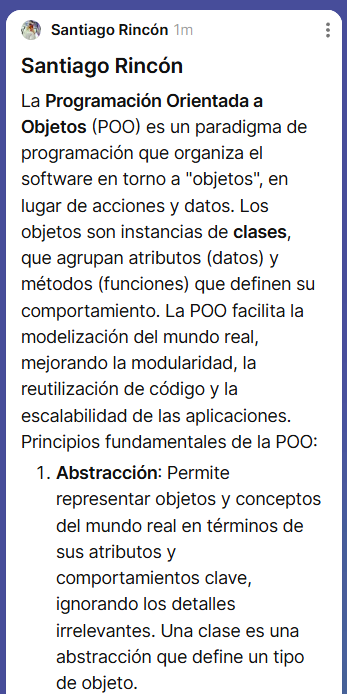
**GUIA POO  
 CRISTIAN SANTIAGO RINCÓN CHAPARRO - 2827167**

**Evidencia 1**

Interfaz de usuario gráfica, Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente

**Evidencia 2 : Conceptos POO**

**¿Qué es la Programación Orientada a Objetos (POO)?** La POO es un paradigma de programación que utiliza "objetos" para representar datos y métodos para manipular esos datos. Este enfoque organiza el código en torno a objetos que representan entidades del mundo real, facilitando la reutilización y el mantenimiento del código.

**¿Cuál es la historia de la Programación Orientada a Objetos?** La POO comenzó a desarrollarse en la década de 1960. El concepto inicial se presentó en el lenguaje Simula, diseñado por Ole-Johan Dahl y Kristen Nygaard en Noruega. Posteriormente, el enfoque se popularizó con el lenguaje Smalltalk en la década de 1970 y, más tarde, con lenguajes como C++ y Java en los años 80 y 90.

**¿Cuáles son los lenguajes de programación que soportan la POO?** Algunos lenguajes de programación que soportan la POO incluyen Java, C++, Python, C#, Ruby, y Objective-C.

**¿Qué es un objeto?** Un objeto es una instancia de una clase que encapsula datos y métodos que operan sobre esos datos. Los objetos tienen atributos (propiedades) y comportamientos (métodos).

**¿Qué es un atributo?** Un atributo es una variable de una clase que representa una característica o propiedad del objeto. Por ejemplo, en una clase Coche, los atributos podrían ser color y modelo.

**¿Qué es un método?** Un método es una función definida dentro de una clase que describe el comportamiento de los objetos de esa clase. Los métodos pueden manipular los atributos del objeto y realizar otras acciones.

**¿Qué es un método void?** Un método void es un método que no devuelve ningún valor. En lugar de retornar un valor, simplemente realiza una acción o modifica el estado del objeto.

**¿Cuáles son los tipos de método?** Los tipos de métodos incluyen métodos de instancia (que operan sobre instancias específicas de una clase), métodos de clase (que operan a nivel de clase y se definen con static), y métodos abstractos (declarados en clases abstractas sin implementación).

**¿Qué es una clase, describa sus características?** Una clase es una plantilla o modelo para crear objetos. Define un conjunto de atributos y métodos que los objetos de esa clase tendrán. Las características de una clase incluyen:

* **Encapsulamiento:** Agrupa datos y métodos en una sola unidad.
* **Constructores:** Métodos especiales para inicializar objetos.
* **Herencia:** Permite que una clase derive de otra.
* **Modificadores de acceso:** Controlan la visibilidad de los atributos y métodos.

**¿Para qué sirve Getter y Setter?** Los métodos **Getter** y Setter se utilizan para acceder y modificar los atributos privados de una clase. Un **Getter** devuelve el valor de un atributo, mientras que un Setter permite modificar el valor del atributo.

**¿Para qué sirve this?** La palabra clave **this** se refiere al objeto actual en el que se está ejecutando el método. Se utiliza para acceder a los atributos y métodos del objeto actual y para diferenciar entre atributos de instancia y parámetros de método.

**¿Para qué sirve null?** **null** representa la ausencia de un valor o una referencia en programación. Se utiliza para indicar que una variable no apunta a ningún objeto.

**¿Qué es instanciar?** Instanciar significa crear un nuevo objeto a partir de una clase. Al instanciar una clase, se crea una instancia de esa clase con sus propios atributos y métodos.

**¿Cuáles son los pilares de la POO?** Los pilares de la POO son:

* **Abstracción:** Representar entidades del mundo real mediante clases y objetos.
* **Encapsulamiento:** Ocultar la complejidad interna y exponer solo lo necesario.
* **Herencia:** Permitir que una clase derive de otra.
* **Polimorfismo:** Permitir que diferentes clases sean tratadas como instancias de una misma clase a través de una interfaz común.

**¿Qué es abstracción?** La abstracción es el proceso de simplificar la representación de un objeto al enfocarse en sus características esenciales y ocultar los detalles no relevantes.

**¿Qué es polimorfismo?** El polimorfismo permite que un objeto sea tratado como una instancia de su clase base. Existen dos tipos: polimorfismo de método (múltiples métodos con el mismo nombre pero diferente implementación) y polimorfismo de sobrecarga (mismo nombre de método con diferentes parámetros).

**¿Qué es encapsulamiento?** El encapsulamiento es el principio de ocultar los detalles internos de una clase y exponer solo una interfaz pública para interactuar con ella. Esto ayuda a proteger el estado interno del objeto y controlar el acceso a sus datos.

**¿Cuáles son los tipos de modificadores de acceso en JAVA?** Los modificadores de acceso en Java son:

* **public:** Accesible desde cualquier clase.
* **protected:** Accesible dentro del mismo paquete y en subclases.
* **default (sin modificador):** Accesible solo dentro del mismo paquete.
* **private:** Accesible solo dentro de la clase en la que está definido.

**¿Qué es herencia?** La herencia es un mecanismo que permite que una clase derive de otra, heredando sus atributos y métodos. La clase derivada (subclase) puede extender o modificar el comportamiento de la clase base (superclase).

**¿Qué es un constructor?** Un constructor es un método especial que se ejecuta automáticamente cuando se crea un objeto. Se utiliza para inicializar el objeto con valores predeterminados o proporcionados.

**¿Qué es un destructor?** Un destructor es un método especial que se ejecuta cuando un objeto es destruido o eliminado. En Java, los destructores se manejan automáticamente con el recolector de basura, por lo que no se requiere definir destructores explícitos.

**¿En qué consiste el concepto de memoria stack y heap?**

* **Stack:** La memoria stack se utiliza para almacenar variables locales y llamadas de método. La memoria se gestiona de forma automática y se libera cuando la función o el método termina.
* **Heap:** La memoria heap se utiliza para almacenar objetos y datos que necesitan una vida útil más larga. La memoria se gestiona manualmente mediante el recolector de basura, que libera la memoria no utilizada.

**¿Qué es una clase anidada?** Una clase anidada es una clase definida dentro de otra clase. Las clases anidadas pueden ser estáticas (clases anidadas estáticas) o no estáticas (clases internas).

**¿Qué es una clase abstracta?** Una clase abstracta es una clase que no se puede instanciar directamente. Está diseñada para ser extendida por otras clases y puede contener métodos abstractos (sin implementación) que las clases derivadas deben implementar.

**¿Qué es un mensaje en POO?** En POO, un mensaje es una solicitud enviada a un objeto para que ejecute uno de sus métodos. Los mensajes permiten la comunicación entre objetos.

**¿En qué consiste la sobrecarga de constructores y métodos?** La sobrecarga de constructores y métodos permite definir múltiples versiones de un constructor o método con el mismo nombre pero con diferentes parámetros. Esto permite adaptar el comportamiento a diferentes contextos.

**¿En qué consiste el uso de la palabra final?** La palabra clave **final** se utiliza para:

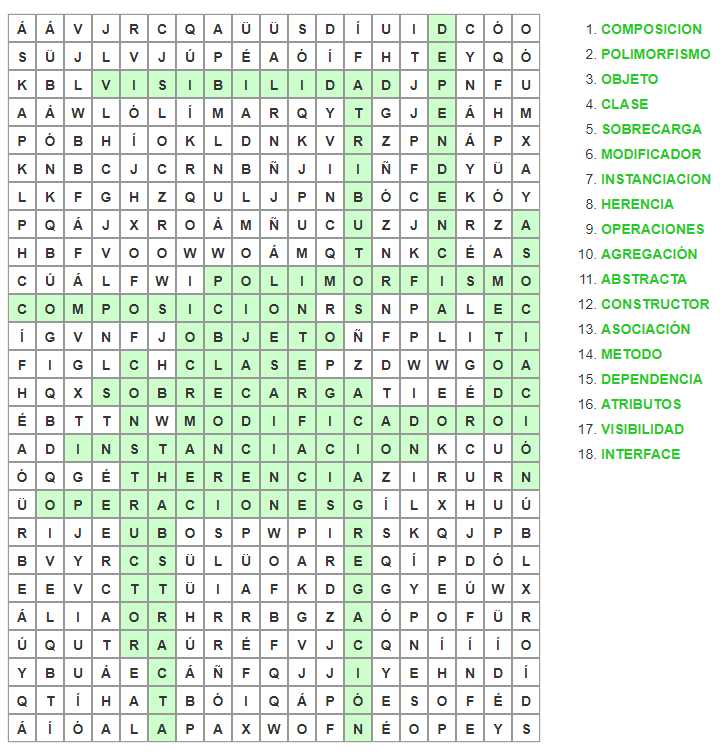
* **Variables:** Indicar que el valor de la variable no puede cambiar después de la inicialización.
* **Métodos:** Impedir que un método sea sobreescrito por una subclase.
* **Clases:** Impedir que una clase sea extendida.

**¿Qué es una colección?** Una colección es una estructura de datos que agrupa múltiples elementos en una sola unidad. Las colecciones permiten almacenar, recuperar y manipular datos de manera eficiente.

**¿Qué es ArrayList?** ArrayList es una clase en Java que implementa una colección dinámica de elementos. Ofrece métodos para agregar, eliminar y acceder a los elementos, y ajusta su tamaño automáticamente.

**¿Qué es una interfaz?** Una interfaz es un tipo de referencia en Java que define un conjunto de métodos abstractos que otras clases deben implementar. Las interfaces permiten definir comportamientos comunes sin especificar cómo se implementan.

**¿Qué es una excepción?** Una excepción es un evento que interrumpe el flujo normal de ejecución de un programa. Las excepciones se utilizan para manejar errores y situaciones excepcionales de manera controlada.

**SOPA DE LETRAS-EVIDENCIA3**  


### 

### **¿Cuál es la diferencia entre: Public, Private, Protected y no tener modificador de acceso en una variable?**

* **Public:** Cuando una variable es pública, puede ser accesible desde cualquier parte del programa, es decir, desde cualquier clase o paquete.
* **Private:** Una variable privada solo puede ser accedida dentro de la clase donde está definida. No puede ser vista ni modificada desde otras clases.
* **Protected:** Las variables protegidas son accesibles dentro de la misma clase, las subclases (herencia), y también dentro del mismo paquete.
* **Sin modificador (default):** Si una variable no tiene ningún modificador de acceso, se aplica el "acceso por defecto" o "paquete", lo que significa que solo puede ser accesible desde clases dentro del mismo paquete.

### **¿Cuál es la diferencia entre métodos void y tipo?**

* **Métodos void:** Son aquellos que no devuelven ningún valor. Solo ejecutan una serie de instrucciones pero no tienen un valor de retorno. Ejemplo: un método que imprime un mensaje en la pantalla.
* **Métodos con tipo:** Son aquellos que devuelven un valor de algún tipo específico (como int, String, boolean, etc.). Estos métodos deben incluir una sentencia return para devolver el valor correspondiente.

### 

### **Ejemplo de una clase en POO señalando cada uno de los términos definidos anteriormente:**

// Abstracción: Definimos la clase abstracta Animal para representar un animal genérico.

abstract class Animal {

protected String name; // Encapsulamiento: Protegemos el acceso a los atributos usando protected.

// Constructor: Inicializa el nombre del animal.

public Animal(String name) {

this.name = name;

}

// Método abstracto: Obliga a las subclases a definir cómo "hablan".

public abstract void makeSound();

// Método común para todas las subclases.

public void eat() {

System.out.println(name + " está comiendo.");

}

}

// Herencia: La clase Dog hereda de Animal.

class Dog extends Animal {

public Dog(String name) {

super(name);

}

// Polimorfismo: Implementación específica del método abstracto makeSound.

@Override

public void makeSound() {

System.out.println(name + " dice: ¡Guau, guau!");

}

}

// Herencia: La clase Cat hereda de Animal.

class Cat extends Animal {

public Cat(String name) {

super(name);

}

// Polimorfismo: Implementación específica del método abstracto makeSound.

@Override

public void makeSound() {

System.out.println(name + " dice: ¡Miau, miau!");

}

}

// Clase principal para ejecutar el programa.

public class Main {

public static void main(String[] args) {

// Creamos objetos Dog y Cat.

Animal dog = new Dog("Fido");

Animal cat = new Cat("Michi");

// Abstracción: Interactuamos con los objetos sin preocuparnos de los detalles internos.

dog.makeSound(); // Polimorfismo en acción

dog.eat(); // Comportamiento común

cat.makeSound(); // Polimorfismo en acción

cat.eat(); // Comportamiento común

}

}

### **¿Qué es un método?**

Un método es una función dentro de una clase que realiza una tarea específica o conjunto de acciones. Los métodos permiten que los objetos realicen acciones o comportamientos definidos por el programador, como acceder o modificar atributos o interactuar con otros objetos.

### **¿Qué estructura debe tener un método?**

Un método tiene la siguiente estructura básica:

* **Modificador de acceso** (public, private, etc.)
* **Tipo de retorno** (void si no devuelve nada o un tipo de dato como int, String, etc.)
* **Nombre del método**
* **Parámetros** (opcional, si se requieren para el método)
* **Cuerpo del método**, que incluye las instrucciones o el código que debe ejecutar el método.

EJEMPLO:

// Definimos una clase llamada Calculadora

public class Calculadora {

// Método: suma dos números enteros y devuelve el resultado

public int sumar(int a, int b) {

// Cuerpo del método: suma de los dos números

int resultado = a + b;

return resultado; // Retorna el resultado de la suma

}

// Método principal para ejecutar el programa

public static void main(String[] args) {

// Crear un objeto de la clase Calculadora

Calculadora calc = new Calculadora();

// Llamar al método 'sumar' y mostrar el resultado

int suma = calc.sumar(5, 3);

System.out.println("La suma es: " + suma); // Imprime: La suma es: 8

### }

### **¿Qué significa cuando se menciona que un método retorna o no retorna un valor?**

* **Retorna un valor:** Significa que después de ejecutar las instrucciones dentro del método, se devuelve un valor (como un número, un texto, etc.) al lugar donde se invocó el método. El tipo de retorno debe ser especificado en la declaración del método.
* **No retorna un valor:** Si el método no devuelve nada, se declara como void. En este caso, el método solo realiza una acción pero no regresa ningún valor.

### **¿Qué son y para qué sirven los parámetros de un método? ¿Para qué se utiliza la palabra reservada return?**

* **Parámetros:** Son variables que se pasan a un método cuando se llama o invoca. Sirven para proporcionar datos que el método necesita para realizar su tarea. Por ejemplo, en el método sumar(int a, int b), a y b son parámetros que se sumarán dentro del método.
* **return:** La palabra reservada return se utiliza para devolver un valor desde un método con tipo de retorno. Cuando se invoca un método que tiene un valor de retorno, return es la forma en que se envía ese valor al código que llamó al método.

### **¿A qué se refiere cuando se menciona llamar o invocar un método?**

Llamar o invocar un método significa ejecutar el método desde algún lugar de tu código. Cuando invocas un método, el programa salta a la definición de ese método, ejecuta el código que contiene y luego regresa a la parte del programa que lo llamó.

Ejemplo de invocar un método:

// Definimos una clase llamada Calculator

class Calculator {

// Método que suma dos números

public int add(int a, int b) {

return a + b; }

}

public class Main {

public static void main(String[] args) {

// Crear una instancia (objeto) de la clase Calculator

Calculator calc = new Calculator();

// Invocar el método 'add' y almacenar el resultado

int result = calc.add(5, 3);

// Mostrar el resultado en la consola

System.out.println("La suma es: " + result);

}

}

