredSize(new Dimension(width:760, height:200));
add(GCgridPanel, BorderLayout.CENTER);
GCgridPanel.add(GCscrollPane, BorderLayout.CENTER);

```

```
// Configuración del panel inferior
JPanel GCBottomPanel = new JPanel(new GridLayout(rows:3, cols:2, hgap:80, vgap:10));
GCBbutton GCGenoAlimento = new GCBbutton(text:"GenoAlimento");
GCBbutton GCIgestaNatativa = new GCBbutton(text:"Ingesta Nativa");
GCBbutton3 GCBtnAlimenGeno = new GCBbutton3(text:"Alimentar GenoAlimento");
GCBbutton3 GCBtnAlimenNat = new GCBbutton3(text:"Alimentar Ingesta Nativa");

JPopupMenu GCGenoAlimentoMenu = new JPopupMenu();
JMenuItem GCX = new JMenuItem(text:"X");
JMenuItem GCXX = new JMenuItem(text:"XX");
JMenuItem GCXY = new JMenuItem(text:"XY");

JPopupMenu GCIgestaNatativaMenu = new JPopupMenu();
JMenuItem GCCar = new JMenuItem(text:"Carnívoro");
JMenuItem GCHer = new JMenuItem(text:"Herbívoro");
JMenuItem GCOmn = new JMenuItem(text:"Omnívoro");
JMenuItem GCIIns = new JMenuItem(text:"Insectívoro");
JMenuItem GCNec = new JMenuItem(text:"Nectarívoro");

GCX.addActionListener(e -> GCGenoAlimento.setText(GCX.getText()));
GCXX.addActionListener(e -> GCGenoAlimento.setText(GCXX.getText()));
GCXY.addActionListener(e -> GCGenoAlimento.setText(GCXY.getText()));
GCCar.addActionListener(e -> GCIgestaNatativa.setText(GCCar.getText()));
GCHer.addActionListener(e -> GCIgestaNatativa.setText(GCHer.getText()));
GCOmn.addActionListener(e -> GCIgestaNatativa.setText(GCOmn.getText()));
GCIIns.addActionListener(e -> GCIgestaNatativa.setText(GCIIns.getText()));
```

## Semana 14

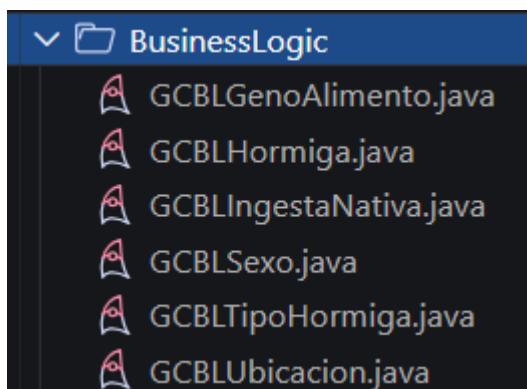
---

### Clase 36

#### Business Logic en Bases de Datos: Resumen

##### 1. ¿Qué es Business Logic en Bases de Datos?

- En el contexto de bases de datos, la Business Logic se refiere a las reglas y procesos que determinan cómo se gestionan y manipulan los datos almacenados. Esto incluye la validación de datos, la ejecución de operaciones condicionales, y la implementación de procesos específicos del negocio directamente en la base de datos.



##### 2. ¿Para qué sirve?

- La Business Logic en Bases de Datos permite a los desarrolladores implementar lógica.
- Permite la validación de datos antes de insertarlos en la base de datos.
- La Business Logic en bases de datos asegura que los datos almacenados sean consistentes, válidos y se manejen de acuerdo con las reglas de negocio. También permite automatizar tareas críticas, como el cálculo de totales, la generación de informes, y la validación de entradas, directamente en la capa de la base de datos, reduciendo la necesidad de lógica adicional en la aplicación.

The screenshot shows a database interface with a sidebar titled 'Tables' containing items like 'GCCatalogo', 'GCCatalogoTipo', 'GCHormiga', 'GCUbicacion', and 'sqlite\_sequence'. The main area displays the 'GCCatalogo' table with the following data:

|    | IdCatalogo | IdCatalogoTipo | Nombre      | Descripcion                 | Estado |
|----|------------|----------------|-------------|-----------------------------|--------|
| 1  | 1          | 1              | Larva       | Tipo de hormiga             | A      |
| 2  | 2          | 2              | Solado      | Tipo de hormiga             | A      |
| 3  | 3          | 3              | Zangano     | Tipo de hormiga             | A      |
| 4  | 4          | 4              | Reina       | Tipo de hormiga             | A      |
| 5  | 5          | 5              | Masculino   | tipos de sexualida          | A      |
| 6  | 6          | 6              | Femenino    | tipos de sexualida          | A      |
| 7  | 7          | 7              | Asexual     | tipos de sexualida          | A      |
| 8  | 8          | 8              | Carnívoro   | Tipo de Dieta de la hormiga | A      |
| 9  | 9          | 9              | Herbívoro   | Tipo de Dieta de la hormiga | A      |
| 10 | 10         | 10             | Omnívoro    | Tipo de Dieta de la hormiga | A      |
| 11 | 11         | 11             | Insectívoro | Tipo de Dieta de la hormiga | A      |
| 12 | 12         | 4              | XX          | Tipo de Genoalimento        | A      |
| 13 | 13         | 4              | XY          | Tipo de Genoalimento        | A      |
| 14 | 14         | 4              | X           | Tipo de Genoalimento        | A      |
| 15 | 15         | 3              | Nectarívoro | Tipo de Dieta de la hormiga | A      |

### 3. ¿Cómo se usa?

- **Triggers:**

Son procedimientos almacenados que se ejecutan automáticamente en respuesta a ciertos eventos en la tabla, como inserciones, actualizaciones o eliminaciones. Ejemplo: un trigger que ajusta el inventario cada vez que se inserta una nueva orden.

- **Stored Procedures:**

Son funciones almacenadas en la base de datos que pueden ser llamadas desde la aplicación para ejecutar operaciones complejas. Ejemplo: un procedimiento que calcula descuentos en función de la cantidad de productos comprados.

- **Constraints:**

Se utilizan para asegurar la integridad de los datos, como claves foráneas, claves primarias y validaciones de datos.

- **Views:**

Son consultas predefinidas que actúan como tablas virtuales, simplificando el acceso a datos complejos.

```

private void loadRow() {
 try {
 rowNum = 1;
 sexoDAO = sexoBL.getBy(rowNum);
 idRowMaxSexo = sexoBL.getRowCount();
 } catch (Exception e) {
 IASTyle.showMsg(e.getMessage());
 }
}

private void showRow() {
 boolean sexoNull = (sexoDAO == null);
 txtRowNum.setText((sexoNull) ? " " : sexoDAO.getIdCatalogo().toString());
 txtNombre.setText((sexoNull) ? " " : sexoDAO.getNombre());
 lblTotalReg.setText(rowNum.toString() + " de " + idRowMaxSexo.toString());
}

private void btnNuevoClick() {
 sexoDAO = null;
 showRow();
}

private void btnGuardarClick() {
 boolean sexoNull = (sexoDAO == null);
 // String buttonText = ((JButton) e.getSource()).getText();
 try {
 if (IAStyle.showConfirmYesNo("¿Seguro que desea " + ((sexoNull) ? "AGREGAR ?" : "ACTUALIZAR ?"))){

 if (sexoNull)
 sexoDAO = new SexoDTO(txtNombre.getText().trim());
 else
 sexoDAO.setNombre(txtNombre.getText());

 if(!((sexoNull) ? sexoBL.add(sexoDAO) : sexoBL.update(sexoDAO)))
}

```

Launchpad    0 34 1 0 Java: Ready    Share Code Link    Explain Code    Comment Code    Find Bugs    Code Chat

#### 4. ¿Cómo funciona?

- **Centralización:**

La lógica de negocio en la base de datos permite centralizar las reglas de negocio, asegurando que todas las aplicaciones y usuarios interactúen con los datos de manera coherente.

- **Consistencia:**

La implementación de la lógica de negocio en la base de datos.

- **Automatización:**

A través de triggers y procedimientos almacenados, la base de datos puede automatizar tareas repetitivas y garantizar que las reglas de negocio se apliquen sin intervención manual.

- **Validación y Seguridad:**

Mediante constraints y otros mecanismos, se garantiza que solo datos válidos y consistentes se almacenen en la base de datos, reduciendo errores y problemas de integridad.

- **Optimización:**

Al ejecutar lógica directamente en la base de datos, se puede reducir la carga en la aplicación, mejorar el rendimiento y aprovechar las optimizaciones nativas del motor de base de datos.

You, hace 7 horas | 1 author (You)

package BusinessLogic;



import java.util.List;

import DataAccess.DAO.GCDAOHormiga;

import DataAccess.DTO.GCDTOHormiga;

You, hace 7 horas | 1 author (You)

public class GCBLHormiga {

    private GCDTOHormiga mjSexo;

    private GCDAOHormiga mjSDAO = new GCDAOHormiga();

    public GCBLHormiga(){}

    public List<GCDTOHormiga> getAll() throws Exception{

        List<GCDTOHormiga> lst = mjSDAO.readAll();

        for (GCDTOHormiga mjSDTO : lst)

            mjSDTO.setGCIIdHormiga(mjSDTO.getGCIIdHormiga());

        return lst;

}

    public GCDTOHormiga getBy(int mjIdReg) throws Exception{

        mjSexo = mjSDAO.readBy(mjIdReg);

        return mjSexo;

}

    public boolean add(GCDTOHormiga mjRegDTO) throws Exception{

        return mjSDAO.create(mjRegDTO);

}

    public boolean update(GCDTOHormiga mjRegDTO) throws Exception{

        return mjSDAO.update(mjRegDTO);

}

    public boolean delete(int mjIdReg) throws Exception{

        return mjSDAO.delete(mjIdReg);

You, hace 7 horas • examen

}

}

## Conclusión:

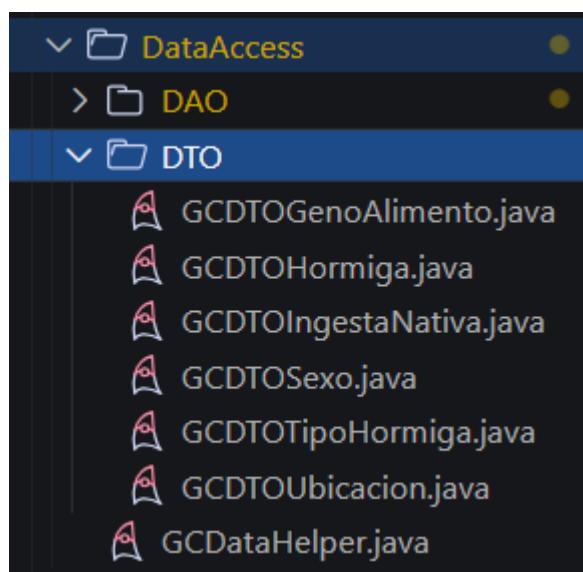
La Business Logic en bases de datos es crucial para mantener la integridad y consistencia de los datos, automatizar procesos y centralizar reglas de negocio, lo que asegura que todas las interacciones con la base de datos sigan las normas y políticas del negocio.

# Clase 37

## Clases DTO (Data Transfer Object) en Bases de Datos

### 1. ¿Qué es un DTO (Data Transfer Object)?

- Un DTO es un objeto simple que se utiliza para transferir datos entre diferentes capas de una aplicación, como entre la capa de presentación y la capa de negocio, o entre la capa de negocio y la capa de acceso a datos.
- Su objetivo es encapsular los datos necesarios para una operación específica, facilitando la conexión de la base de datos.
- Los DTOs no contienen lógica de negocio; solo tienen atributos (campos) y métodos getters/setters.



### 2. ¿Para qué sirven los DTOs?

- **Encapsulamiento de Datos:**

Los DTOs encapsulan los datos que se necesitan transferir, reduciendo el acoplamiento entre las capas y permitiendo que los datos se manejen de manera más controlada.

- **Optimización del Tráfico de Datos:**

Los DTOs permiten enviar solo los datos necesarios en una operación, lo que puede mejorar el rendimiento, especialmente en sistemas distribuidos o con llamadas a la red.

- **Separación de Preocupaciones:**

Facilitan la separación de las preocupaciones al dividir claramente la lógica de presentación, la lógica de negocio y el acceso a datos.

```

GPanelCentral.java 7 | GCDTOHormiga.java • | GCDAOHormiga.java 1 | GCMainForm.java | App.java

DataAccess > DTO > GCDTOHormiga.java > GCDTOHormiga > toString()
You, hace / horas | 1 author (You)

public class GCDTOHormiga {

 private Integer gcIdHormiga;
 private Integer gcIdClgTipoHormiga;
 private Integer gcIdClgIngestaNativa;
 private Integer gcIdClgGenoAlimento;
 private Integer gcIdClgSexo;
 private Integer gcIdUbicacion;
 private String gcEstado;
 private String gcFechaCreacion;
 private String gcFechaModifica;

 public GCDTOHormiga(){}
 public GCDTOHormiga(Integer gcIdHormiga, Integer gcIdClgTipoHormiga, Integer gcIdClgIngestaNativa,
 Integer gcIdClgGenoAlimento, Integer gcIdClgSexo, Integer gcIdUbicacion, String gcEstado,
 String gcFechaCreacion, String gcFechaModifica) {
 this.gcIdHormiga = gcIdHormiga;
 this.gcIdClgTipoHormiga = gcIdClgTipoHormiga;
 this.gcIdClgIngestaNativa = gcIdClgIngestaNativa;
 this.gcIdClgGenoAlimento = gcIdClgGenoAlimento;
 this.gcIdClgSexo = gcIdClgSexo;
 this.gcIdUbicacion = gcIdUbicacion;
 this.gcEstado = gcEstado;
 this.gcFechaCreacion = gcFechaCreacion;
 this.gcFechaModifica = gcFechaModifica;
 }
 public Integer getGCIdHormiga() {
 return gcIdHormiga;
 }
 public void setGCIdHormiga(Integer gcIdHormiga) {
 this.gcIdHormiga = gcIdHormiga;
 }
 public Integer getGCIdClgTipoHormiga() {
 return gcIdClgTipoHormiga;
 }
 public void setGCIdClgTipoHormiga(Integer gcIdClgTipoHormiga) {
 ...
 }
}

```

### 3. ¿Cómo se usan los DTOs en la interacción con Bases de Datos?

- **Mapeo de Datos:**

En la capa de acceso a datos (DAO), los DTOs se utilizan para mapear los datos obtenidos de la base de datos (generalmente a través de consultas SQL) a objetos en la aplicación.

- **Transferencia entre Capas:**

Los DTOs se pasan entre la capa de acceso a datos y la capa de negocio (Business Logic) para realizar operaciones como la validación, el procesamiento o la transformación de los datos antes de enviarlos a la capa de presentación.

- **Sincronización con la Base de Datos:**

Los DTOs también pueden ser utilizados para recibir datos de la capa de presentación, que luego son transformados y almacenados en la base de datos a través de la capa de acceso a datos.



```
java 3 | SexoPanel.java | SexoDTO.java X | FLS.dio | pizarra.drawio
> src > DataAccess > DTO > SexoDTO.java > SexoDTO > setNombre(String)
public class SexoDTO {

 private Integer idCatalogoTipo ;
 private String nombre ;
 private String descripcion ;
 private String estado ;
 private String fechaCreacion ;
 private String fechaModifica ;

 public SexoDTO(Integer RowNum , Integer idCatalogo, Integer idCatalogoTipo, String nombre, String descripcion, String estado,
 String fechaCreacion, String fechaModifica) {
 this.RowNum = RowNum;
 this.idCatalogo = idCatalogo;
 this.idCatalogoTipo = idCatalogoTipo;
 this.nombre = nombre;
 this.descripcion = descripcion;
 this.estado = estado;
 this.fechaCreacion = fechaCreacion;
 this.fechaModifica = fechaModifica;
 }

 public SexoDTO(){}

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

 public Integer getRowNum() {
 return RowNum;
 }

 public void setRowNum(Integer rowNum) {
 RowNum = rowNum;
 }
}
```

#### 4. Conexión entre DTOs y Business Logic (BL)

- **Intermediarios de Datos:**

Los DTOs actúan como intermediarios entre la base de datos y la lógica de negocio. Reciben datos de la base de datos a través de la capa DAO y los pasan a la capa de negocio para su procesamiento.

- **Transformación de Datos:**

En la lógica de negocio, los DTOs pueden ser transformados o combinados para crear estructuras de datos que cumplen con las reglas del negocio antes de ser enviados a la capa de presentación.

- **Validación y Procesamiento:**

Aunque los DTOs no deben contener lógica de negocio, los datos dentro de ellos son validados y procesados en la capa de negocio para asegurar que cumplen con las reglas del negocio antes de cualquier operación en la base de datos o en la interfaz de usuario.

#### Conclusión:

Los DTOs son fundamentales para estructurar el flujo de datos entre la base de datos y las diferentes capas de una aplicación, facilitando la separación de responsabilidades, mejorando el rendimiento, y asegurando que la lógica de negocio se aplique de manera efectiva en todo el sistema.

## Clase 38

### Clases DAO (Data Access Object) en Bases de Datos

## 1. ¿Qué es un DAO (Data Access Object)?

- Un DAO es un patrón de diseño que se utiliza para encapsular el acceso a la base de datos. Proporciona una interfaz abstracta para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre una base de datos, permitiendo que el resto de la aplicación no dependa directamente de la base de datos.

## 2. ¿Para qué sirven los DAOs?

- **Abstracción del Acceso a Datos:**

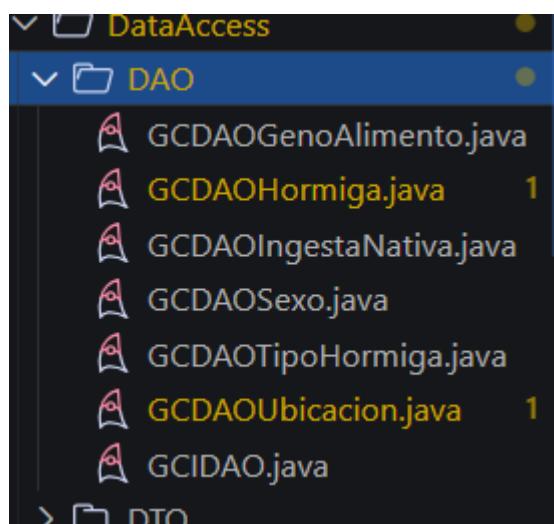
Los DAOs ocultan los detalles de cómo los datos son almacenados y recuperados, permitiendo que otras capas de la aplicación interactúen con la base de datos sin conocer sus detalles internos.

- **Reutilización de Código:**

Centralizan el código de acceso a datos, permitiendo su reutilización en diferentes partes de la aplicación.

- **Facilitan el Mantenimiento:**

Al aislar la lógica de acceso a datos en un solo lugar, se facilita el mantenimiento y la actualización de la capa de datos sin afectar el resto de la aplicación.



## 3. ¿Cómo se usan los DAOs en la interacción con Bases de Datos?

- **Implementación de Operaciones CRUD:**

Los DAOs implementan métodos para crear, leer, actualizar y eliminar registros en la base de datos. Por ejemplo, `saveUser(User user)`, `getUserById(int id)`, `updateUser(User user)`, `deleteUser(int id)`.

- **Conexión con la Base de Datos:**

Los DAOs manejan la creación y cierre de conexiones a la base de datos, la ejecución de consultas SQL, y la transformación de los resultados en objetos de la aplicación (a menudo DTOs).

- **Manejo de Excepciones:**

Encapsulan el manejo de errores y excepciones relacionadas con la base de datos, asegurando que estas no se propaguen directamente a otras capas de la aplicación.

```
You, hace 7 horas | 1 author (You)
public class GCDAOHormiga extends GCDATAHelper implements GCDAO<GCDTOHormiga> {
 You, hace 23 horas • database
 @Override
 public boolean create(GCDTOHormiga entity) throws Exception {
 DateTimeFormatter dtf = DateTimeFormatter.ofPattern(pattern:"yyyy-MM-dd HH:mm:ss");
 LocalDateTime now = LocalDateTime.now();
 String query = " INSERT INTO GCHORMIGA (IdClgTipoHormiga, IdClgIngestaNativa, IdClgGenoAlimento, IdClgSexo,
 try {
 Connection conn = openConnection();
 PreparedStatement pstmt = conn.prepareStatement(query);
 pstmt.setInt(parameterIndex:1, entity.getGCIdClgTipoHormiga());
 pstmt.setInt(parameterIndex:2, entity.getGCIdClgIngestaNativa());
 pstmt.setInt(parameterIndex:3, entity.getGCIdClgGenoAlimento());
 pstmt.setInt(parameterIndex:4, entity.getGCIdClgSexo());
 pstmt.setInt(parameterIndex:5, entity.getGCIdUbicacion());
 pstmt.setString(parameterIndex:6, dtf.format(now).toString());

 pstmt.executeUpdate();
 return true;
 }
 catch (SQLException e) {
 throw new GCEException(e.getMessage(), getClass().getName(), metodo:"create()");
 }
 }

 public List<GCDTOHormiga> readAll() throws Exception {
 List <GCDTOHormiga> lst = new ArrayList<>();
 String query =" SELECT IdHormiga
 "
 +" ,IdClgTipoHormiga
 "
 +" ,IdClgIngestaNativa
 "
 +" ,IdClgGenoAlimento
 "
 +" ,IdClgSexo
 "
 }
}
```

#### 4. Conexión entre DAOs y Business Logic (BL)

- **Interacción Directa:**

La capa de negocio (Business Logic) utiliza los DAOs para interactuar con la base de datos. La lógica de negocio invoca métodos del DAO para obtener, modificar, o eliminar datos.

- **Separación de Responsabilidades:**

Los DAOs se centran exclusivamente en el acceso a los datos, mientras que la Business Logic se encarga de procesar esos datos según las reglas del negocio. Esto asegura que cada capa tenga una responsabilidad clara y separada.

- **Transformación de Datos:**

Los DAOs transforman los datos de la base de datos en objetos de la aplicación (DTOs o entidades) que luego son procesados por la Business Logic. Una vez procesados, los DAOs también son responsables de actualizar la base de datos con los datos transformados.

#### Conclusión:

Los DAOs son esenciales para gestionar el acceso a la base de datos de manera estructurada y eficiente, proporcionando una interfaz clara para la Business Logic. Este patrón permite una mayor modularidad, mantenimiento y escalabilidad en aplicaciones que interactúan con bases de datos.

# Semana 15

## Clase 39

### Conexión de la Base de Datos en Java

#### 1. ¿Qué se necesita para establecer la conexión con una base de datos?

- **Driver JDBC:**

Un controlador JDBC (Java Database Connectivity) es necesario para permitir que Java se comunique con la base de datos específica (MySQL, PostgreSQL, SQLite, etc.).

- **URL de Conexión:**

Una URL que describe la ubicación de la base de datos y las credenciales necesarias para conectarse.

- **Credenciales:**

Nombre de usuario y contraseña para autenticar la conexión con la base de datos.

#### 2. Pasos para establecer la conexión con la base de datos

- **Cargar el Driver JDBC:**

```
Class.forName("org.sqlite.JDBC"); // Para SQLite, por ejemplo
```

Este paso carga el driver necesario para establecer la conexión.

- **Crear la conexión:**

```
String url = "jdbc:sqlite:path_to_database.db"; // URL de la base de
dados
Connection conn = DriverManager.getConnection(url);
```

Aquí se crea una instancia de **Connection**, que representa la conexión activa con la base de datos.

- **Manejo de Excepciones:**

```
try (Connection conn = DriverManager.getConnection(url)) {
 // Operaciones con la base de datos
} catch (SQLException e) {
 e.printStackTrace(); // Manejo de errores
}
```

Es importante manejar las excepciones para asegurarse de que los errores en la conexión no afecten la aplicación.

### 3. Uso de DAO para interactuar con la base de datos

- **Implementación de un DAO:**

- Crear una clase DAO, por ejemplo `UserDAO`, que maneje las operaciones específicas para la tabla `User`.
- Métodos típicos incluyen `save(User user)`, `findById(int id)`, `update(User user)`, `delete(int id)`.

```
public class UserDAO {
 private Connection conn;

 public UserDAO(Connection conn) {
 this.conn = conn;
 }

 public void save(User user) {
 String sql = "INSERT INTO users (name, email) VALUES (?, ?)";
 try (PreparedStatement stmt = conn.prepareStatement(sql)) {
 stmt.setString(1, user.getName());
 stmt.setString(2, user.getEmail());
 stmt.executeUpdate();
 } catch (SQLException e) {
 e.printStackTrace();
 }
 }

 // Métodos findById, update, delete, etc.
}
```

### 4. Uso de DTOs para transferir datos entre capas

- **Definición del DTO:**

- Crear una clase DTO, por ejemplo `UserDTO`, que contenga los campos relevantes.
- Los DTOs se utilizan para transferir datos entre capas de la aplicación.
- Por ejemplo, un DTO `UserDTO` podría tener campos como `id`, `name`, `email`, etc.
- Los DTOs pueden ser utilizados para transferir datos entre capas de la aplicación.
- Crear una clase DTO, por ejemplo `UserDTO`, que representa los datos que se transfieren entre la capa DAO y la capa de negocio.

```
public class UserDTO {
 private int id;
```

```
 private String name;
 private String email;

 // Getters y setters
}
```

- **Interacción entre DAO y Business Logic utilizando DTOs:**

- La Business Logic (por ejemplo, `UserService`) utiliza el DAO para acceder a la base de datos y maneja los DTOs para procesar los datos.

```
public class UserService {
 private UserDAO userDAO;

 public UserService(UserDAO userDAO) {
 this.userDAO = userDAO;
 }

 public void registerUser(UserDTO userDTO) {
 // Validar datos, aplicar reglas de negocio
 userDAO.save(userDTO);
 }

 // Otros métodos de negocio
}
```

## 5. Conexión General del Sistema

- **Paso 1:**

Establecer la conexión a la base de datos en el arranque de la aplicación.

- **Paso 2:**

Inyectar la conexión en los DAOs.

- **Paso 3:**

Usar los DAOs en la capa de negocio para interactuar con la base de datos.

- **Paso 4:**

Transferir datos entre la capa de presentación y la capa de negocio mediante DTOs.

### Diagrama del Flujo:

1. **Capa de Presentación:**

Interfaz de usuario -> Recibe datos de entrada.

- 2. **DTO:**

Transfiere datos desde la presentación hacia la lógica de negocio.

### 3. Capa de Negocio (BL):

Valida y procesa los datos -> Invoca métodos del DAO.

### 4. DAO:

Interactúa con la base de datos, ejecuta consultas SQL -> Devuelve resultados.

### 5. Capa de Negocio (BL):

Recibe resultados y aplica más lógica si es necesario.

### 6. DTO:

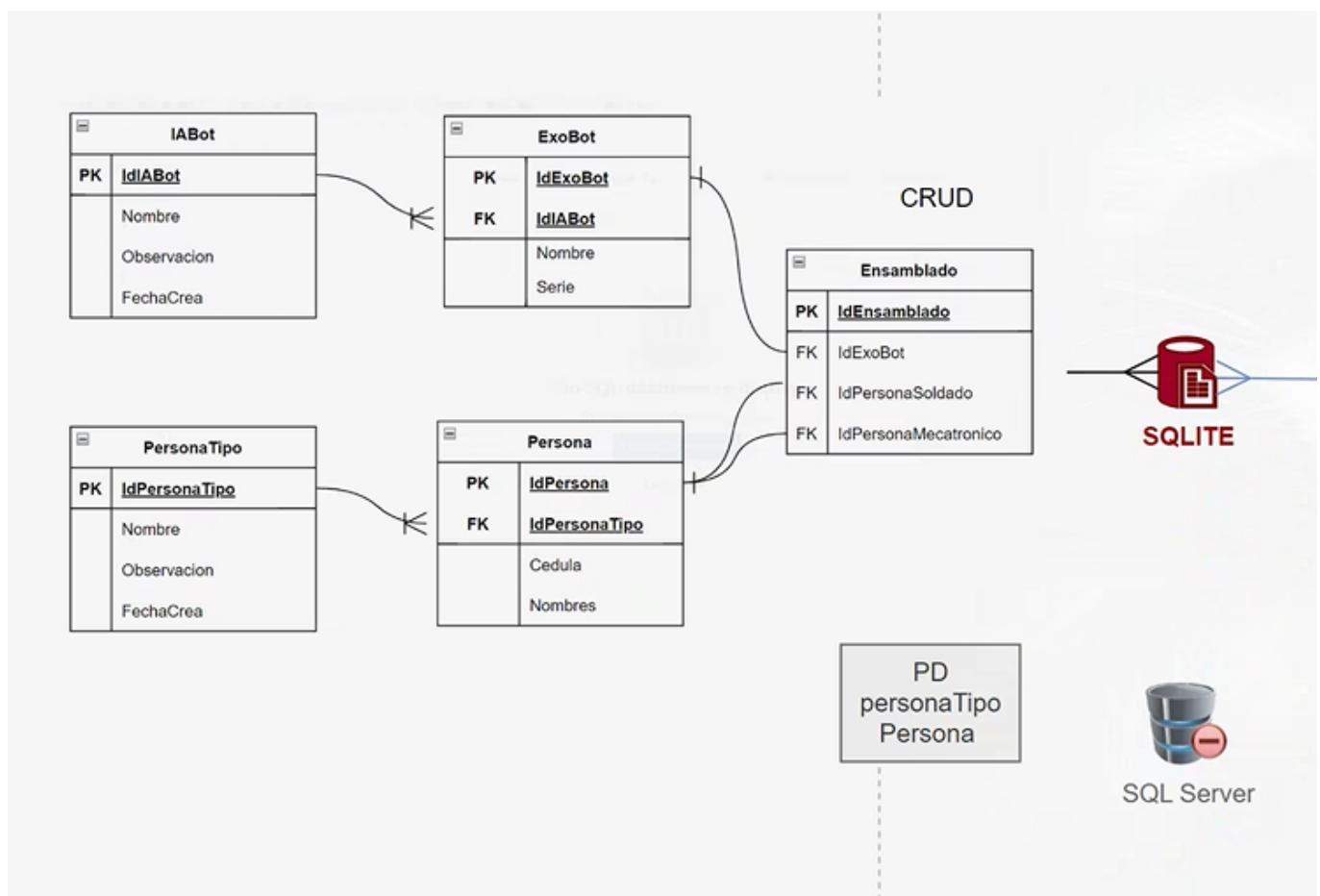
Transfiere datos procesados de vuelta a la presentación.

### Conclusión:

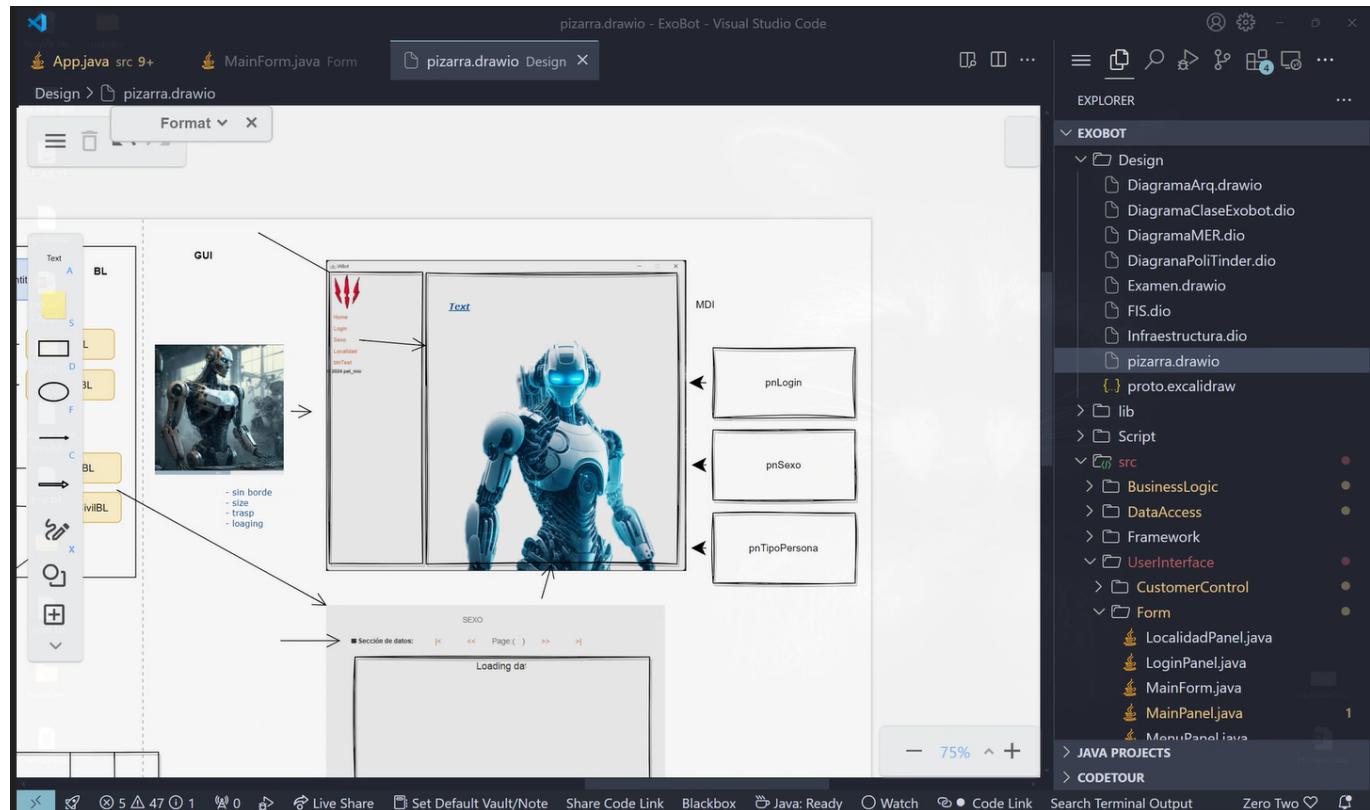
La conexión a la base de datos en Java se logra mediante JDBC y se maneja a través de DAOs para encapsular el acceso a datos. Los DTOs permiten transferir datos entre las capas de la aplicación, mientras que la lógica de negocio se encarga de procesar y validar los datos antes de interactuar con la base de datos, asegurando un sistema modular, escalable y mantenable.

## Clase 40

Diseño de la base de datos



## Clase 41



```

• MainForm.java - ExoBot - Visual Studio Code

src > UserInterface > Form > MainForm.java > customizeComponent(String)
 customizeComponent(@TitleApp);
 pnlMenu.btnAdd.addActionListener(e -> setPanel(new MainPanel()));
 pnlMenu.btnAddLogin.addActionListener(e -> setPanel(new LoginPanel()));
 pnlMenu.btnAddSexo.addActionListener(e -> setPanel(new SexoPanel()));
 pnlMenu.btnAddLocalidad.addActionListener(e -> setPanel(new MainPanel()));
 //agregar
 pnlMenu.btnAddTest.addActionListener(e -> IAStyle.showMsgError("mensaje de error")); }

 private void setPanel(JPanel formularioPanel) { ...
 //JOptionPane.showMessageDialog(this, "Seleccionaste Opción 3");

 private void customizeComponent(String titleApp) {
 setTitle(titleApp);
 setSize(950, 800);
 setResizable(false);
 setLocationRelativeTo(null); // Centrar en la pantalla
 setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

 // Crear un contenedor para Los dos paneles usando BorderLayout
 Container container = getContentPane();
 container.setLayout(new BorderLayout());

 // Agregar Los paneles al contenedor
 container.add(pnlMenu, BorderLayout.WEST);
 container.add(formularioPanel, BorderLayout.CENTER);
 setVisible(true);
 }
}

```

The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** • MainForm.java - ExoBot - Visual Studio Code
- Left Side:** Explorer sidebar showing the project structure under 'EXOBOT' (src, UserInterface, Form).
- Central Area:** Code editor showing Java code for 'MainForm.java'.
- Bottom:** Status bar with various icons and text.

```

 8
 9 import UserInterface.IAStyle;
10
11 Comment Code
12 public class MainForm extends JFrame{
13 JPanel pnlMenu = new JPanel();
14 JPanel pnlMain = new JPanel();
15
16 public MainForm(String titleApp) {
17 customizeComponent(titleApp);
18 pnlMenu.btnAdd.addActionListener(e -> setPanel(new MainPanel()));
19 pnlMenu.btnAddLogin.addActionListener(e -> setPanel(new LoginPanel()));
20 pnlMenu.btnAddSexo.addActionListener(e -> setPanel(new SexoPanel()));
21 pnlMenu.btnAddLocalidad.addActionListener(e -> setPanel(new MainPanel()));
22 //agregar I
23 pnlMenu.btnAddTest.addActionListener(e -> { IAStyle.showError("mensaje de error");});
24 }
25
26 > private void setPanel(JPanel formularioPanel) { ...
27
28 //JOptionPane.showMessageDialog(this, "Seleccionaste Opcion 3");
29
30 > private void customizeComponent(String titleApp) {[...
31
32 }
33
34
35
36
37
38
39
40
41
42
43

```

EXPLORER

- EXOBOT
  - .vscode
  - bin
  - DataBase
  - Design
  - lib
  - Script
  - src
    - BusinessLogic
    - DataAccess
    - Framework
    - UserInterface
      - CustomerControl
      - Form
        - LocalidadPanel.java
        - LoginPanel.java
        - MainForm.java
        - MainPanel.java
        - MenuPanel.java
        - SexoPanel.java
        - SplashScreenForm.java
    - GUI
    - Resource
      - IStyle.java
      - Whiterun.java

JAVA PROJECTS

CODETOUR

```

12
13 import UserInterface.IAStyle;
14 import UserInterface.CustomerControl.PatButton;
15
16 Comment Code
17 public class MenuPanel extends JPanel {
18 public PatButton btnHome = new PatButton("Home");
19 public PatButton btnLogin = new PatButton("Login");
20 public PatButton btnSexo = new PatButton("Sexo");
21 public PatButton btnLocalidad= new PatButton("Localidad");
22 public PatButton btnTest = new PatButton("btnTest");
23
24 public MenuPanel(){
25 customizeComponent();
26 }
27
28 private void customizeComponent() {
29 setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
30 setPreferredSize(new Dimension(350, getHeight()));
31
32 // add-Logo
33 try {
34 Image Logo = ImageIO.read(IASyle.URL_LOGO);
35 Logo = Logo.getScaledInstance(100, 100, Image.SCALE_SMOOTH);
36 add(new JLabel(new ImageIcon(Logo)));
37 } catch (IOException e) {
38 e.printStackTrace();
39 }
40
41 // add-botones
42 add(btnHome);
43 add(btnLogin);

```

EXPLORER

- EXOBOT
  - .vscode
  - bin
  - DataBase
  - Design
  - lib
  - Script
  - src
    - BusinessLogic
    - DataAccess
    - Framework
    - UserInterface
      - CustomerControl
      - Form
        - LocalidadPanel.java
        - LoginPanel.java
        - MainForm.java
        - MainPanel.java
        - MenuPanel.java
        - SexoPanel.java
        - SplashScreenForm.java
    - GUI
    - Resource
      - IStyle.java
      - Whiterun.java

JAVA PROJECTS

CODETOUR

The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** MenuPanel.java - ExoBot - Visual Studio Code
- Left Panel:** Shows the file structure of the EXOBOT project under the 'EXOBOT' heading.
- Code Editor:** Displays the Java code for `MenuPanel.java`. The code initializes a logo image and adds it to the panel, along with other buttons and a copyright label.
- Right Panel:** Shows the file tree for the EXOBOT project, including files like LocalidadPanel.java, LoginPanel.java, MainForm.java, etc.

```

src > UserInterface > Form > MenuPanel.java > customizeComponent()
 btnLocaL = new JButton("Localidad");
 btnTest = new JButton("btnTest");

public MenuPanel(){
 customizeComponent();
}

private void customizeComponent() {
 setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
 setPreferredSize(new Dimension(350, getHeight()));

 // add-logo
 try {
 Image Logo = ImageIO.read(IAStyle.URL_LOGO);
 Logo = Logo.getScaledInstance(100, 100, Image.SCALE_SMOOTH);
 add(new JLabel(new ImageIcon(Logo)));
 } catch (IOException e) {
 e.printStackTrace();
 }

 // add-botones
 add(btnHome);
 add(btnLogin);
 add(btnSexo);
 add(btnLocalidad);
 add(btnTest);

 // add-copyright
 add(new JLabel("\u00A9 2024 pat_mlc"));
}

```

Para poder poner imágenes

The screenshot shows the Java code for `MenuPanel.java` with a tooltip for the variable `URL_URL_LOGO`.

```

// add-Logo
try {
 Image Logo = ImageIO.read(IAStyle.URL_LOGO);
 Logo = Logo.getScaledInstance(100, 100, Image.SCALE_SMOOTH);
 add(new JLabel(new ImageIcon(Logo)));
} catch (IOException e) {
 e.printStackTrace();
}

```

## Semana 16

### Clase 42

traer todo el esqueleto para la base de datos.

diagrama de arquitectura 0,5 login punto extra para proceso de logueo con encriptacion

diagrama de clase 0,5

Mejorar el codigo, menos lineas

### Proyecto

Codigo

Interfaz Grafica

```
You, hace 4 días | 1 author (You)
public class panelCentral extends JFrame{
 PanelBotones pnlApart = new PanelBotones();
 JPanel pnlInicio = new PanelInicio();

 public panelCentral(String tilteApp) {
 customizeComponent(tilteApp);
 pnlApart.btnHOME.addActionListener(
 e -> setPanel(new PanelInicio()));
 pnlApart.btnREGISTRO.addActionListener(
 e -> setPanel(new RegistroPanel()));
 pnlApart.btnVENTAS.addActionListener(
 e -> setPanel(new VentasPanel()));
 //pnlApart.btnCOMPRA.addActionListener(
 e -> setPanel(new ComprasPanel()));
 pnlApart.btnINVENTARIO.addActionListener(
 e -> setPanel(new InventarioPanel()));
 pnlApart.btnFACTURACION.addActionListener(
 e -> setPanel(new FacturacionPanel()));
 pnlApart.btnExit.addActionListener(
 e -> System.exit(status:0));
 }

 private void setPanel(JPanel formPanel) {
 Container container = getContentPane();
 container.remove(pnlInicio);
 pnlInicio = formPanel;
 container.add(pnlInicio, BorderLayout.CENTER);
 revalidate();
 repaint();
 }
}
```

## Business Logic

AlejandroTTD, hace 5 días | 1 author (AlejandroTTD)

```
▽ public class Detalle_VentaBL {

 private Detalle_VentaDTO detalle_Venta;
 private Detalle_VentaDAO detalle_VentaDAO;

 public Detalle_VentaBL(){}

 ▽ public boolean create(Detalle_VentaDTO detalle_VentaDTO) throws Exception{
 return detalle_VentaDAO.create(detalle_VentaDTO);
 }

 ▽ public Detalle_VentaDTO readBy(Integer id) throws Exception {
 detalle_Venta = detalle_VentaDAO.readBy(id);
 return detalle_Venta;
 }

 ▽ public List<Detalle_VentaDTO> getAll() throws Exception {
 List<Detalle_VentaDTO> lst = detalle_VentaDAO.readAll();
 return lst;
 }

 ▽ public boolean update(Detalle_VentaDTO detalle_VentaDTO) throws Exception{
 return detalle_VentaDAO.update(detalle_VentaDTO);
 }

 ▽ public boolean delete (int id) throws Exception {
 return detalle_VentaDAO.delete(id);
 }
}
```

## Interfaz

AlejandroTTD, hace 5 días | 1 author (AlejandroTTD)

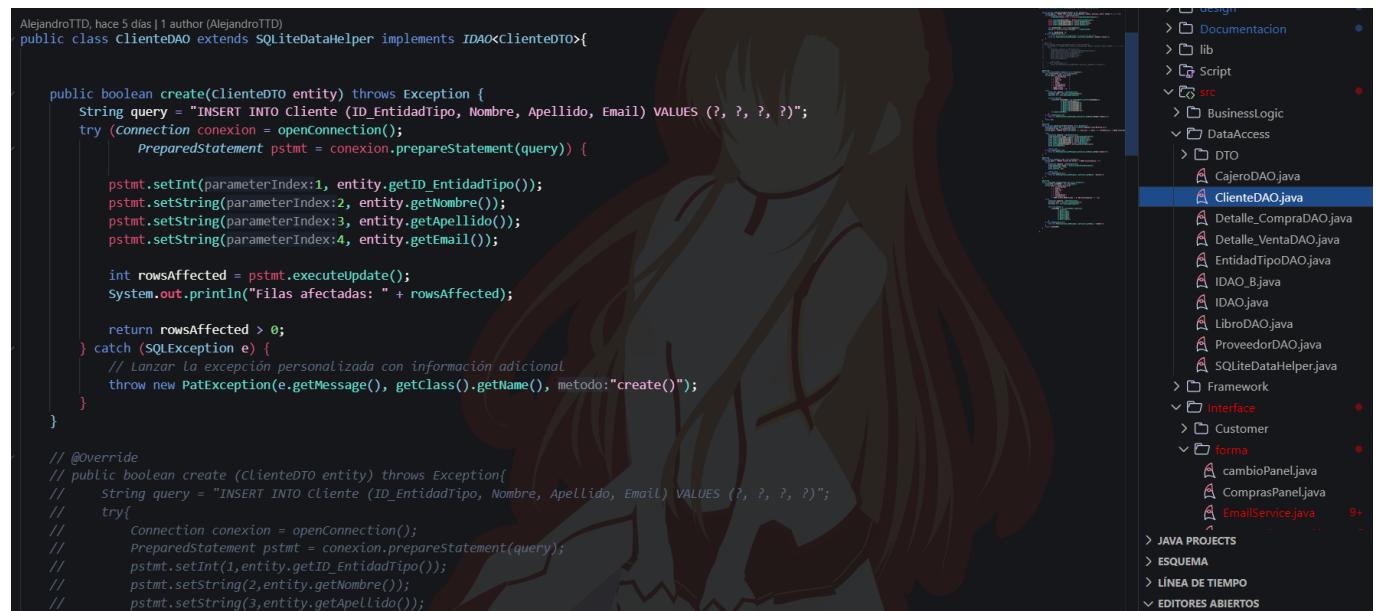
```

public interface IDAO_B <T>{
 public boolean create(T entity) throws Exception;
 public List<T> readAll() throws Exception;
 public boolean update(T entity) throws Exception;
 public boolean delete(String codigoB) throws Exception;

 public T readBy(String codigoB) throws Exception;
}

```

## DAO



AlejandroTTD, hace 5 días | 1 author (AlejandroTTD)

```

public class ClienteDAO extends SQLiteDataHelper implements IDAO<ClienteDTO>{

 public boolean create(ClienteDTO entity) throws Exception {
 String query = "INSERT INTO Cliente (ID_EntidadTipo, Nombre, Apellido, Email) VALUES (?, ?, ?, ?)";
 try (Connection conexion = openConnection();
 PreparedStatement pstmt = conexion.prepareStatement(query)) {
 pstmt.setInt(parameterIndex:1, entity.getID_EntidadTipo());
 pstmt.setString(parameterIndex:2, entity.getNombre());
 pstmt.setString(parameterIndex:3, entity.getApellido());
 pstmt.setString(parameterIndex:4, entity.getEmail());

 int rowsAffected = pstmt.executeUpdate();
 System.out.println("Filas afectadas: " + rowsAffected);

 return rowsAffected > 0;
 } catch (SQLException e) {
 // Lanzar la excepción personalizada con información adicional
 throw new PatException(e.getMessage(), getClass().getName(), metodo:"create()");
 }
 }

 // @Override
 // public boolean create (ClienteDTO entity) throws Exception{
 // String query = "INSERT INTO Cliente (ID_EntidadTipo, Nombre, Apellido, Email) VALUES (?, ?, ?, ?)";
 // try{
 // Connection conexion = openConnection();
 // PreparedStatement pstmt = conexion.prepareStatement(query);
 // pstmt.setInt(1,entity.getID_EntidadTipo());
 // pstmt.setString(2,entity.getNombre());
 // pstmt.setString(3,entity.getApellido());
 // }
 // }
}

```

The right side of the screenshot shows a file tree:

- design
- Documentacion
- lib
- Script
- src
  - BusinessLogic
  - DataAccess
    - DTO
      - CajeroDAO.java
      - ClienteDAO.java**
      - Detalle\_CompraDAO.java
      - Detalle\_VentaDAO.java
    - IDAO\_B.java
    - IDAO.java
    - LibroDAO.java
    - ProveedorDAO.java
    - SQLiteDataHelper.java
- Framework
- Interface
- Customer
- forma
  - cambioPanel.java
  - ComprasPanel.java
  - EmailService.java
- JAVA PROJECTS
- ESQUEMA
- LINERA DE TIEMPO
- EDITORES ABIERTOS

## DTO

```
public class CajeroDTO {
 private Integer ID_Cajero ;
 private Integer ID_EntidadTipo;
 private String Usuario ;
 private String Contrasena;
 private String Estado ;
 private String FechaCreacion;
 private String FechaModifica;

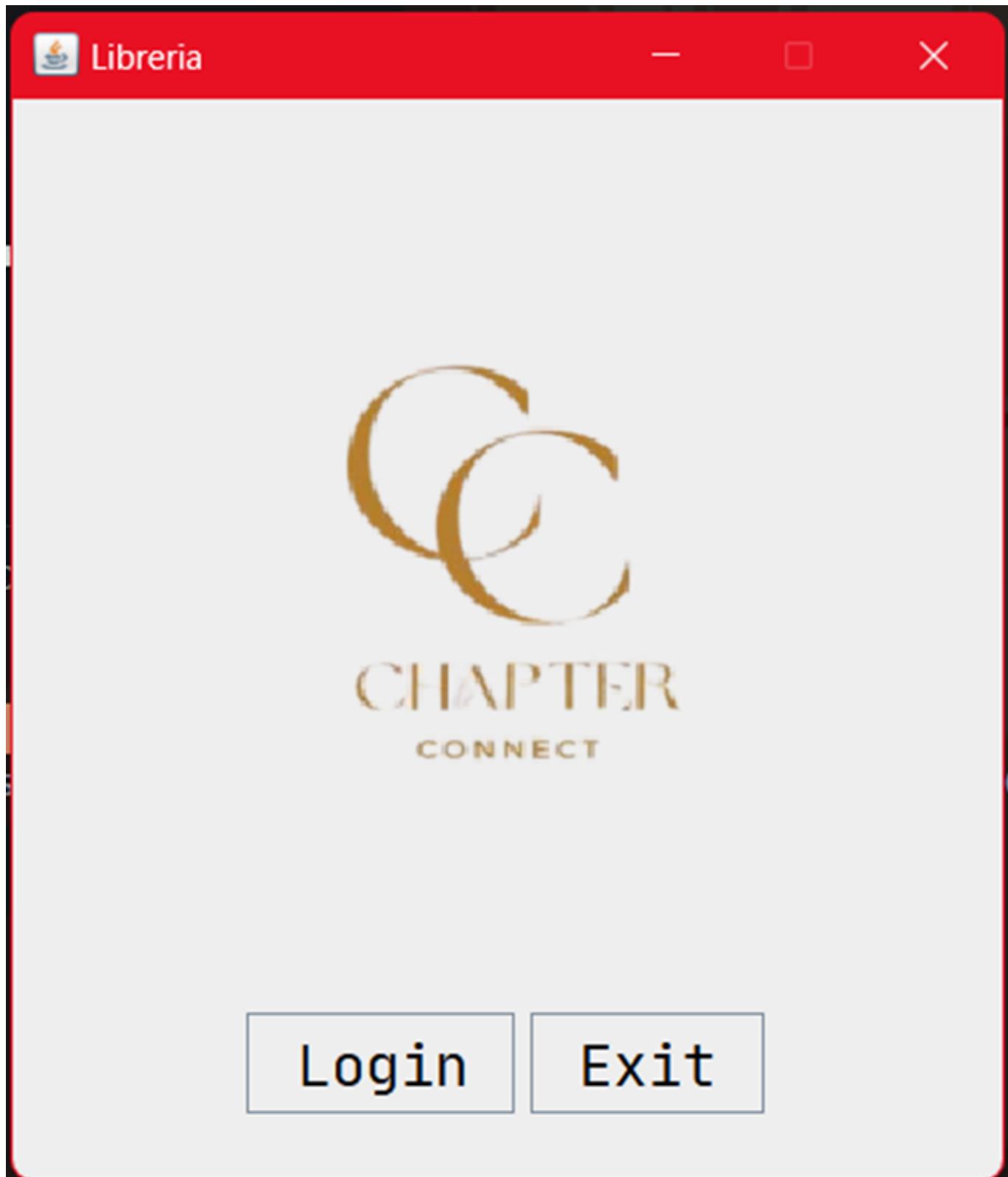
 public CajeroDTO(Integer iD_Cajero, Integer iD_EntidadTipo, String usuario, String contrasena, String estado,
 String fechaCreacion, String fechaModifica) {
 ID_Cajero = iD_Cajero;
 ID_EntidadTipo = iD_EntidadTipo;
 Usuario = usuario;
 Contrasena = contrasena;
 Estado = estado;
 FechaCreacion = fechaCreacion;
 FechaModifica = fechaModifica;
 }

 public CajeroDTO(){}

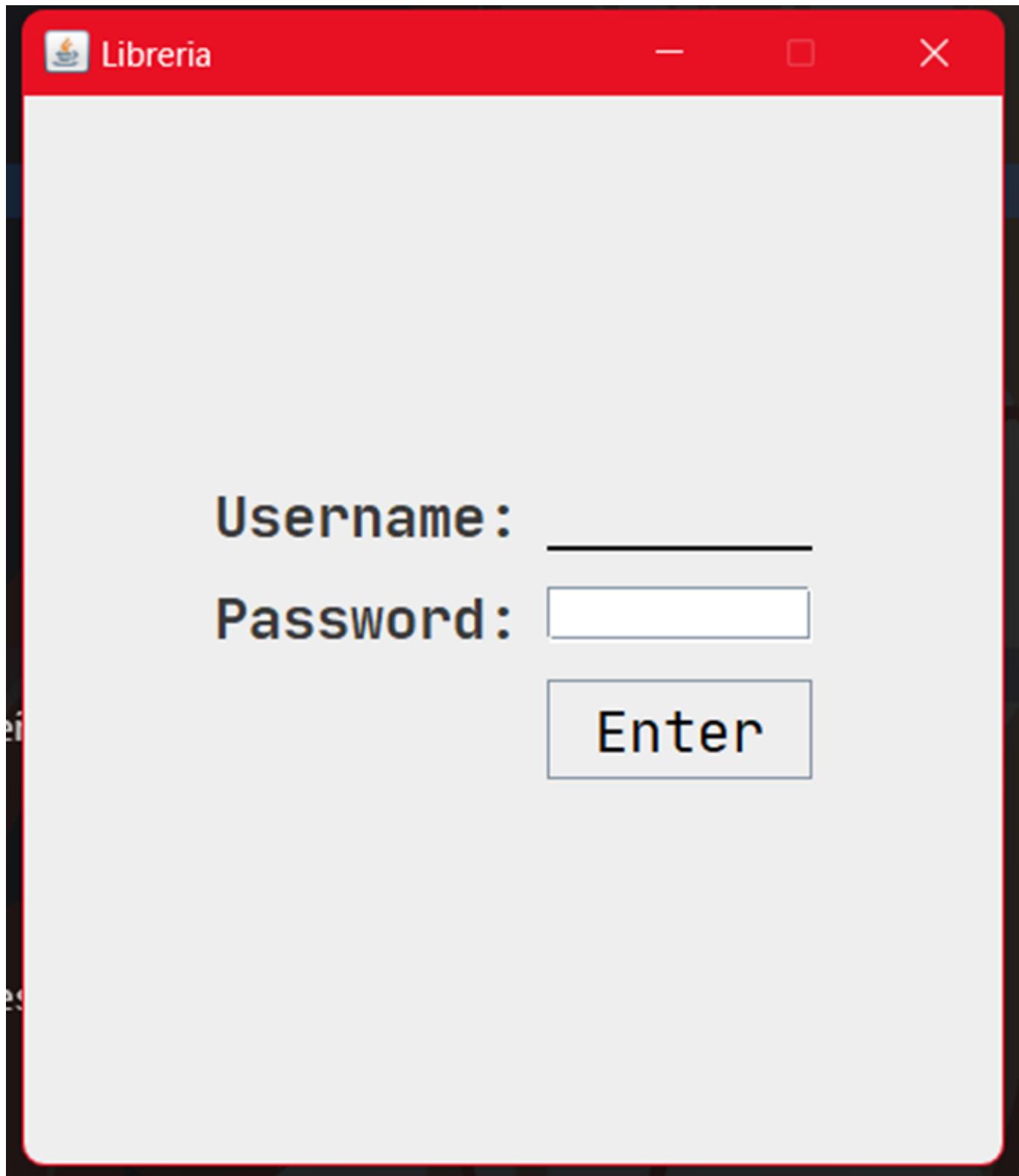
 public Integer getID_Cajero() {
 return ID_Cajero;
 }
 public void setID_Cajero(Integer iD_Cajero) {
 ID_Cajero = iD_Cajero;
 }
 public String getUsuario() {
 return Usuario;
 }
 public void setUsuario(String usuario) {
 Usuario = usuario;
 }
}
```

## Aplicacion

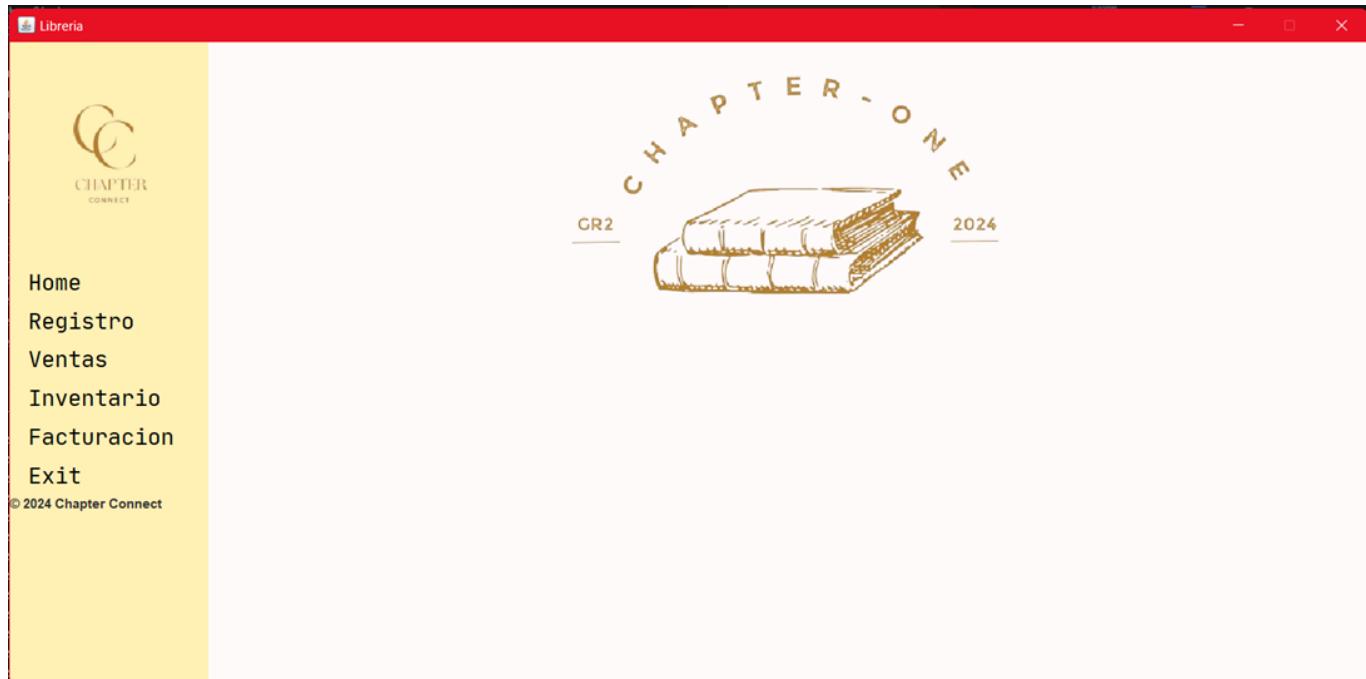
pestaña login



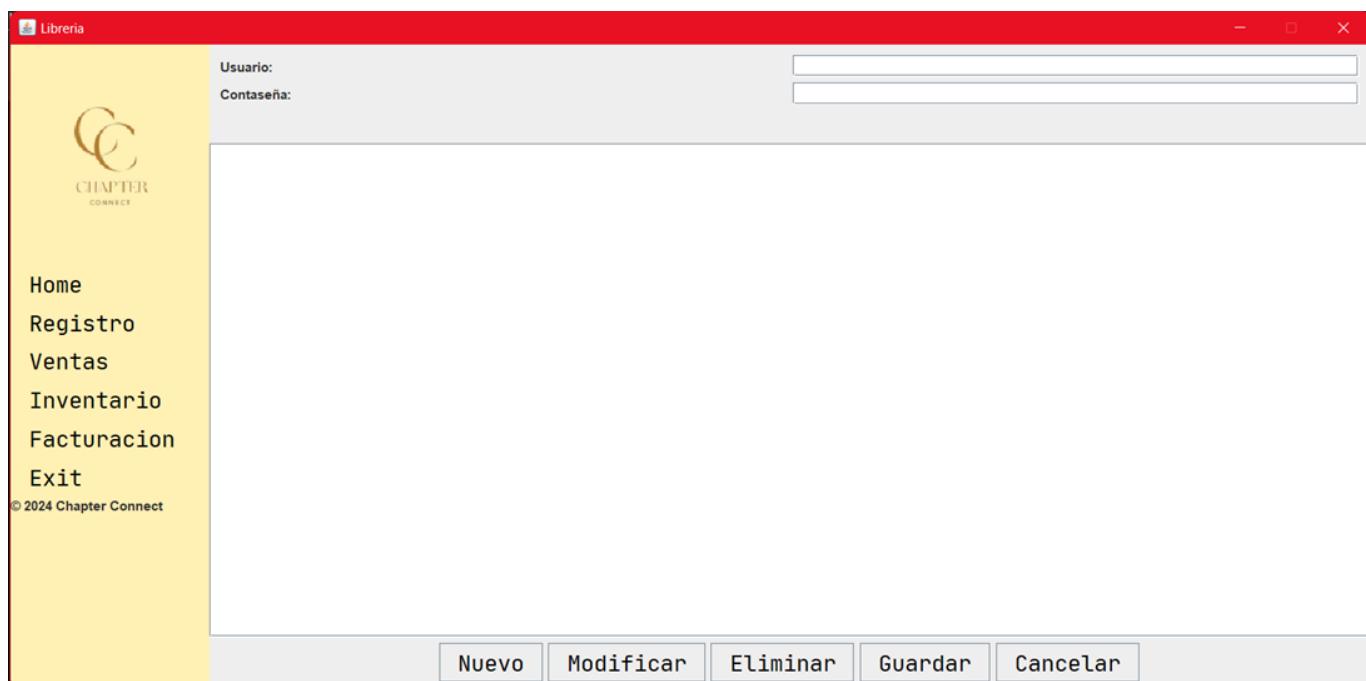
panel login



pantalla principal



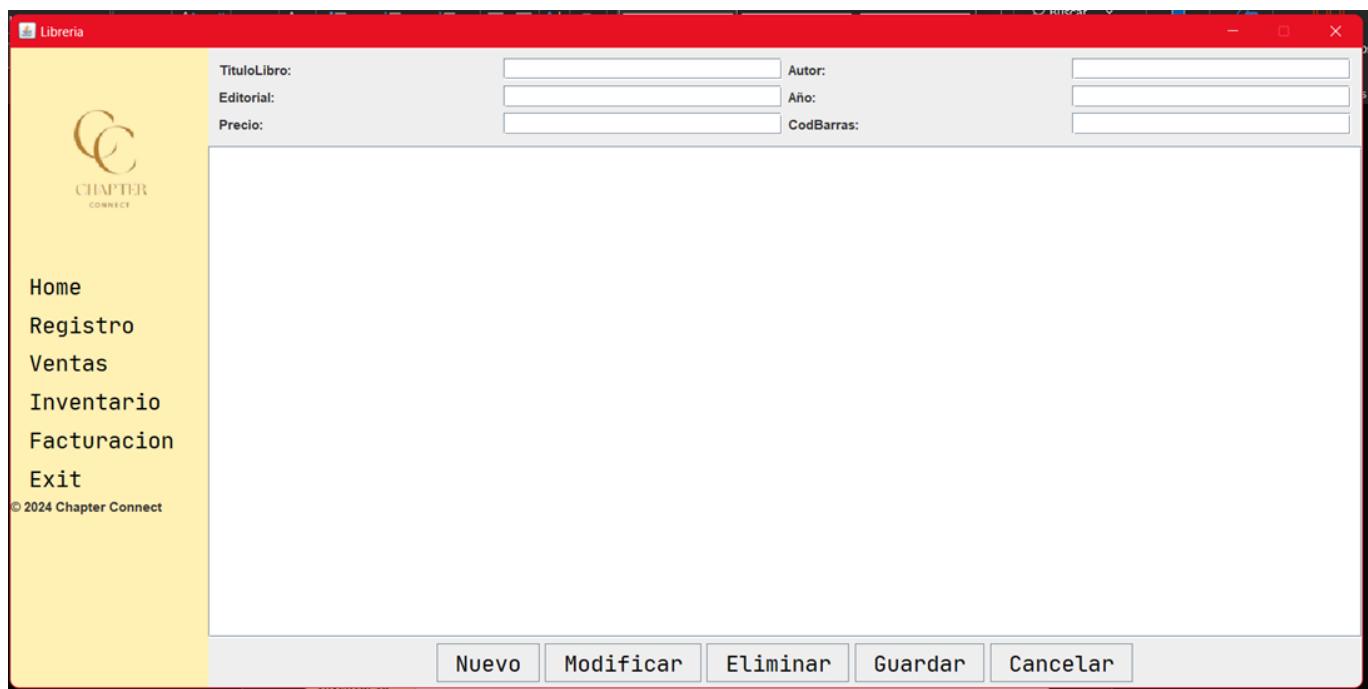
pantalla de registro



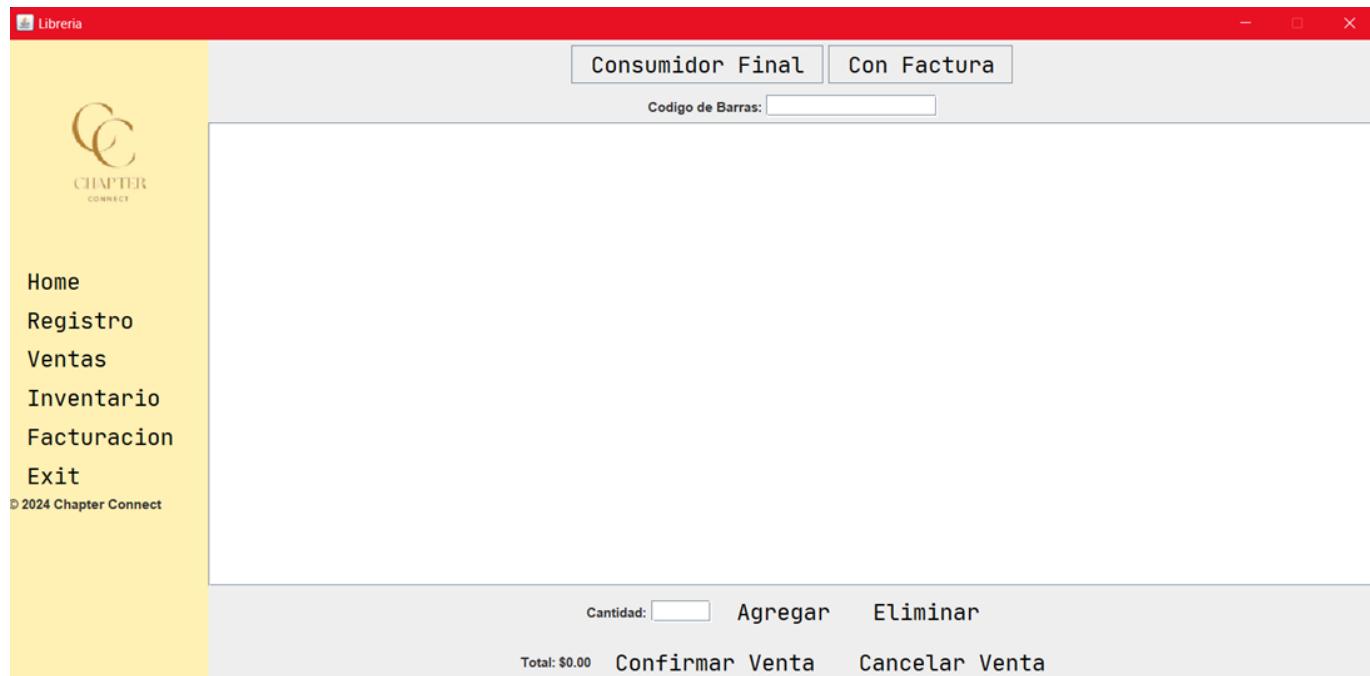
pantalla de ventas



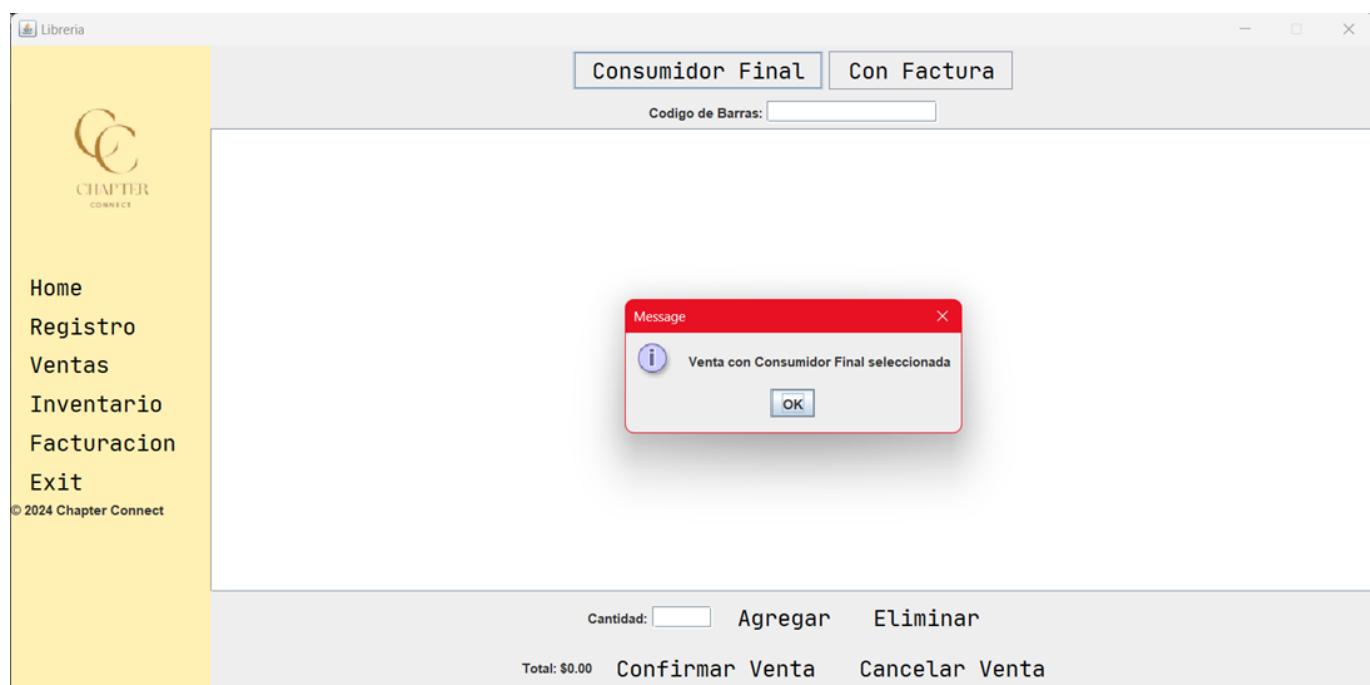
pantalla Inventario



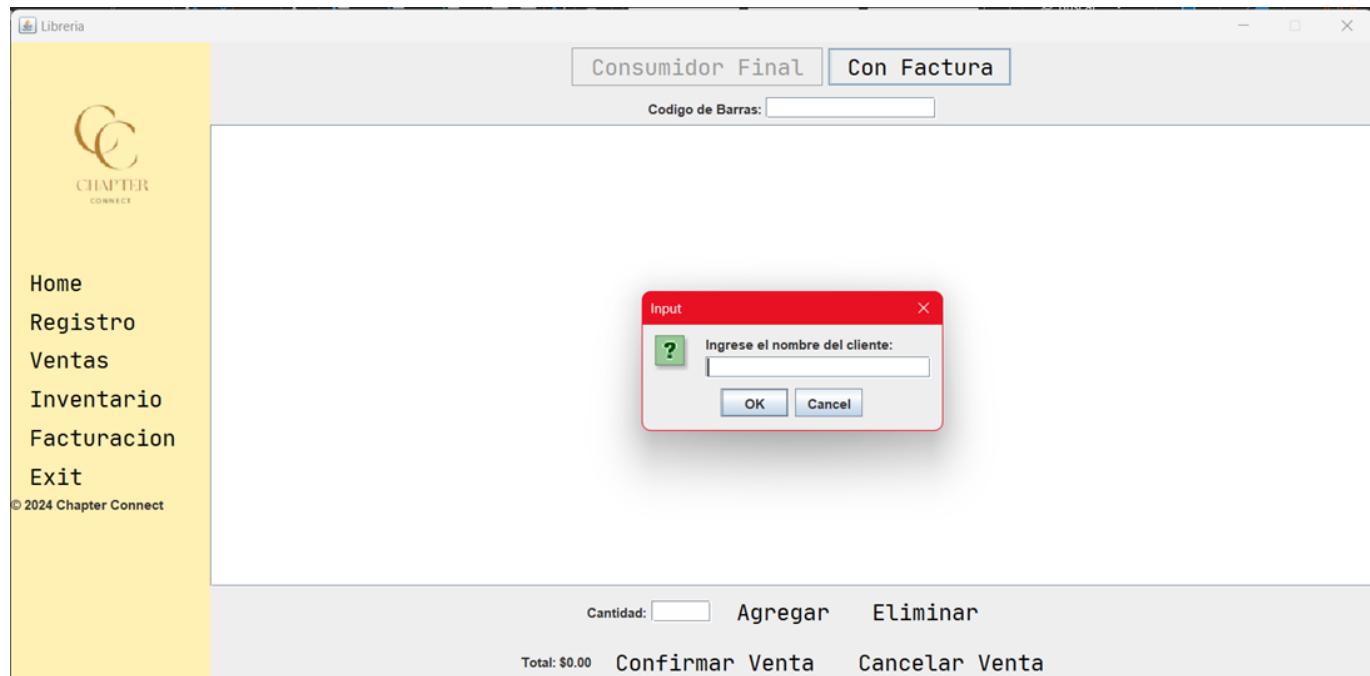
pantalla factura



pantalla consumidor final



pantalla con Factura



## EXAMEN:

*Documento*

**CASO DE ESTUDIO:**

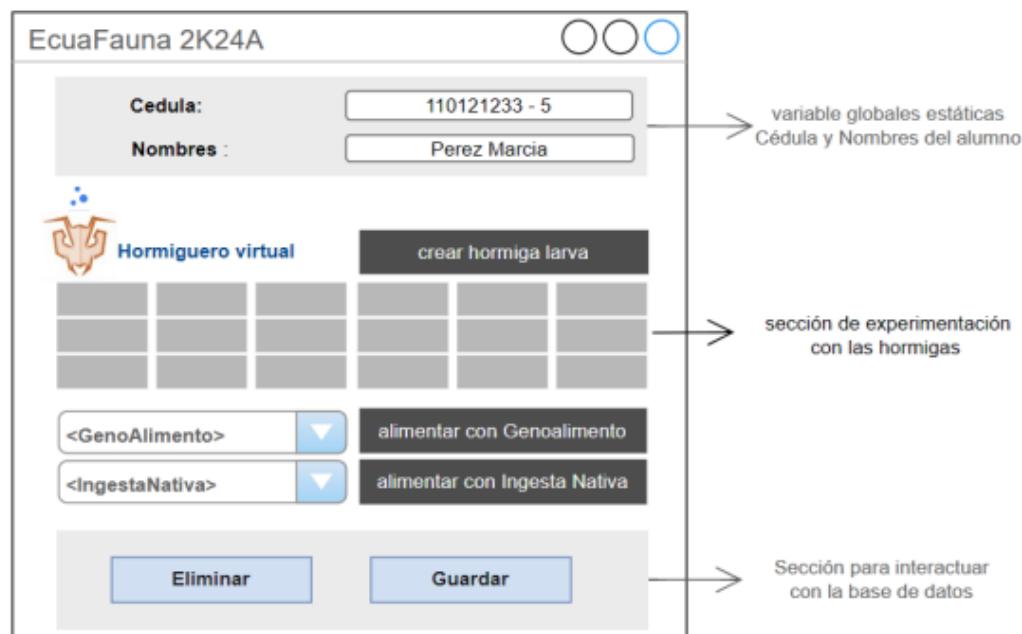
Lea cuidadosamente su caso de estudio. Se requiere crear una aplicación denominada "**EcuaFauna 2K24A**" que permita recrear un hormiguero virtual para que entomólogos expertos en hormigas endémicas del Ecuador puedan experimentar creando hormigas larvas que evolucionan según el tipo de alimentación. El tipo de alimentación se basa en **GenoAlimentos** e **ingesta nativa** del entorno.

El estándar de programación para este Proyecto, codificación (java) y base de datos (SQLite) debe cumplir con el formato camelCase/UppercamelCase para variables globales, locales, paquetes, Clases, atributos, métodos y artefactos. Se debe anteponer el prefijo PrimeraLetraApellidoPrimeraLetraNombre en singular como parte del estándar. Donde:

**Ejemplo, si el alumno es:** Marco Pérez

|           |                                                                           |
|-----------|---------------------------------------------------------------------------|
| Proyecto: | <b>PM</b> EcuaFauna2K24A                                                  |
| Clase:    | <b>PM</b> Hormiga, <b>PM</b> Alimento, ...                                |
| Método:   | <b>pmGetNombre()</b> , ...                                                |
| Variable: | <b>pmNombre</b> , ...                                                     |
| tabla:    | <b>MLugar</b> , los campos de la tabla usar UpperCamelCase sin el prefijo |

- a. La aplicación debe implementar la arquitectura N-LAYER, documentada y subida/versionado en el GitHub cumpliendo con el estándar descrito en el ítem anterior.
  - Agregar un directorio "Design" que contenga los diagramas de: caso de uso, base de datos, clase (mínimo 3 atributos o métodos relevantes), y arquitectura de su solución.
  - Agregar un directorio "DataBase" para la base de datos.
- b. Crear entidades de base de datos para almacenar información de:
  - País [Ecuador], Región [Costa, Sierra, Oriente, Galápagos], Provincia [...]
  - Alimento (IngestaNativa [ Carnívoro, Herbívoro, Omnívoro, Insectívoros], GenoAlimento [ X, XX, XY]).
  - Sexo [Macho, Hembra, Asexual], Hormiga [TipoHormiga, Sexo, Provincia, GenoAlimento, IngestaNativa, Estado, ...]
- c. Mostrar un formulario "**EcuaFauna 2K24A**" con los componentes detallados en el prototipo. Al inicializarse el formulario se debe mostrar la **cédula, nombres** del alumno (sus datos) y los tipos de alimento en selectores(combo) "**GenoAlimento**" e "**IngestaNativa**". La sección del "Hormiguero virtual" muestra los datos de la hormiga.



*Codigo*

Interfaz Grafica

The screenshot shows a Java IDE interface with the following details:

- Editor Area:** Displays the code for `GCPPanelCentral.java`. The code includes methods for initializing a table model, handling cell editability, and loading data from the database.
- Explorer Panel:** Shows the project structure `GCECUFAUNA2K24A` with packages like `src`, `BusinessLogic`, `DAO`, and `DTO`, along with various Java files such as `GCDTOGenoAlimento.java`, `GCDTOHormiga.java`, etc.
- Bottom Bar:** Includes standard IDE navigation buttons like back, forward, search, and file operations, along with links for Java Ready, Share Code Link, Generate Commit Message, Explain Code, Comment Code, Find Bugs, Code Chat, CRLF, Java, AI Code Chat, Background, and Asuna.

## Business Logic

The screenshot shows a Java IDE interface with the following details:

- Editor Area:** Displays the code for `GCBLHormiga.java`. The code implements business logic for managing `GCDTOHormiga` objects, interacting with `GCDAOHormiga` and `GCDAODAO`.
- Explorer Panel:** Shows the project structure `GCECUFAUNA2K24A` with packages like `src`, `BusinessLogic`, `DataAccess`, `DAO`, and `DTO`, along with various Java files such as `GCBLGenoAlimento.java`, `GCBLHormiga.java`, etc.
- Bottom Bar:** Includes standard IDE navigation buttons like back, forward, search, and file operations, along with links for Java Ready, Share Code Link, Generate Commit Message, Explain Code, Comment Code, Find Bugs, Code Chat, CRLF, Java, AI Code Chat, Background, and Asuna.

## Interfaz

```

src > DataAccess > DAO > GCIDAO.java > ...
You, ayer | 1 author (You)
1 package DataAccess.DAO;
2
3 import java.util.List;
4
5 You, ayer | 1 author (You)
6 public interface GCIDAO<T> {
7
8 public boolean create(T entity) throws Exception;
9 public List<T> readAll() throws Exception;
10 public boolean update(T entity) throws Exception;
11 public boolean delete(int id) throws Exception;
12 public T readBy(Integer id) throws Exception;
13 //public Integer getMaxRow() throws Exception;
14 }
15

```

## DAO

```

src > DataAccess > DAO > GCDAOHormiga.java > GCDAOHormiga
15 import Framework.GCException;
16
17 You, hace 9 horas | 1 author (You)
18 public class GCDAOHormiga extends GCDATAHelper implements GCIDAO<GCDTOHormiga> {
19 You, ayer + database
20 @Override
21 public boolean create(GCDTOHormiga entity) throws Exception {
22 DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
23 LocalDateTime now = LocalDateTime.now();
24 String query = "INSERT INTO GCHORMIGA (IdClgTipoHormiga, IdClgIngestaNativa, IdClgGenoAlimento, IdClgSexo, IdUbicacion, Fecha, Hora) VALUES (?, ?, ?, ?, ?, ?, ?)";
25 try {
26 Connection conn = openConnection();
27 PreparedStatement pstmt = conn.prepareStatement(query);
28 pstmt.setInt(parameterIndex:1, entity.getIdClgTipoHormiga());
29 pstmt.setInt(parameterIndex:2, entity.getIdClgIngestaNativa());
30 pstmt.setInt(parameterIndex:3, entity.getIdClgGenoAlimento());
31 pstmt.setInt(parameterIndex:4, entity.getIdClgSexo());
32 pstmt.setInt(parameterIndex:5, entity.getIdClgUbicacion());
33 pstmt.setString(parameterIndex:6, dtf.format(now.toString()));
34
35 pstmt.executeUpdate();
36 return true;
37 } catch (SQLException e) {
38 throw new GCException(e.getMessage(), getClass().getName(), metodo:"create()");
39 }
40 }
41
42
43 public List<GCDTOHormiga> readAll() throws Exception {
44 List<GCDTOHormiga> lst = new ArrayList<>();
45 String query = "SELECT IdHormiga
46 ,IdClgTipohormiga
47 ,IdClgIngestaNativa
48 ,IdClgGenoAlimento
49 ,IdClgSexo
50 ,IdUbicacion
51 ";
52
53 try {
54 Connection conn = openConnection();
55 PreparedStatement pstmt = conn.prepareStatement(query);
56 ResultSet rs = pstmt.executeQuery();
57 while (rs.next()) {
58 GCDTOHormiga hormiga = new GCDTOHormiga();
59 hormiga.setIdHormiga(rs.getInt("IdHormiga"));
60 hormiga.setIdClgTipohormiga(rs.getInt("IdClgTipohormiga"));
61 hormiga.setIdClgIngestaNativa(rs.getInt("IdClgIngestaNativa"));
62 hormiga.setIdClgGenoAlimento(rs.getInt("IdClgGenoAlimento"));
63 hormiga.setIdClgSexo(rs.getInt("IdClgSexo"));
64 hormiga.setIdClgUbicacion(rs.getInt("IdUbicacion"));
65 lst.add(hormiga);
66 }
67 } catch (SQLException e) {
68 throw new GCException(e.getMessage(), getClass().getName(), metodo:"readAll()");
69 }
70 }
71
72 }

```

## DTO

The screenshot shows the Visual Studio Code interface. The left side displays the code for `GCDTOHormiga.java`, which defines a class with various fields and methods related to ants. The right side shows the project structure in the "EXPLORADOR" (Explorer) sidebar, under the folder `GCECUFAUNA2K24A`. The `lib` folder contains the file `GCDTOHormiga.java`, which is highlighted in blue. Other files in the `lib` folder include `GCDTOGenoAlimento.java`, `GCDTOIngestaNativa.java`, `GCDTOSexo.java`, `GCDTOTipoHormiga.java`, and `GCDTOUbicacion.java`. The `src` folder contains subfolders like `BusinessLogic`, `DAO`, and `DTO`, each containing several Java files. The status bar at the bottom shows various icons and text, including "Java Ready" and "Code Chat".

```

public class GCDTOHormiga {
 private Integer gcIdHormiga;
 private Integer gcIdClgSexo;
 private Integer gcIdUbicacion;
 private String gcEstado;
 private String gcFechaCreacion;
 private String gcFechaModifica;

 public GCDTOHormiga(){}
 public GCDTOHormiga(Integer gcIdHormiga, Integer gcIdClgTipoHormiga, Integer gcIdClgIngestaNativa,
 Integer gcIdClgGenoAlimento, Integer gcIdClgSexo, Integer gcIdUbicacion, String gcEstado,
 String gcFechaCreacion, String gcFechaModifica) {
 this.gcIdHormiga = gcIdHormiga;
 this.gcIdClgTipoHormiga = gcIdClgTipoHormiga;
 this.gcIdClgIngestaNativa = gcIdClgIngestaNativa;
 this.gcIdClgGenoAlimento = gcIdClgGenoAlimento;
 this.gcIdClgSexo = gcIdClgSexo;
 this.gcIdUbicacion = gcIdUbicacion;
 this.gcEstado = gcEstado;
 this.gcFechaCreacion = gcFechaCreacion;
 this.gcFechaModifica = gcFechaModifica;
 }
 public Integer getgcIdHormiga() {
 return gcIdHormiga;
 }
 public void setgcIdHormiga(Integer gcIdHormiga) {
 this.gcIdHormiga = gcIdHormiga;
 }
 public Integer getgcIdClgTipoHormiga() {
 return gcIdClgTipoHormiga;
 }
 public void setgcIdClgTipoHormiga(Integer gcIdClgTipoHormiga) {
 this.gcIdClgTipoHormiga = gcIdClgTipoHormiga;
 }
 public Integer getgcIdClgIngestaNativa() {
 return gcIdClgIngestaNativa;
 }
 public void setgcIdClgIngestaNativa(Integer gcIdClgIngestaNativa) {
 }
}

```

## DataHelper

This screenshot shows the Visual Studio Code interface again. The left side displays the code for `GDDataHelper.java`, which is part of the `DataAccess` package. It contains methods for opening and closing database connections. The right side shows the project structure in the "EXPLORADOR" sidebar, under the folder `GCECUFAUNA2K24A`. The `lib` folder contains the file `GDDataHelper.java`, which is highlighted in blue. Other files in the `lib` folder include `GCDTOGenoAlimento.java`, `GCDTOHormiga.java`, `GCDTOIngestaNativa.java`, `GCDTOSexo.java`, `GCDTOTipoHormiga.java`, and `GCDTOUbicacion.java`. The `src` folder contains subfolders like `BusinessLogic`, `DAO`, and `DTO`, each containing several Java files. The status bar at the bottom shows various icons and text, including "Java Ready" and "Code Chat".

```

package DataAccess;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

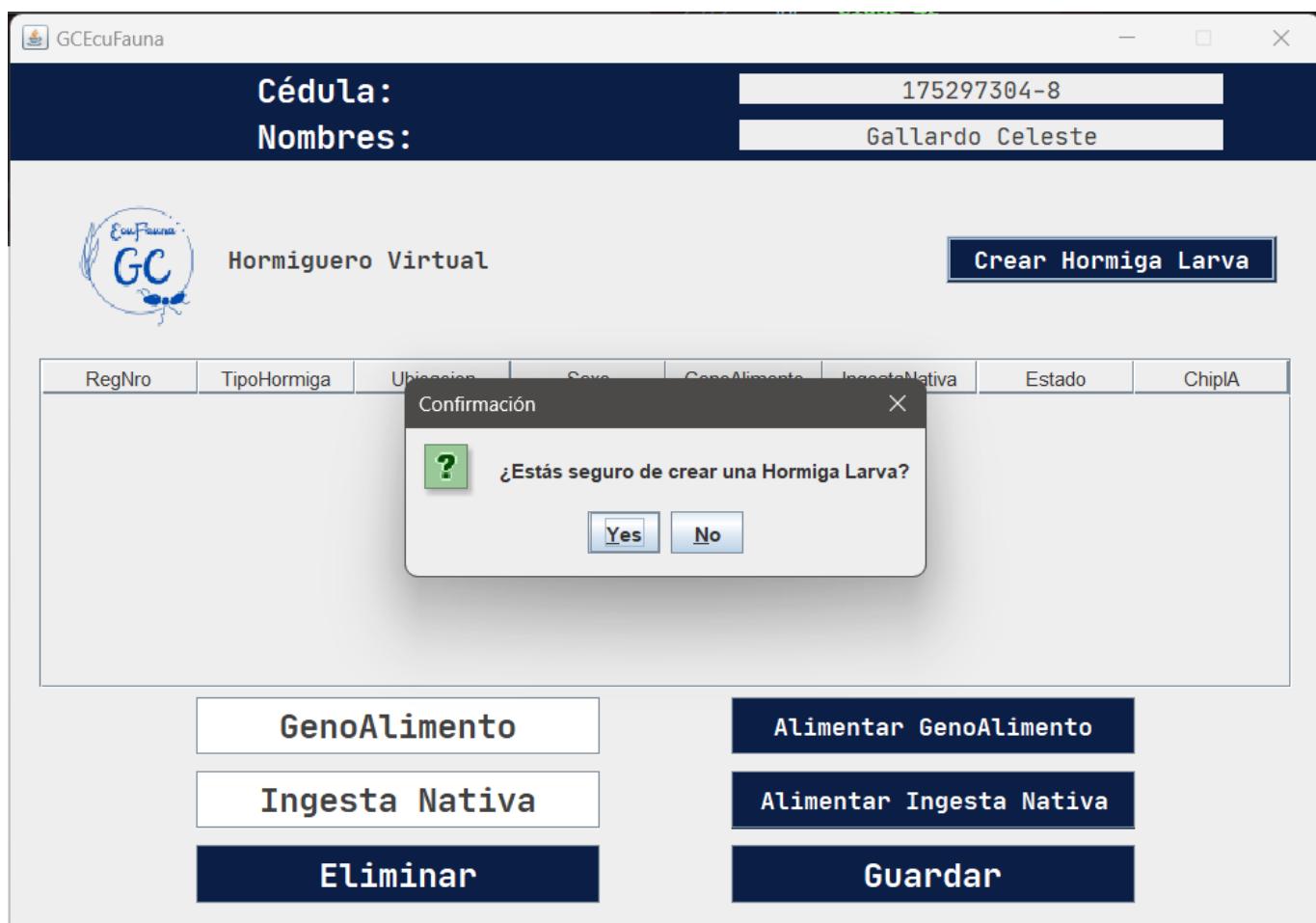
```

## Aplicacion

### Pantalla de inicio



Boton de confirmacion crear



panel creado

The screenshot shows a window titled "GC EcuFauna". At the top, there are input fields for "Cédula:" (with value "175297304-8") and "Nombres:" (with value "Gallardo Celeste"). Below the header, there is a logo for "EcuFauna GC" and the text "Hormiguero Virtual". On the right, a button says "Crear Hormiga Larva". A table below lists an ant colony entry:

| RegNro | TipoHormiga | Ubicacion | Sexo    | GenoAlimento | IngestaNativa | Estado | ChiplA            |
|--------|-------------|-----------|---------|--------------|---------------|--------|-------------------|
| 1      | Larva       | Galápagos | Asexual |              |               | VIVA   | Aprendiendo Es... |

At the bottom, there are several buttons: "GenoAlimento", "Ingesta Nativa", "Eliminar" (highlighted in dark blue), "Alimentar GenoAlimento", "Alimentar Ingesta Nativa", and "Guardar".

Boton desplegable 1

GCEcuFauna

**Cédula:** 175297304-8  
**Nombres:** Gallardo Celeste

 Hormiguero Virtual Crear Hormiga Larva

| RegNro | TipoHormiga | Ubicacion | Sexo    | GenoAlimento | IngestaNativa | Estado | ChiplA            |
|--------|-------------|-----------|---------|--------------|---------------|--------|-------------------|
| 1      | Larva       | Galápagos | Asexual |              |               | VIVA   | Aprendiendo Es... |

**GenoAlimento**

X  
xx  
XY

**Ingesta Nativa**

**Eliminar**

**Alimentar GenoAlimento**

**Alimentar Ingesta Nativa**

**Guardar**

agregar reina

GCEcuFauna

**Cédula:** 175297304-8  
**Nombres:** Gallardo Celeste

 Hormiguero Virtual Crear Hormiga Larva

| RegNro | TipoHormiga | Ubicacion | Sexo     | GenoAlimento | IngestaNativa | Estado | ChiplA        |
|--------|-------------|-----------|----------|--------------|---------------|--------|---------------|
| 1      | Reina       | Galápagos | Femenino | XX           |               | VIVA   | Habla Español |

**XX**

**Ingesta Nativa**

**Eliminar**

**Alimentar GenoAlimento**

**Alimentar Ingesta Nativa**

**Guardar**

agregar comida boton desplegable 2

The screenshot shows a software application window titled "GCEcuFauna". At the top, there are two input fields: "Cédula:" containing "175297304-8" and "Nombres:" containing "Gallardo Celeste". Below this, a logo for "GCEcuFauna" is displayed next to the text "Hormiguero Virtual". On the right, a button labeled "Crear Hormiga Larva" is visible. A table below lists ant details: RegNro (1), TipoHormiga (Reina), Ubicacion (Galápagos), Sexo (Femenino), GenoAlimento (XX), IngestaNativa (VIVA), Estado (VIVA), and ChipIA (Habla Español). In the center, there are several interactive buttons: "XX", "Ingesta Nativa" (highlighted in blue), "Liminar" (highlighted in dark blue), "Alimentar GenoAlimento", "Alimentar Ingesta Nativa", and "Guardar". A sidebar on the left lists feeding categories: Carnívoro, Herbívoro, Omnívoro, Insectívoro, and Nectarívoro.

| RegNro | TipoHormiga | Ubicacion | Sexo     | GenoAlimento | IngestaNativa | Estado | ChipIA        |
|--------|-------------|-----------|----------|--------------|---------------|--------|---------------|
| 1      | Reina       | Galápagos | Femenino | XX           | VIVA          | VIVA   | Habla Español |

cuadro vive

GCEcuFauna

**Cédula:** 175297304-8  
**Nombres:** Gallardo Celeste

 Hormiguero Virtual Crear Hormiga Larva

| RegNro | TipoHormiga | Ubicacion | Sexo     | GenoAlimento | IngestaNativa | Estado | ChipIA        |
|--------|-------------|-----------|----------|--------------|---------------|--------|---------------|
| 1      | Reina       | Galápagos | Femenino | XX           | Insectívoro   | VIVA   | Habla Español |

XX Alimentar GenoAlimento  
Insectívoro Alimentar Ingesta Nativa  
Eliminar Guardar

cuadro muere

GCEcuFauna

**Cédula:** 175297304-8  
**Nombres:** Gallardo Celeste

 Hormiguero Virtual Crear Hormiga Larva

| RegNro | TipoHormiga | Ubicacion | Sexo     | GenoAlimento | IngestaNativa | Estado | ChipIA        |
|--------|-------------|-----------|----------|--------------|---------------|--------|---------------|
| 1      | Reina       | Galápagos | Femenino | XX           | Herbívoro     | MUERTA | Habla Español |

XX Alimentar GenoAlimento  
Herbívoro Alimentar Ingesta Nativa  
Eliminar Guardar

boton guardar

The screenshot shows a software window titled "Hormiguero Virtual". At the top, there are fields for "Cédula:" (175297304-8) and "Nombres:" (Gallardo Celeste). On the left, there's a logo for "GC EcuFauna". On the right, there's a button labeled "Crear Hormiga Larva". Below these, a table lists 5 entries for nests (RegNro 1-5), each with columns for Type, Location, Sex, Genotype, Diet, Status, and Language. A modal dialog box titled "Confirmación" asks "¿Estás seguro de guardar el hormiguero en la base de datos?" with "Yes" and "No" buttons. In the background, buttons for "Insectívoro", "Alimentar Ingesta Nativa", "Eliminar", and "Guardar" are visible.

| RegNro | TipoHormiga | Ubicacion  | Sexo     | GenoAlimento | IngestaNativa | Estado | ChipLA        |
|--------|-------------|------------|----------|--------------|---------------|--------|---------------|
| 1      | Reina       | Galápagos  | Femenino | XX           | Herbívoro     | MUERTA | Habla Español |
| 2      | Reina       | Bolívar    | Femenino | XX           | Insectívoro   | VIVA   | Habla Español |
| 3      | Reina       | Guayas     | Femenino | XX           | Omnívoro      | MUERTA | Habla Español |
| 4      | Reina       | Guayas     | Femenino | XX           | Carnívoro     | MUERTA | Habla Español |
| 5      | Reina       | Esméraldas | Femenino | XX           | Insectívoro   | VIVA   | Habla Español |

boton eliminar

GCEcuFauna

**Cédula:** 175297304-8  
**Nombres:** Gallardo Celeste

 Hormiguero Virtual      Crear Hormiga Larva

| RegNro | TipoHormiga | Ubicacion  | Sexo     | GenoAlimento | IngestaNativa | Estado | ChiplA        |
|--------|-------------|------------|----------|--------------|---------------|--------|---------------|
| 1      | Reina       | Galápagos  | Femenino | XX           | Herbívoro     | MUERTA | Habla Español |
| 2      | Reina       | Bolívar    | Femenino | XX           | Insectívoro   | VIVA   | Habla Español |
| 3      | Reina       | Guayas     | Femenino | XX           | Omnívoro      | MUERTA | Habla Español |
| 4      | Reina       | Guayas     | Femenino | XX           | Carnívoro     | MUERTA | Habla Español |
| 5      | Reina       | Esmeraldas | Femenino | XX           | Insectívoro   | VIVA   | Habla Español |

Confirmación  
¿Estás seguro de exterminar a la Hormiga?  
Yes    No

XX GenoAlimento

Insectívoro    Alimentar Ingesta Nativa

Eliminar    Guardar

**FIN DEL SEMESTRE**