Universidad Tecnológica de Durango

Tecnologías de la Información

Programación Orientada a Objetos

Actividades

“Evidencias de Actividades y Tareas”

Alumnos:

- Lomas Corral Edson

3°B BIS

Docente:

- Ing. Dagoberto Fiscal Gurrola, M.T.I.

Octubre 2025

UNIVERSIDAD TECNOLÓGICA DE DURANGO

Tabla de Ilustraciones

UNIVERSIDAD TECNOLÓGICA DE DURANGO

Actividad 1

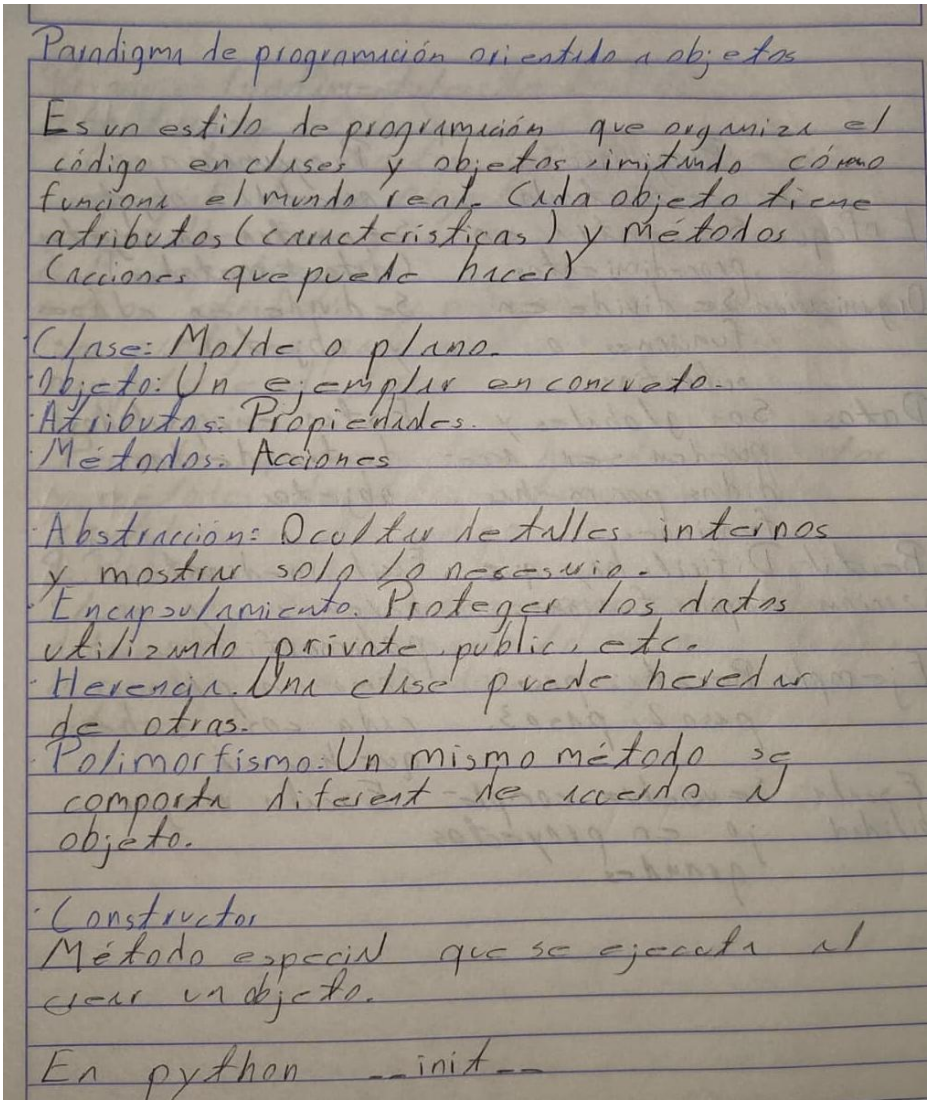
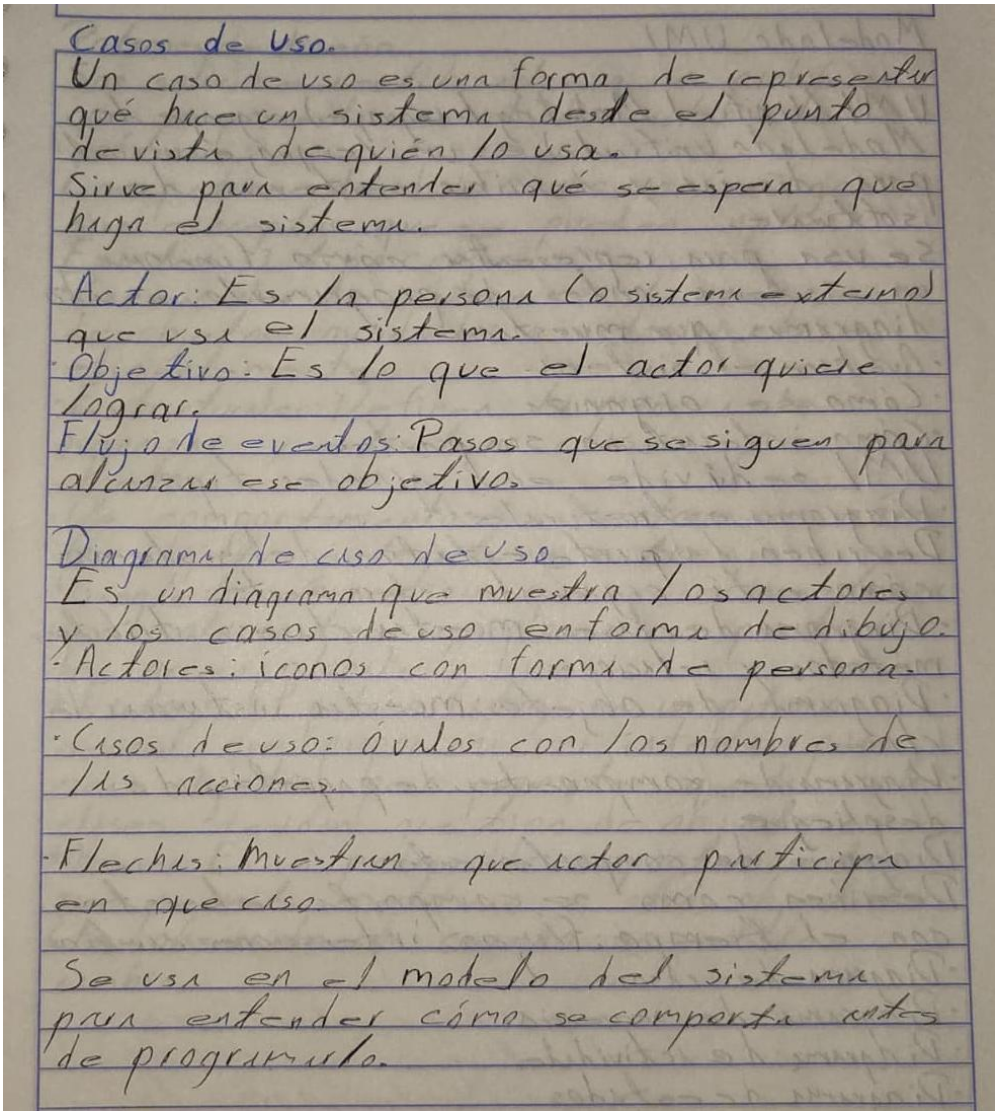


Ilustración 1:Introducción a los conceptos del paradigma orientado a objetos.

En esta evidencia se presentan los conceptos fundamentales del paradigma de programación orientado a objetos (POO), explicando cómo este modelo organiza el código en clases y objetos dotados de atributos y métodos. Además, se definen sus pilares clave como la abstracción, encapsulamiento, herencia y polimorfismo, principios que en conjunto permiten dividir un programa en componentes lógicos y reutilizables, mejorando así la claridad, organización y mantenimiento del software.

UNIVERSIDAD TECNOLÓGICA DE DURANGO

Actividad 2



Casos de Uso.
Un caso de uso es una forma de representar qué hace un sistema desde el punto de vista de quién lo usa.
Sirve para entender qué se espera que haga el sistema.

Actor: Es la persona (o sistema externo) que usa el sistema.
Objetivo: Es lo que el actor quiere lograr.
Flujo de eventos: Pasos que se siguen para alcanzar ese objetivo.

Diagrama de caso de uso
Es un diagrama que muestra los actores y los casos de uso en forma de dibujo.
Actores: iconos con forma de persona.

Casos de uso: óvalos con los nombres de las acciones.

Flechas: Muestran qué actor participa en qué caso.

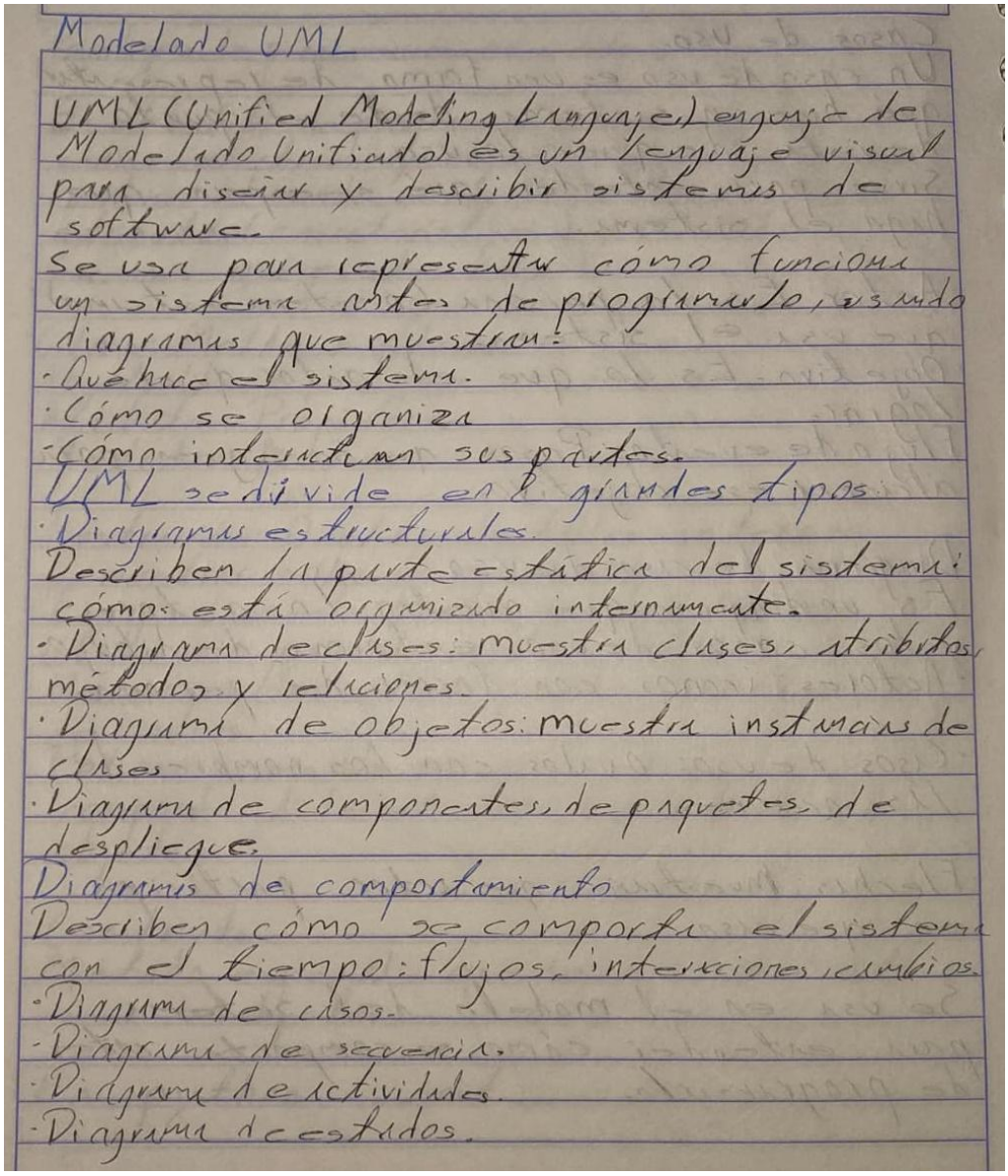
Se usa en el modelo del sistema para entender cómo se comporta antes de programarlo.

Ilustración 2: Concepto de casos de uso, elementos de caso de uso

Esta evidencia define el concepto de casos de uso y sus componentes principales: el actor, el objetivo y el flujo de eventos. Además, explica cómo estos elementos se usan en un diagrama para representar la funcionalidad de un sistema desde la perspectiva de quien lo utiliza.

UNIVERSIDAD TECNOLÓGICA DE DURANGO

Actividad 3


Modelado UML

UML (Unified Modeling Language) (lenguaje de Modelado Unificado) es un lenguaje visual para diseñar y describir sistemas de software.
Se usa para representar cómo funciona un sistema antes de programarlo, usando diagramas que muestran:
- Qué hace el sistema.
- Cómo se organiza
- Cómo interactúan sus partes.

UML se divide en 2 grandes tipos.
- Diagramas estructurales.

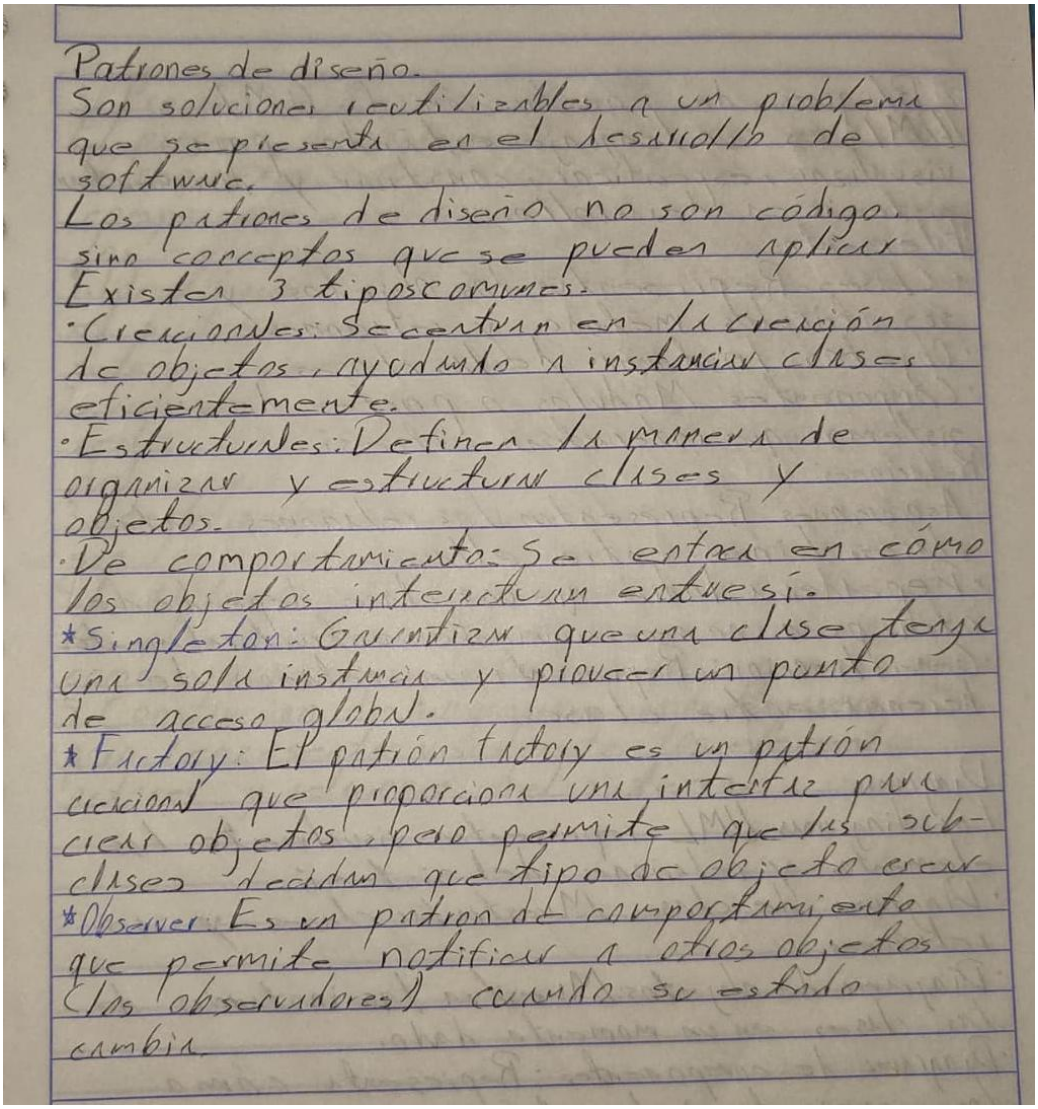
Describen la parte estática del sistema: cómo está organizado internamente.
- Diagrama de clases: muestra clases, atributos, métodos y relaciones.
- Diagrama de objetos: muestra instancias de clases
- Diagrama de componentes, de paquetes, de despliegue.

Diagramas de comportamiento
Describen cómo se comporta el sistema con el tiempo: flujos, interacciones, cambios.
- Diagrama de casos.
- Diagrama de secuencia.
- Diagrama de actividades.
- Diagrama de estados.

Ilustración 3: Modelado UML.

Esta evidencia introduce el Modelado UML (Lenguaje Unificado de Modelado), presentándolo como un lenguaje visual para diseñar y describir sistemas de software. Se explica su división en dos tipos principales de diagramas: los estructurales, que definen la organización estática del sistema (como el diagrama de clases), y los de comportamiento, que muestran su dinámica e interacciones(como el diagrama de secuencia).

Actividad 4



Patrones de diseño.
Son soluciones reutilizables a un problema que se presenta en el desarrollo de software.
Los patrones de diseño no son código, sino conceptos que se pueden aplicar
Existen 3 tipos comunes:
·Creacionales: Se centran en la creación de objetos, ayudando a instanciar clases eficientemente.
·Estructurales: Definen la manera de organizar y estructurar clases y objetos.
·De comportamiento: Se enfocan en cómo los objetos interactúan entre sí.
*Singleton: Garantiza que una clase tenga una sola instancia y provee un punto de acceso global.
*Factory: El patrón factory es un patrón creacional que proporciona una interfaz para crear objetos, pero permite que las subclases decidan que tipo de objeto crear
*Observer: Es un patron de comportamiento que permite notificar a otros objetos (los observadores) cuando su estado cambia.

Ilustración 4:Patrones básicos de diseño (singleton, factory, observer)

Esta evidencia introduce el concepto de patrones de diseño como soluciones reutilizables a problemas comunes en el desarrollo de software. Se presenta su clasificación en tres tipos (creacionales, estructurales y de comportamiento) y se explican tres patrones básicos: Singleton, Factory y Observer, detallando la finalidad de cada uno.

UNIVERSIDAD TECNOLÓGICA DE DURANGO

Retroalimentación

A lo largo del desarrollo de estas actividades, se ha fortalecido significativamente la comprensión de los fundamentos del paradigma orientado a objetos. Cada tema abordado ha contribuido a construir una base sólida, no solo teórica sino también práctica, para modelar y resolver problemas complejos de manera modular y escalable.

El estudio de conceptos como clases, objetos, herencia y polimorfismo permitió entender cómo estructurar el software a imagen de problemas del mundo real. Dominar el encapsulamiento y el instanciamiento de objetos ha sido fundamental para crear componentes de software robustos, reutilizables y fáciles de mantener, promoviendo un código limpio y bien organizado.

La introducción a los casos de uso y al modelado UML aportó una perspectiva crucial sobre la fase de análisis y diseño. Comprender cómo definir actores, objetivos y flujos de eventos, y cómo representar visualmente la arquitectura del sistema, ha facilitado la planificación y la comunicación de las soluciones antes de escribir una sola línea de código.

El análisis de patrones de diseño básicos como Singleton, Factory y Observer brindó una visión estratégica para resolver problemas recurrentes de manera elegante y eficiente. Comprender estas plantillas probadas ha sido clave para fortalecer el pensamiento de diseño y construir software más flexible, adaptable y robusto.

Finalmente, integrar todos estos conocimientos ha demostrado la importancia de seguir un proceso metódico que abarca desde el análisis hasta la implementación. Con esta preparación, se está en condiciones de avanzar hacia el desarrollo de sistemas de software complejos, garantizando soluciones tecnológicas de mayor calidad y valor.