Universidad Corporativa | Programas Académicos

Ingeniería en Desarrollo de Software

Actividad | 2 | Diagramas de Clases y Objetos.

Lenguaje Unificado de Modelado

Ingeniería en Desarrollo de Software

academia global

TUTOR: Eduardo Israel Castillo García.

ALUMNO: Sarahi Jaqueline Gómez Juárez.

FECHA: lunes, 16 de septiembre de 2024.

Índice

Introducción

En el presente proyecto, se elaborará el Diagrama de Clases, utilizando una herramienta de modelado, con base en la primera actividad relacionada con el desarrollo del proyecto "**Sorteo Vacaciones en Familia"** para una tienda departamental, resulta imperativo proceder con la elaboración de los diagramas de clases, estos diagramas no solo permitirán identificar y describir los objetos involucrados en el sistema, sino también detallar sus características y las relaciones que los vinculan, este seguirá las mismas directrices que se encuentran descritas dentro de este enlace**: Etapa 1 UML: "Sorteo Vacaciones en Familia":** https://drive.google.com/file/d/1HadvgL800yKtfO6B1PEUmqDIntCKaNUf/view?usp=sharing

Los **Diagramas de Clases y Objetos** son instrumentos fundamentales en la programación orientada a objetos **(POO)** y en el modelado de sistemas.

El **Diagrama de Clases** representa la estructura estática de un sistema, mostrando las clases, sus atributos, métodos y las relaciones entre ellas.

Está compuesto por:

Clases: Rectángulos divididos en tres secciones: nombre de la clase, atributos y métodos.

Relaciones: Conexiones entre clases que pueden ser asociaciones, herencias, agregaciones, composiciones, etc.

Visibilidad: Indicadores de acceso a atributos y métodos (público +, privado –, protegido #).

Multiciplidad: Define cuántas instancias de una clase pueden estar asociadas a una instancia de otra clase.

Por su parte, el **Diagrama de Objetos** representa una instancia específica de un sistema

en un momento determinado, mostrando objetos y sus relaciones

Está compuesto por:

Objetos: Rectángulos con el nombre del objeto y de la clase subrayado.

Atributos: Muestran los valores específicos de los atributos de los objetos.

Relaciones: Conexiones entre objetos que representan la interacción en un momento determinado.

Estos diagramas son esenciales en la fase de **Diseño y Análisis del Sistema** para visualizar y comunicar la estructura y el comportamiento de un sistema de software.

Estos diagramas son esenciales en las fases de diseño y análisis del sistema, ya que facilitan la visualización y comunicación de la estructura y el comportamiento de un sistema de software, en síntesis, los **Diagramas de Clases** muestran la estructura estática del sistema, incluyendo clases, atributos, métodos y sus relaciones, mientras que los **Diagramas de Objetos** ilustran instancias específicas de dichas clases en un momento dado, mostrando los objetos y sus interacciones, ambos son elementos clave en el diseño y análisis de sistemas orientados a objetos, tal como lo menciona Grady Booch:

> "Los diagramas de clases y objetos en UML proporcionan una representación clara y detallada de la estructura y el comportamiento de un sistema, facilitando la comunicación y el entendimiento entre los desarrolladores y las partes interesadas."

Descripción

Este documento presenta el diseño de un **Diagrama de Clases y Objetos** elaborado mediante una herramienta de modelado, basado en la primera fase del desarrollo del proyecto: "**Sorteo Vacaciones en Familia"** para una tienda departamental.

Un **Diagrama de Clases** es un tipo de diagrama en UML (Lenguaje de Modelado Unificado) ilustra la estructura de un sistema mostrando sus **clases**, **atributos**, **métodos** y las **relaciones** entre las clases, es una representación estática que define cómo los elementos del sistema interactúan entre ellas, este diagrama estático define cómo los elementos del sistema interactúan a través de propiedades y comportamientos.

Características:

Representa la estructura de datos y sus interrelaciones.

Se utiliza para modelar sistemas orientados a objetos.

Permite definir las relaciones entre clases como asociación, herencia, composición y agregación.

Muestra la multiplicidad o cardinalidad de las relaciones.

Una **Clase:** Es una plantilla o modelo a partir del cual se crean objetos, es la unidad básica de la Programación Orientada a Objetos (POO) y contiene atributos (propiedades) y métodos (comportamientos) que definen los objetos.

Características:

Define el tipo de datos y las operaciones que se pueden realizar sobre ellos.

Puede incluir **modificadores de acceso** como public, private, o protected que controlan la visibilidad de los atributos y métodos.

Las clases pueden tener relaciones con otras clases (herencia, asociación, composición, agregación).

Los **Atributos** son las propiedades o características que describen el estado de una clase, representan los datos que un objeto de una clase puede almacenar.

Características:

Cada atributo tiene un Tipo de Dato (como int, string, boolean, etc.).

Pueden tener modificadores de acceso (private, public, etc.).

Pueden tener valores predeterminados o ser calculados dinámicamente.

Son variables que pertenecen a la clase y se definen dentro de ella.

Los **Métodos** son las funciones o procedimientos que definen el comportamiento de los objetos de una clase, los métodos pueden manipular los atributos de la clase y responder a eventos o solicitudes.

Características:

Tienen un nombre, pueden tener parámetros y un valor de retorno.

Permiten a los objetos realizar acciones o comportamientos específicos.

Pueden ser públicos, privados o protegidos según los modificadores de acceso.

Pueden sobrecargarse o sobrescribirse en caso de herencia.

Las **Relaciones** en un diagrama de clases describen cómo las clases interactúan entre sí, las relaciones más comunes son **asociación**, **herencia**, **composición** y **agregación**.

Características:

Asociación: Relación entre clases que indica que los objetos de una clase utilizan o interactúan con los objetos de otra clase.

Herencia: Define una relación”**es-un"** entre clases, donde una clase hija hereda atributos

y métodos de una clase padre.

Composición: Relación fuerte donde una clase contiene a otra clase como parte de su estructura, y la vida de los objetos involucrados está intrínsecamente ligada.

Agregación: Relación débil en la que una clase contiene a otra pero ambas pueden existir independientemente.

La **Multiplicidad** describe el número de instancias de una clase que pueden estar asociadas con una instancia de otra clase.

Características:

Se expresa en forma de rango (1..*, 0..1) que indica cuántos objetos pueden estar relacionados.

Permite especificar la cantidad mínima y máxima de objetos asociados.

Es importante en la modelización de relaciones de agregación y composición para entender las restricciones de cardinalidad.

Este conjunto de conceptos resulta esencial para comprender la estructuración y modelado eficiente de sistemas orientados a objetos, una vez asimilados los elementos que componen un diagrama de clases, es pertinente abordar los componentes de un diagrama de objetos.

Un **Diagrama de Objetos** que es un tipo de diagrama en UML que muestra una instancia concreta de un **Diagrama de Clases** en un momento específico, representa los objetos en el sistema y las relaciones entre ellos, como se encuentran en un estado particular de ejecución del programa.

Características:

Es una representación estática, pero a diferencia del diagrama de clases, muestra

instancias específicas (objetos) y no las plantillas (clases).

Se utiliza para visualizar ejemplos concretos de relaciones, valores de atributos y conexiones entre objetos.

Útil para representar el estado de un sistema en un momento particular, permitiendo la inspección de la estructura en tiempo de ejecución.

Ayuda a clarificar cómo las clases interactúan en una situación particular.

Un **Objeto:** Es una instancia concreta de una clase, representa una entidad del mundo real o lógico en un sistema orientado a objetos, con valores específicos asignados a sus atributos.

Características:

Contiene datos (atributos) y métodos que son implementaciones específicas de una clase.

Cada objeto tiene una identidad única (nombre o referencia) que lo distingue de otros objetos.

Los valores de los atributos pueden variar entre objetos de la misma clase.

En un **Diagrama de Objetos**, los **Atributos** representan los valores concretos que los objetos tienen en el momento específico mostrado en el diagrama.

Características:

Reflejan el estado actual del objeto.

Cada atributo tiene un valor específico asignado en el objeto, que puede diferir entre diferentes instancias.

Se muestran como pares de nombre-valor, indicando los valores actuales de las propiedades del objeto.

En los **Diagramas de Objetos**, los **Métodos** no se representan explícitamente como en los diagramas de clases, ya que el enfoque principal está en las instancias y sus atributos, sin

embargo, los métodos pueden inferirse de las clases que los objetos representan.

Características:

Los métodos están implícitos en los objetos basados en la clase a la que pertenecen.

Aunque no se muestran directamente, la ejecución de métodos cambiará los valores de los atributos de los objetos, afectando su estado.

Las **Relaciones** en un **Diagrama de Objetos** representan las conexiones entre instancias de objetos. Estas relaciones son instancias concretas de las relaciones definidas en el diagrama de clases.

Características:

Muestran cómo los objetos interactúan entre sí en un momento específico.

Se pueden representar como asociaciones, agregaciones o composiciones, dependiendo de cómo se conectan los objetos.

Ayudan a visualizar la comunicación y dependencias entre objetos en el sistema.

Multiplicidad en un Diagrama de Objetos

En un **Diagrama de Objetos**, la **Multiplicidad** se manifiesta mostrando el número de instancias concretas de una clase que están asociadas en un estado particular del sistema.

Características:

Refleja las instancias reales que cumplen con las reglas de multiplicidad definidas en el diagrama de clases.

Permite ver cómo los objetos individuales están relacionados en un contexto específico, verificando si se cumplen las restricciones de cardinalidad.

Como es evidente, en ambos escenarios las relaciones desempeñan un papel esencial en la definición de las interacciones entre clases, cada tipo de relación posee un significado

particular y cumple una función específica dentro del sistema:

Asociación: Es la relación más básica entre dos clases, que indica que las instancias de una clase están conectadas con instancias de otra clase, las asociaciones representan un vínculo lógico entre los objetos.

Características:

Es una relación de”**uso"** entre dos clases.

Se puede etiquetar con un nombre para describir el tipo de interacción.

Tiene **multiplicidad**, lo que define cuántas instancias de una clase pueden asociarse con instancias de la otra clase (por ejemplo, 1..*, 0..1).

Ejemplo: Un “Profesor” puede estar asociado a “Cursos” que enseña.

Agregación: Es un tipo especial de asociación que representa una relación”tiene un" o”**es parte de",** indica que una clase es una colección o conjunto de otras clases, pero las partes pueden existir independientemente del todo.

Características:

Representa una relación débil o flexible entre las clases.

El ciclo de vida de los objetos de la clase agregada no depende de la clase que la contiene.

Se representa gráficamente con una línea que termina en un rombo vacío en la clase contenedora.

Ejemplo: Un “Departamento” tiene “Empleados”, pero los empleados pueden existir independientemente del departamento.

Composición: Es una relación más fuerte que la agregación, donde la clase contenedora controla completamente el ciclo de vida de las clases contenidas. Si el objeto contenedor es

destruido, las instancias de las clases contenidas también lo son.

Características:

Representa una relación de”todo-parte" fuerte.

La existencia de la clase parte depende del todo.

Se representa con una línea que termina en un rombo sólido en la clase contenedora.

Ejemplo: Una”Casa" está compuesta por”Habitaciones", y si la casa es destruida, las habitaciones también lo son.

Herencia: También llamada **Generalización**, describe una relación jerárquica entre una clase base y una o más clases derivadas. Las clases derivadas heredan los atributos y métodos de la clase base.

Características:

Define una relación “es un".

Las clases derivadas pueden tener atributos y métodos adicionales, y pueden sobrescribir los métodos de la clase base.

Se representa con una línea que termina en un triángulo apuntando hacia la clase base.

Ejemplo: Un “Gato" hereda de “Mamífero”, lo que significa que un gato es un tipo de mamífero.

Dependencia: Es una relación débil que indica que una clase utiliza los métodos o atributos de otra clase, pero la relación no es tan fuerte como en la asociación.

Características:

Se usa para representar que un cambio en la clase dependida podría afectar a la clase dependiente.

Se representa con una línea discontinua y una flecha apuntando hacia la clase de la que

depende.

Ejemplo: Una “Clase" depende de una “Biblioteca” para usar ciertas funcionalidades, pero la clase puede existir independientemente de la biblioteca.

Realización: Es una relación utilizada en UML para indicar que una clase concreta implementa una interfaz, la interfaz define el comportamiento esperado, y la clase concreta proporciona la implementación de ese comportamiento.

Características:

Relaciona una interfaz con una clase concreta que la implementa.

Se representa con una línea discontinua y un triángulo hueco apuntando hacia la interfaz.

Ejemplo: La clase “Coche” realiza la interfaz “Vehículo”, implementando los métodos definidos en la interfaz.

Estas relaciones optimizan la estructuración de sistemas orientados a objetos al representar con precisión las interacciones entre las diversas clases que los integran, facilitando la modelación de interacciones complejas de manera clara y eficaz, promoviendo una comprensión profunda y una organización coherente del sistema.

Justificación:

El Objetivo del proyecto de la segunda etapa del “Sorteo Vacaciones en Familia” en el presente documento está centrada en la necesidad de crear Diagramas de Clases para visualizar y estructurar de manera eficiente un sistema de software que gestione Sorteos y compras dentro de una tienda departamental.

El proyecto responde a la necesidad de la tienda departamental de tener un sistema que gestione las compras y Sorteos de manera eficiente, los Diagramas de Clases permiten identificar y describir los objetos involucrados, como usuarios, transacciones, boletos, productos y Sorteos, así como sus relaciones y características.

Los **Diagramas de Clases y objetos** proporcionan una representación visual clara de la estructura y el comportamiento del sistema, esto es crucial para el desarrollo y mantenimiento del sistema, ya que facilita la comprensión tanto para los desarrolladores como para los interesados en el proyecto, mejorando la comunicación y el diseño colaborativo.

Al utilizar UML en el Diseño del Sistema permite un desarrollo más estructurado, lo que favorece la creación de un software escalable, flexible y fácil de mantener, además, estos diagramas ayudan a identificar relaciones clave como la herencia, la agregación y la composición, optimizando la reutilización de componentes.

El sistema debe cumplir con los requisitos establecidos en la primera etapa del proyecto, como lo son: La compra mínima de $500 en departamentos participantes, la gestión de boletos (Boletos Ganadores, Boletos Participantes (Perdedores), Vigencia del Sorteo: (Fechas y Número máximo de Boletos Ganadores) etc..) y Sorteos para los clientes, empleados y administradores, asegurando que los usuarios puedan interactuar correctamente con el sistema en relación a las

compras y Sorteos.

Al contar con un diseño bien estructurado, el sistema puede ser fácilmente actualizado o modificado conforme crecen las necesidades de la tienda, lo cual es esencial para mantener la competitividad y la adaptabilidad del negocio en el futuro.

En resumen, el proyecto se centra en la necesidad de un sistema eficiente, estructurado y escalable que pueda gestionar las compras y Sorteos de la tienda departamental, asegurando un proceso de desarrollo claro y colaborativo.

Desarrollo:

Contextualización:

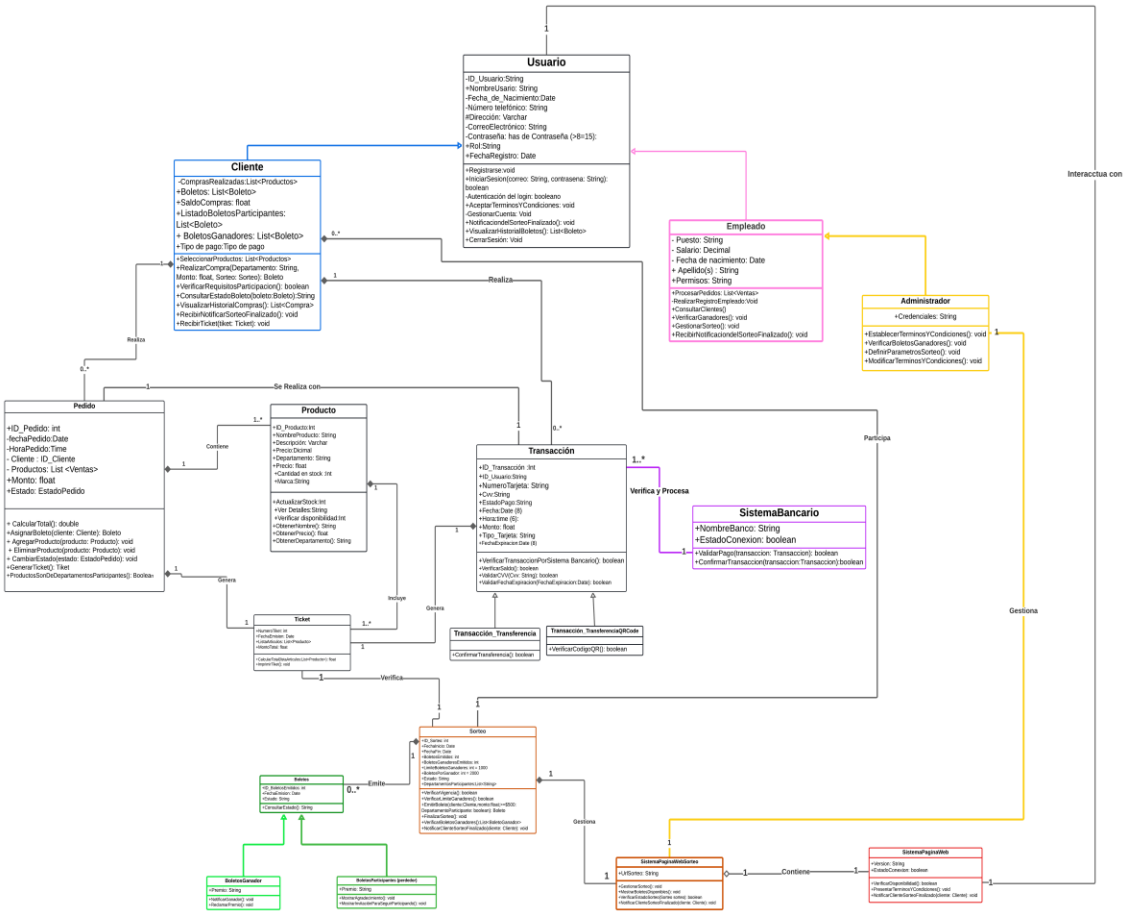
Con base en la primera actividad acerca del desarrollo del”**Sorteo Vacaciones en familia”** para una tienda departamental, será necesario generar los diagramas de clase, estos diagramas permitirán identificar y a su vez describir los objetos involucrados en el sistema, sus características y relaciones; es importante revisar la contextualización de la actividad 1 para recordar los requerimientos.

Actividad:

● Generar el o los Diagrama(s) de Clase(s) con ayuda de una herramienta de modelado libre o privativa.

Figura 1

Diagrama de Clases: "Sorteo en Familia":



Nota: La imagen es un **Diagrama de Clases UML** que representa un sistema de Gestión de Compras, Boletos, Transacciones y de un Sorteo:

Usuario:

Esta clase representa a los usuarios del sistema con atributos como ID, Nombre, Correo Electrónico, Contraseña, Fecha de Registro, entre otros, hay métodos relacionados con el Inicio y Cierre de Sesión, Actualización de Términos y Condiciones, y consultas sobre Boletos Ganados.

Cliente (subclase de Usuario):

Los clientes tienen acceso a información adicional como compras realizadas, Boletos,

Saldo de Compras, y Boletos Ganadores.

Métodos importantes incluyen seleccionar productos, Verificar Estado de Boletos, y recibir notificaciones de sorteos finalizados.

Empleado (también una subclase de Usuario):

Los empleados tienen atributos adicionales como el puesto, salario, permisos, y responsabilidades relacionadas con la revisión de sorteos, verificación de ganadores, y recepción de notificaciones.

Administrador:

Esta clase tiene credenciales y métodos para gestionar términos y condiciones, verificar boletos ganadores, y modificar los parámetros de los sorteos.

Producto:

Representa los productos que están disponibles para la compra, con atributos como ID, nombre, descripción, stock disponible, y precio.

Métodos incluyen la actualización de stock y la conversión de texto a imágenes.

Pedido:

Los pedidos contienen información como el ID, fecha y hora, estado del pedido, y una lista de productos.

Métodos importantes incluyen la asignación de boletos y la generación de facturas.

Transacción:

Maneja las transacciones financieras vinculadas a los pedidos, incluyendo ID, tipo de transacción, estado, fecha, y hora.

Se verifica y procesa a través de un **SistemaBancario**, que contiene métodos para validar y confirmar transacciones.

SistemaBancario:

Se encarga de validar y confirmar las transacciones de las compras realizadas por los clientes.

Sorteo:

Relacionado con el sistema de boletos y sorteos, contiene la información sobre la fecha del sorteo, los ganadores, los boletos emitidos, y las reglas del sorteo.

Se emiten boletos y se asignan a los clientes a través de este sistema.

Ticket:

Los tickets son generados a partir de las transacciones y verifican si las transacciones fueron exitosas o no.

SistemaPagos y SistemaSorteo:

Estos parecen ser sistemas adicionales que contienen utilidades y manejan diferentes aspectos del sistema, como la generación de boletos y la confirmación de transacciones.

En resumen, este Diagrama representa la interacción entre Clientes, Empleados, Administradores, Productos, Pedidos, Transacciones y Sorteos, junto con los Sistemas Bancarios y de Pagos necesarios para procesar dichas acciones, el propósito general es manejar compras, la asignación de Boletos y la Verificación de Ganadores del Sorteo, para más detalles, puedes visualizar el siguiente enlace en la herramienta de modelado Lucid chart: https://lucid.app/lucidchart/e3dd704d-9f4a-4961-8271-ce7dbee56154/edit?viewport_loc=-3184%2C-1620%2C4440%2C1844%2C0_0&invitationId=inv_d1a389ab-1fbf-4662-b6b4-187d89d234c4

Conclusión:

La adquisición del conocimiento sobre **Diagramas de Clases y Objetos** es de suma relevancia, tanto en el ámbito profesional como en situaciones cotidianas:

Estos diagramas, propios de la metodología de modelado orientado a objetos (UML, por sus siglas en inglés), no solo facilitan la comprensión de sistemas complejos en el desarrollo de software, sino que también promueven un enfoque estructurado para la resolución de problemas en general, este enfoque permite desglosar la estructura de un sistema de software en términos de clases, objetos y sus interacciones, cada clase representa una entidad con atributos y comportamientos definidos, ofreciendo una manera clara de modelar las funcionalidades y responsabilidades de cada componente del sistema, este recurso no solo resulta útil para el desarrollo de sistemas complejos, sino que también es aplicable a cualquier situación que requiera una organización sistemática de la información.

En el entorno laboral, este enfoque es indispensable para cualquier organización que busque soluciones bien estructuradas, ya que permite a los equipos dividir grandes problemas en partes más pequeñas y manejables, el pensamiento orientado a objetos fomenta la creación de sistemas modulares y escalables, lo que resulta esencial en proyectos que requieren mantenimiento y evolución constante.

Otro beneficio importante es que promueve una comunicación eficaz entre equipos, que resulta ser uno de los mayores desafíos en el desarrollo de software y otros proyectos complejos es la comunicación entre los diferentes miembros del equipo, los diagramas de clases y objetos actúan como un lenguaje visual común, ayudando a desarrolladores, diseñadores, analistas de negocio y otros actores del proyecto a comprender cómo se interrelacionan los elementos del sistema, reduciendo las ambigüedades y facilita la colaboración, incluso entre personas con

distintos niveles de experiencia técnica.

En proyectos a gran escala, como el desarrollo de sistemas empresariales o aplicaciones distribuidas, esta capacidad para representar visualmente la arquitectura del sistema es esencial para evitar malentendidos y garantizar que todas las partes involucradas compartan una visión unificada del producto.

Asimismo, los diagramas de clases contribuyen a una planificación y diseño eficientes del software, este paso es vital antes de escribir código, contar con un diseño bien planificado permite a los desarrolladores identificar relaciones clave como herencia, composición o asociación entre clases, al identificar estas relaciones tempranamente, es posible optimizar el código para mejorar la reutilización de componentes y evitar redundancias.

Además, estos diagramas permiten realizar cambios en el diseño a bajo costo, ya que es mucho más sencillo ajustar un esquema visual que modificar grandes cantidades de código, esta capacidad de planificación anticipada impacta directamente en la eficiencia y calidad del software, reduciendo la probabilidad de errores costosos y mejorando la flexibilidad del sistema.

A pesar de que los diagramas de clases y objetos están diseñados principalmente para el desarrollo de software, los principios subyacentes de abstracción, encapsulamiento y modularidad pueden aplicarse en la vida diaria para resolver problemas complejos, por ejemplo, si se enfrenta una tarea como organizar un evento, construir una casa o diseñar una campaña publicitaria, el uso de diagramas de clases puede ayudar a descomponer el proyecto en componentes individuales y definir las interacciones entre ellos, al pensar de manera estructurada y modelar los objetos que intervienen en un problema, se puede visualizar mejor la interconexión de cada elemento y encontrar soluciones más eficientes, siendo un enfoque analítico útil tanto en la toma de decisiones personales como profesionales.

Además, los diagramas de clases y objetos proporcionan una documentación clara y son clave para el mantenimiento a largo plazo, un aspecto crucial en la vida laboral de cualquier ingeniero de software o equipo de desarrollo, estos diagramas ofrecen una representación concisa del sistema que puede servir como referencia a lo largo de su ciclo de vida, esta documentación es especialmente valiosa durante el mantenimiento, ya que permite a los desarrolladores comprender rápidamente cómo se organiza el sistema sin tener que revisar grandes volúmenes de código, para organizaciones que trabajan con sistemas heredados o que frecuentemente deben realizar modificaciones, los diagramas de clases ofrecen una base sólida para la toma de decisiones, facilitando el análisis de impacto y la planificación de cambios de manera controlada y segura.

El estudio de diagramas de clases y objetos fomenta el desarrollo de habilidades cognitivas como el pensamiento abstracto, la resolución de problemas y el razonamiento lógico, son habilidades fundamentales en la vida diaria, ya que permiten abordar problemas desde una perspectiva más analítica y organizada, además, el modelado de clases promueve una comprensión profunda de cómo interactúan los diferentes elementos en un sistema, lo cual es útil para gestionar proyectos tanto tecnológicos como no tecnológicos.

Un aspecto crucial adicional es la capacidad de los diagramas de clases para representar sistemas que puedan escalar en tamaño y complejidad, al organizar la información de manera modular, es posible añadir nuevas funcionalidades o componentes al sistema sin necesidad de rediseñarlo completamente, esto es crucial para cualquier empresa que busque crecer o adaptarse rápidamente a nuevas demandas del mercado.

En conclusión, adquirir conocimiento sobre **Diagramas de Clases y Objetos** es una habilidad esencial que trasciende el ámbito técnico y en la vida laboral, proporciona una

herramienta indispensable para la planificación, desarrollo y mantenimiento de sistemas complejos, mejorando la comunicación, organización y eficiencia de los proyectos.

En la vida cotidiana, fomenta un enfoque analítico y estructurado para la resolución de problemas, lo que se traduce en una mayor capacidad para gestionar y ejecutar proyectos de manera efectiva, con esta base, los profesionales pueden enfrentar retos con mayor claridad y organización, asegurando un mejor control y adaptación en sus actividades diarias.

Finalmente, como dice **Grady Booch, uno de los creadores** de **UML:** "Un sistema bien estructurado es aquel que es resiliente al cambio y fácil de entender, el diseño orientado a objetos permite la creación de arquitecturas de software que son flexibles, reutilizables y escalables".

Referencias:

Admin. (2022a, noviembre 24). *¿Cuáles son los seis tipos de relaciones en los diagramas de clases UML? - Visual Paradigm Blog español*. Visual Paradigm Blog Español. https://blog.visual-paradigm.com/es/what-are-the-six-types-of-relationships-in-uml-class-diagrams/#Realizacion_Implementacion

Admin. (2022b, noviembre 24). *¿Cuáles son los seis tipos de relaciones en los diagramas de clases UML? - Visual Paradigm Blog español*. Visual Paradigm Blog Español. https://blog.visual-paradigm.com/es/what-are-the-six-types-of-relationships-in-uml-class-diagrams/#Realizacion_Implementacion

Walker, A. (2024, 24 febrero). *UML Relationships types: association, dependency, generalization*. Guru99. https://www.guru99.com/es/uml-relationships-with-example.html