**Taller programación de Software**

**Ana Sofía Rodríguez Álvarez**

**Ficha: 3192191**

**Ins. Mairon Salazar**

**Año 2025**

**Parte 1 —Investigación Teórica**

**PREGUNTA 1**

**¿Qué es la Programación Orientada a Objetos (POO) y cuáles son sus ventajas frente a la programación estructurada?**

La programación orientada a objetos (POO) es un paradigma de programación que parte del concepto de “objetos” como base, los cuales contienen información en forma de campos (a veces también referidos como atributos, cualidades o propiedades) y código en forma de métodos.

La programación orientada a objetos (POO) se diferencia de la programación estructurada en que integra datos y comportamientos dentro de objetos, mientras que la estructurada separa funciones y datos, enfocándose en el procesamiento secuencial. En POO, primero se modelan las entidades como objetos con atributos y métodos, y luego se interactúa con ellos, en cambio, la programación estructurada prioriza funciones que operan sobre datos externos sin una relación directa entre ambos.

**PREGUNTA 2**

**¿Qué es una clase? ¿Qué es un objeto? Explica con tus palabras y da un ejemplo.**

Una clase puede entenderse como una plantilla o modelo de atributos y métodos predeterminados de un tipo de objeto.

Un objeto es un conjunto de atributos (variables o datos) y métodos o funciones relacionados entre sí.

**PREGUNTA 3**

**Define los conceptos de atributo, método y constructor en POO**

*Métodos:* Algoritmo cuya ejecución se desencadena tras la recepción de un “mensaje”, es lo que el objeto puede hacer.

*Atributos*: Características que tiene la clase.

*Constructor*: Es un método especial que se llama automáticamente cuando se crea una instancia u objeto de una clase. Su objetivo principal es inicializar los atributos de la clase y realizar cualquier otra inicialización necesaria antes de que el objeto esté listo para su uso. Los constructores tienen el mismo nombre que la clase y no tienen tipo de retorno.

**PREGUNTA 4**

**Explica qué significa encapsulación, herencia, polimorfismo y abstracción.**

**El encapsulamiento**: es cuando limitamos el acceso o damos un acceso restringido de una propiedad a los elementos que necesita un miembro y no a ninguno más.

El elemento más común de encapsulamiento son las clases, donde encapsulamos y englobamos tanto métodos como propiedades.

**La herencia:** es un mecanismo que permite derivar una clase a otra clase. En otras palabras, tendremos unas clases que serán hijos, y otras clases que serán padres.

**El polimorfismo:** se refiere a la capacidad de un objeto para adoptar más de una forma o comportarse de diferentes maneras dependiendo las situaciones que se presenten durante la ejecución.

**Abstracción** Denota las características esenciales de un objeto, donde se capturan sus comportamientos, enfocándose en lo que es relevante para el sistema y ocultando los detalles internos de su implementación.

**PREGUNTA 5**

**¿Qué es UML y para qué sirven los diagramas de clases, de objetos, de secuencia y de actividad?**

**UML** ; abreviatura de Lenguaje de Modelado Unificado, es un lenguaje de modelado estandarizado que consiste en un conjunto integrado de diagramas, desarrollado para ayudar a los desarrolladores de sistemas y software a especificar, visualizar, construir y documentar los artefactos de los sistemas de software, así como para el modelado de negocios y otros sistemas no software.

**El diagrama de clases** es una técnica de modelado fundamental que se aplica a casi todos los métodos orientados a objetos. Este diagrama describe los tipos de objetos del sistema y los diversos tipos de relaciones estáticas que existen entre ellos.

**Un diagrama de objetos** es un grafo de instancias, que incluye objetos y valores de datos. Un diagrama de objetos estático es una instancia de un diagrama de clases; muestra una instantánea del estado detallado de un sistema en un momento dado. La diferencia radica en que un diagrama de clases representa un modelo abstracto compuesto por clases y sus relaciones. Sin embargo, un diagrama de objetos representa una instancia en un momento determinado, que es de naturaleza concreta.

**El diagrama de secuencia:**  modela la colaboración de objetos basándose en una secuencia temporal. *Muestra cómo interactúan los objetos entre sí en un escenario específico de un caso de uso*

**Los diagramas de actividad**: son representaciones gráficas *de flujos de trabajo de actividades y acciones graduales, compatibles con la elección, la iteración y la concurrencia.* Describen el flujo de control del sistema de destino, como la exploración de reglas y operaciones de negocio complejas, la descripción del caso de uso y el proceso de negocio.

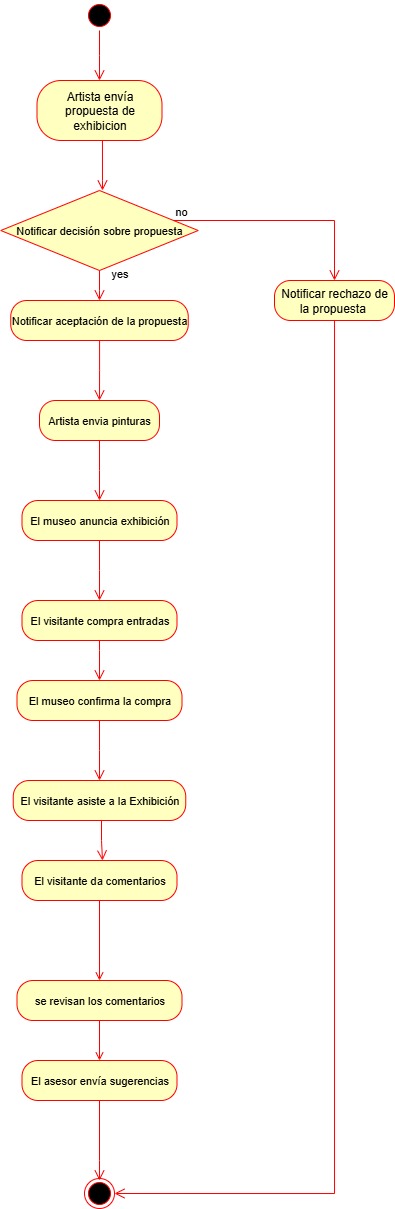
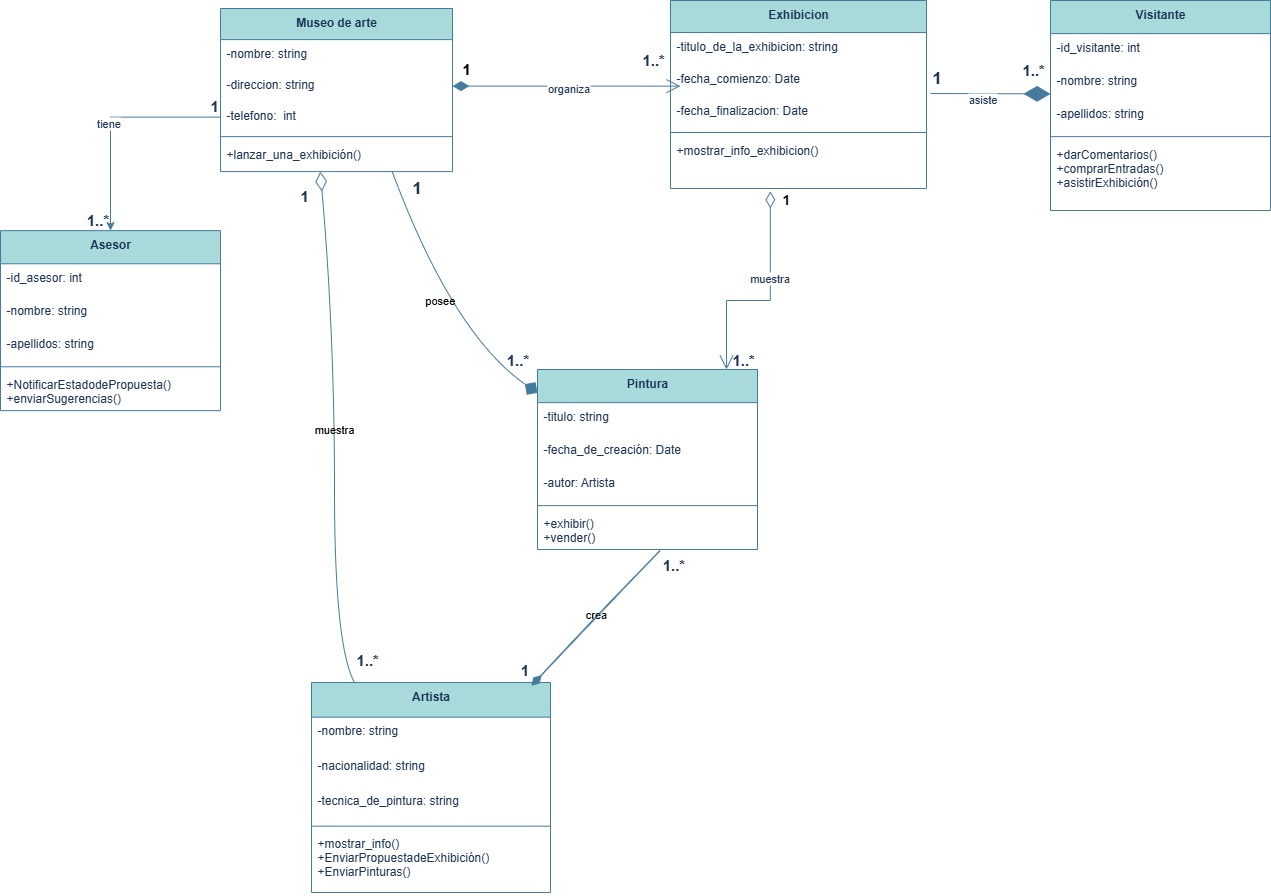
**PREGUNTA 6**

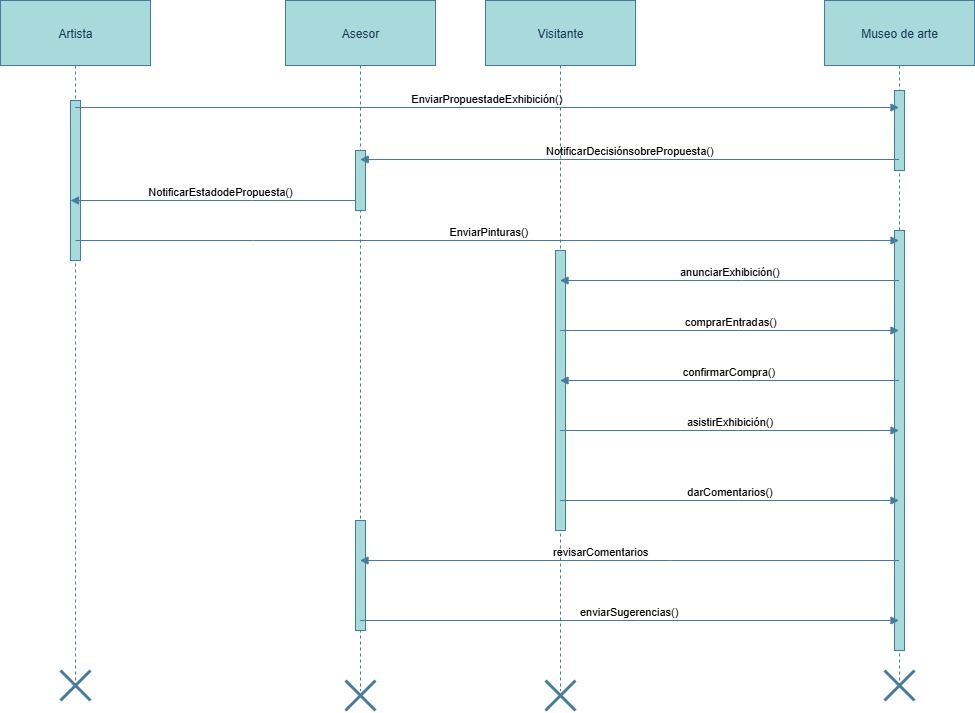
**¿Cuáles son las diferencias principales al implementar POO en Python, Javay C++?**

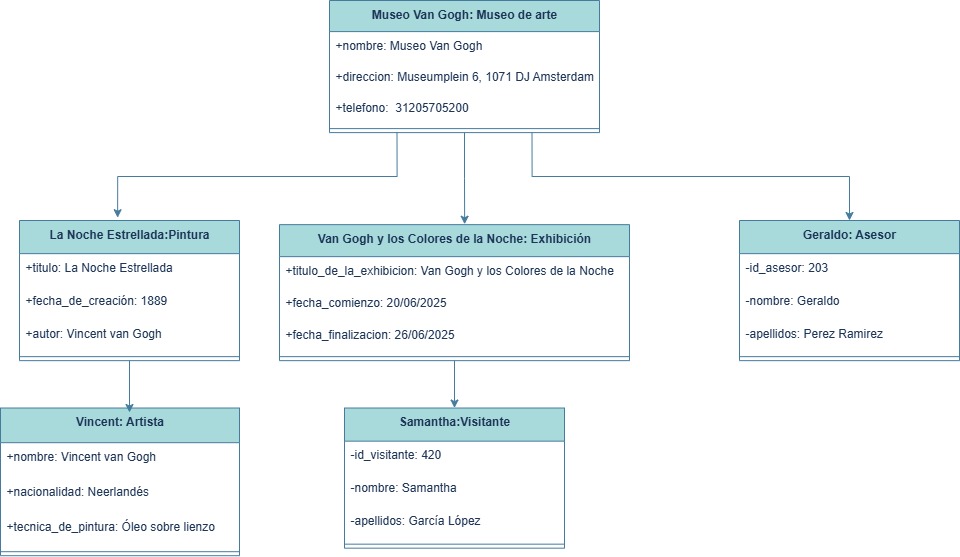
Python ofrece una Sintaxis más flexible permitiendo herencia múltiple y tipado dinámico, lo que facilita la estructura rápida, pero exige mayor cuidado en el diseño.

Java, en cambio, impone una estructura más estricta todo está dentro de clases no permite herencia múltiple directa y empleada tipado estático lo que favorece la robustez y mantenibilidad.

C++ combina paradigmas imperativos y orientados a objetos permite herencia múltiple directa manejo manual de memoria y programación de bajo nivel lo que le da gran poder, pero también mayor complejidad y riesgo de errores.







**Parte 3 —Implementación en Python**

1. Implementa en Python las clases de tu diagrama.
2. Agrega atributos, métodos y una relación de herencia.
3. Muestra un ejemplo de polimorfismo en tu código.
4. Ejecuta tu programa y muestra la salida.

class Persona: #Se crea la clase Persona

    def \_\_init\_\_(self, id, nombre, apellidos):

        self.id = id

        self.nombre = nombre

        self.apellidos = apellidos

    def mostrar\_info(self):

        print(f"Persona: {self.nombre} {self.apellidos}, ID: {self.id}")

class Artista(Persona): #Se crea la clase Artista

    def \_\_init\_\_(self, id\_artista, nombre, apellidos, nacionalidad, tecnica\_pintura): #el \_\_init\_\_ sirve para iniciar los atributos de la clase

        super().\_\_init\_\_(id\_artista, nombre, apellidos) #se llama a la clase padre con super() que sirve para heredar atributos y métodos de la clase padre

        self.nacionalidad = nacionalidad

        self.tecnica\_pintura = tecnica\_pintura

        self.pinturas = [] # Lista para almacenar muchas pinturas del artista

    def mostrar\_info(self): #método para mostrar la información del artista

        print(f"Artista: {self.nombre}, Nacionalidad: {self.nacionalidad}, Técnica: {self.tecnica\_pintura}") #se llama self para referirse a los atributos de la clase

    def enviar\_propuesta\_exhibicion(self): #método para enviar una propuesta de exhibición

        print(f"{self.nombre} envió una propuesta de exhibición.")

    def enviar\_pinturas(self, pintura): #método para enviar pinturas

        self.pinturas.append(pintura)

class Pintura: #Se crea la clase Pintura

    def \_\_init\_\_(self, titulo, fecha\_creacion, artista): #se inician los atributos de la clase

        self.titulo = titulo

        self.fecha\_creacion = fecha\_creacion

        self.autor = artista # Relación con la clase Artista

    def exhibir(self):

        print(f"Exhibiendo pintura: {self.titulo} de {self.autor.nombre}")

class MuseoDeArte:

    def \_\_init\_\_(self, id\_museo, nombre, direccion, telefono):

        self.id\_museo = id\_museo

        self.nombre = nombre

        self.direccion = direccion

        self.telefono = telefono

        self.pinturas = [] # Lista para almacenar muchas pinturas en el museo

        self.asesores = [] # Lista para almacenar muchos asesores en el museo

    def lanzar\_exhibicion(self, exhibicion):

        print(f"Museo {self.nombre} lanza la exhibición: {exhibicion.titulo}")

    def mostrar\_info(self):

        print(f"Museo: {self.nombre}, Dirección: {self.direccion}, Tel: {self.telefono}")

class Asesor(Persona):

    def \_\_init\_\_(self, id\_asesor, nombre, apellidos):

        super().\_\_init\_\_(id\_asesor, nombre, apellidos) #hereda de la clase padre

    def notificar\_estado\_propuesta(self):

        print(f"Asesor {self.nombre} notificó estado de propuesta.")

    def enviar\_sugerencias(self): # método para enviar sugerencias

        print(f"Asesor {self.nombre} envió sugerencias.")

class Exhibicion:

    def \_\_init\_\_(self, id\_exhibicion, titulo, fecha\_comienzo, fecha\_finalizacion):

        self.id\_exhibicion = id\_exhibicion

        self.titulo = titulo

        self.fecha\_comienzo = fecha\_comienzo

        self.fecha\_finalizacion = fecha\_finalizacion

        self.pinturas = [] # Lista para almacenar muchas pinturas en la exhibición

    def mostrar\_info(self):

        print(f"Exhibición: {self.titulo}, Empieza: {self.fecha\_comienzo}, Termina: {self.fecha\_finalizacion}")

class Visitante(Persona):

    def \_\_init\_\_(self, id\_visitante, nombre, apellidos):

        super().\_\_init\_\_(id\_visitante, nombre, apellidos)

    def dar\_comentarios(self):

        print(f"Visitante {self.nombre} dejó un comentario.")

    def comprar\_entradas(self):

        print(f"{self.nombre} compró una entrada.")

    def asistir\_exhibicion(self, exhibicion):

        print(f"{self.nombre} asistió a la exhibición: {exhibicion.titulo}")

def mostrar\_informacion\_objeto(objeto):

    objeto.mostrar\_info()

#Ejemplo

# Artista

vincent = Artista(

    id\_artista=1,

    nombre="Vincent",

    apellidos="van Gogh",

    nacionalidad="Neerlandés",

    tecnica\_pintura="Óleo sobre lienzo"

)

# Pintura

noche\_estrellada = Pintura(

    titulo="La Noche Estrellada",

    fecha\_creacion="1889",

    artista=vincent

)

noche\_estrellada.exhibir()  # pintura en exhibición

# Asesor

geraldo = Asesor(

    id\_asesor=203,

    nombre="Geraldo",

    apellidos="Perez Ramirez"

)

geraldo.enviar\_sugerencias()  # El asesor envía sugerencias

# Visitante

samantha = Visitante(

    id\_visitante=420,

    nombre="Samantha",

    apellidos="García López"

)

samantha.comprar\_entradas()  # El visitante compra entradas

# Exhibición

expo = Exhibicion(

    id\_exhibicion=1,

    titulo="Van Gogh y los Colores de la Noche",

    fecha\_comienzo="20/08/2025",

    fecha\_finalizacion="26/08/2025"

)

# Museo de arte

museo\_van\_gogh = MuseoDeArte(

    id\_museo=100,

    nombre="Museo Van Gogh",

    direccion="Museumplein 6, 1071 DJ Amsterdam",

    telefono=31205705200

)

#relaciones

museo\_van\_gogh.pinturas.append(noche\_estrellada) #se cita primero a el museo, la lista pinturas y se añade la pintura

museo\_van\_gogh.asesores.append(geraldo)

expo.pinturas.append(noche\_estrellada)

samantha.asistir\_exhibicion(expo)

# Comentario visitante

samantha.dar\_comentarios()

objetos = [vincent, expo, museo\_van\_gogh]

for objeto in objetos:

    mostrar\_informacion\_objeto(objeto)

**Parte 4 —Implementación en Java o C++**

1. Implementa el mismo ejercicio en **Java** o **C++**.
2. Explica cómo se definen las clases y métodos en el lenguaje que elegiste.
3. Muestra cómo aplicaste herencia y polimorfismo en tu programa.
4. Ejecuta tu código y muestra la salida.

public class GaleriaArte {  //se crea la clase principal GaleriaArteApp

    // Clase Persona

    static class Persona {

        protected int id;

        protected String nombre;

        protected String apellidos;

        public Persona(int id, String nombre, String apellidos) {

            this.id = id;

            this.nombre = nombre;

            this.apellidos = apellidos;

        }

        public void mostrarInfo() {

            System.out.println("Persona: " + nombre + " " + apellidos + ", ID: " + id);

        }

    }

    // Clase Artista

    static class Artista extends Persona {

        private String nacionalidad;

        private String tecnicaPintura;

        public Artista(int id, String nombre, String apellidos, String nacionalidad, String tecnicaPintura) {

            super(id, nombre, apellidos);

            this.nacionalidad = nacionalidad;

            this.tecnicaPintura = tecnicaPintura;

        }

        public void mostrarInfo() {

            System.out.println("Artista: " + nombre + ", Nacionalidad: " + nacionalidad + ", Técnica: " + tecnicaPintura);

        }

        public void enviarPropuestaExhibicion() {

            System.out.println(nombre + " envió una propuesta de exhibición.");

        }

        public void enviarPinturas(Pintura pintura) {

            System.out.println(nombre + " recibió la pintura: " + pintura.getTitulo());

        }

    }

    // Clase Pintura

    static class Pintura {

        private String titulo;

        private String fechaCreacion;

        private Artista autor;

        public Pintura(String titulo, String fechaCreacion, Artista autor) {

            this.titulo = titulo;

            this.fechaCreacion = fechaCreacion;

            this.autor = autor;

        }

        public void exhibir() {

            System.out.println("Exhibiendo pintura: " + titulo + " de " + autor.nombre);

        }

        public String getTitulo() {

            return titulo;

        }

    }

    // Clase MuseoDeArte

    static class MuseoDeArte {

        private int idMuseo;

        private String nombre;

        private String direccion;

        private long telefono;

        public java.util.ArrayList<Pintura> pinturas = new java.util.ArrayList<>();

        public java.util.ArrayList<Asesor> asesores = new java.util.ArrayList<>();

        public MuseoDeArte(int idMuseo, String nombre, String direccion, long telefono) {

            this.idMuseo = idMuseo;

            this.nombre = nombre;

            this.direccion = direccion;

            this.telefono = telefono;

        }

        public void lanzarExhibicion(Exhibicion exhibicion) {

            System.out.println("Museo " + nombre + " lanza la exhibición: " + exhibicion.getTitulo());

        }

        public void mostrarInfo() {

            System.out.println("Museo: " + nombre + ", Dirección: " + direccion + ", Tel: " + telefono);

        }

    }

    // Clase Asesor

    static class Asesor extends Persona {

        public Asesor(int id, String nombre, String apellidos) {

            super(id, nombre, apellidos);

        }

        public void notificarEstadoPropuesta() {

            System.out.println("Asesor " + nombre + " notificó estado de propuesta.");

        }

        public void enviarSugerencias() {

            System.out.println("Asesor " + nombre + " envió sugerencias.");

        }

    }

    // Clase Exhibicion

    static class Exhibicion {

        private int idExhibicion;

        private String titulo;

        private String fechaComienzo;

        private String fechaFinalizacion;

        public java.util.ArrayList<Pintura> pinturas = new java.util.ArrayList<>();

        public Exhibicion(int idExhibicion, String titulo, String fechaComienzo, String fechaFinalizacion) {

            this.idExhibicion = idExhibicion;

            this.titulo = titulo;

            this.fechaComienzo = fechaComienzo;

            this.fechaFinalizacion = fechaFinalizacion;

        }

        public void mostrarInfo() {

            System.out.println("Exhibición: " + titulo + ", Empieza: " + fechaComienzo + ", Termina: " + fechaFinalizacion);

        }

        public String getTitulo() {

            return titulo;

        }

    }

    // Clase Visitante

    static class Visitante extends Persona { //extends indica que hereda de la clase Persona

        public Visitante(int id, String nombre, String apellidos) {

            super(id, nombre, apellidos);

        }

        public void darComentarios() {

            System.out.println("Visitante " + nombre + " dejó un comentario.");

        }

        public void comprarEntradas() {

            System.out.println(nombre + " compró una entrada.");//El sistem out es para imprimir en consola

        }

        public void asistirExhibicion(Exhibicion exhibicion) {//PUBLIC ES UN ESPECIFICADOR DE ACCESO, VOID INDICA QUE NO RETORNA NADA

            System.out.println(nombre + " asistió a la exhibición: " + exhibicion.getTitulo());

        }

    }

    // Método main: es el punto de entrada del programa

    public static void main(String[] args) { //public static especificador de acceso y modificador de método, void indica que no retorna nada

        Artista vincent = new Artista(1, "Vincent", "van Gogh", "Neerlandés", "Óleo sobre lienzo");

        vincent.mostrarInfo();

        vincent.enviarPropuestaExhibicion();

        Pintura nocheEstrellada = new Pintura("La Noche Estrellada", "1889", vincent); //new indica que se está creando un nuevo objeto

        nocheEstrellada.exhibir();

        Asesor geraldo = new Asesor(203, "Geraldo", "Perez Ramirez");

        geraldo.enviarSugerencias();

        Visitante samantha = new Visitante(420, "Samantha", "García López");

        samantha.comprarEntradas();

        Exhibicion expo = new Exhibicion(1, "Van Gogh y los Colores de la Noche", "20/08/2025", "26/08/2025");

        expo.mostrarInfo();

        MuseoDeArte museoVanGogh = new MuseoDeArte(100, "Museo Van Gogh", "Museumplein 6, 1071 DJ Amsterdam", 3120505200L);

        museoVanGogh.mostrarInfo();

        museoVanGogh.pinturas.add(nocheEstrellada);

        museoVanGogh.asesores.add(geraldo);

        expo.pinturas.add(nocheEstrellada);

        samantha.asistirExhibicion(expo);

        samantha.darComentarios();

        }

    }

**Parte 5 —Reflexión Final**

1. **¿Qué similitudes y diferencias encontraste entre Python, Java y C++ al trabajar con POO?**

**Similitudes**

Todos permiten que una clase herede de otra para reutilizar código, al igual que permiten el polimorfismo.

**Diferencias**

Python: Es más simple y legible, no usa llaves {} ni punto y coma ;. No se necesita declarar el tipo de variable. El constructor se llama \_\_init\_\_. Tipado dinámico (no necesita especificar el tipo de variable).

Java: Usa llaves {} para bloques de código y punto y coma al final de cada línea. Se debe declarar el tipo de cada variable. El constructor tiene el mismo nombre que la clase. Tipado estático (especificar el tipo de cada variable).

1. **¿Qué dificultades tuviste al diseñar los diagramas UML?**

Al momento de diseñar las relaciones en el diagrama de clases.

1. **¿Qué parte del taller consideras más importante para tu formación como técnico en desarrollo de software?**

La implementación del diagrama de clases, ya que con este se puede hacer un código bien estructurado y tener una mejor visualización acerca de cómo podrá funcionar.

1. **Propón una mejora o extensión para tu ejercicio (ej.: agregar persistencia, interfaz gráfica o más clases).**

Se implementó listas a el código de Python y de Java para poder agregar muchos objetos dentro de las clases.