UNIVERSIDAD DE LAS FUERZAS ARMADAS-ESPE

SEDE SANTO DOMINGO DE LOS TSÁCHILAS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN - DCCO-SS

CARRERA DE INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

PERIODO	:	202350 Mayo– Septiembre 2023
ASIGNATURA	:	Programación Orientada a Objetos
TEMA	:	Tarea 1
ESTUDIANTE	:	Bautista Bravo Dayanara Lisseth
NIVEL-PARALELO - NRC:		Primero A - Nrc 16129
DOCENTE	:	Ing. Verónica Martínez C., Mgs.
FECHA DE ENTREGA	:	15 de noviembre del 2023

SANTO DOMINGO – ECUADOR

Tabla de Contenido

Tabla de Ilustraciones

1. Introducción

El propósito de este documento es llevar a cabo una investigación exhaustiva que abarque diversos conceptos y definiciones relacionadas con la Programación Orientada a Objetos (POO). Entre estos conceptos se incluyen aspectos fundamentales como los Sistemas de Control de Versiones (SCV), paradigmas de programación, Entornos de Desarrollo, así como definiciones específicas dentro del ámbito de la POO, tales como clases, objetos, atributos, y el modelado de clases y objetos.

La Programación Orientada a Objetos (POO) se presenta como un paradigma de programación que proporciona pautas claras sobre cómo estructurar y trabajar con el código. Fundamentado en el concepto de clases y objetos, este enfoque se utiliza para organizar programas de software en módulos simples y reutilizables (clases), permitiendo la creación de instancias individuales de objetos. (Canelo, 2020)

Los Sistemas de Control de Versiones (SCV) son herramientas que almacenan las diferentes versiones de un software a lo largo de su desarrollo. Estos sistemas gestionan los diversos estados por los que pasa una aplicación, manteniendo un historial de todos los cambios realizados entre versiones. (Leal, R., & Leal, 2012)

Este documento se estructura en varias secciones, que incluyen los objetivos del trabajo, un marco teórico donde se exploran diversas definiciones sobre los temas mencionados, un apartado de desarrollo que contiene ilustraciones, así como ejemplos prácticos relacionados con la POO y UML.

2. Objetivos

2.1 Objetivo General:

Investigar y proporcionar una visión integral de conceptos esenciales en la Programación Orientada a objetos.

2.2 Objetivos Específicos:

- Investigar y explicar los principios fundamentales de los sistemas de control de versiones, destacando su importancia en el desarrollo colaborativo de software.
- Definir y explicar los conceptos fundamentales relacionados con el modelado de clases y objetos en el contexto de la programación orientada a objetos.
- Presentar ejemplos prácticos y aplicados para ilustrar la implementación de los conceptos investigados

3. Marco Teórico/ Desarrollo/ Práctica

3.1 Marco Teórico

3.1.1 Sistemas de control de versionamiento (VCS)

Los sistemas de control de versiones (VCS) son herramientas que permiten gestionar cambios en el código fuente u otros conjuntos de archivos a lo largo del tiempo. Estos sistemas rastrean los cambios, facilitan la colaboración entre desarrolladores y ayudan a mantener un historial completo de los cambios realizados en un proyecto. El concepto de sistemas de control de versiones ha ido evolucionando con el tiempo y no tiene un único creador. Sin embargo, uno de los primeros sistemas de control de versiones ampliamente utilizados fue el Sistema de control de versiones concurrente (CVS), desarrollado por Dick Gruen en la década de 1980. (Piedra, Santos, & Santos, 2022)

Los sistemas de control de versiones se crearon para resolver los problemas asociados con el desarrollo colaborativo de software y la gestión de cambios en el código fuente. Permiten a los equipos de desarrollo trabajar de manera más eficiente, colaborar sin problemas, realizar un seguimiento de los cambios realizados en el código y volver a versiones anteriores cuando sea necesario. También son útiles para la gestión de proyectos y la documentación de cambios.

El uso de sistemas de control de versiones resuelve varios problemas comunes en el desarrollo de software, como la pérdida de código debido a cambios no deseados, la falta de colaboración efectiva entre desarrolladores y la dificultad para mantener un historial completo de cambios en un proyecto. Estos sistemas proporcionan una estructura organizada para el desarrollo colaborativo y la gestión de lanzamientos. (Platziteam, 2019)

3.1.2 Software VCS: Git, GitHub

Git es un proyecto de código abierto maduro y con un mantenimiento activo que desarrolló originalmente Linus Torvalds, el famoso creador del kernel del sistema operativo Linux, en 2005.

Linus Torvalds diseñó Git como un sistema de control de versiones distribuido, lo que significa que cada desarrollador tiene una copia completa del historial de cambios del proyecto en su máquina local. Esto permite un rendimiento rápido y una mayor flexibilidad en términos de colaboración y ramificación.

Git se creó para gestionar eficientemente el desarrollo del kernel de Linux, pero su diseño flexible y distribuido lo ha convertido en una herramienta ampliamente utilizada para controlar versiones en una variedad de proyectos de software. Git se utiliza para realizar un seguimiento de los cambios en el código fuente, facilitar la colaboración entre desarrolladores y permitir la gestión de versiones de manera eficiente.

La gran plataforma GitHub fue creada por Chris Wanstrath, PJ Hyett, Tom Preston-Werner y Scott Chacon en febrero de 2008 en San Francisco, EE. UU. y permanece en el mismo lugar hasta el día de hoy. (Colectiva, 2023)

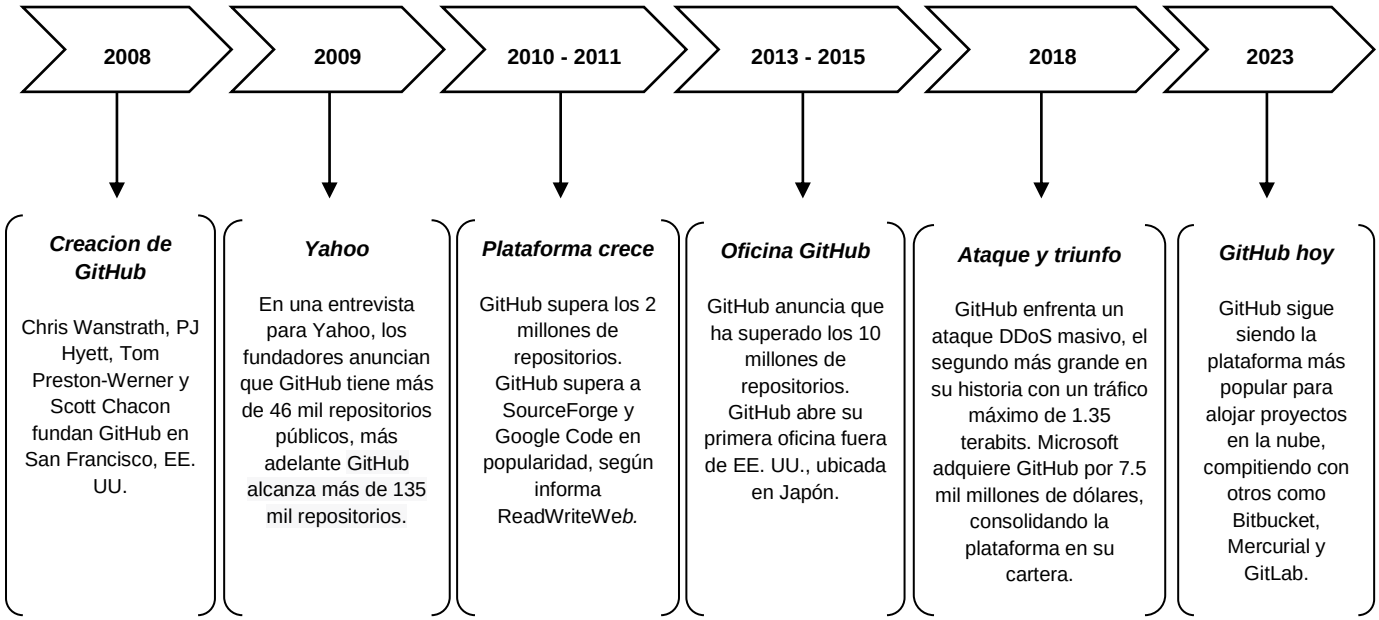


Ilustración 1. Línea de tiempo GitHub

GitHub se diseñó para facilitar la colaboración entre desarrolladores al proporcionar un espacio centralizado para alojar y gestionar proyectos, permitiendo un seguimiento eficiente de cambios en el código fuente y fomentando el desarrollo de software colaborativo. La plataforma ha evolucionado para convertirse en un componente esencial en el ecosistema de desarrollo de software, ofreciendo herramientas para el control de versiones, la colaboración, la revisión de código y la gestión de proyectos.

3.1.2.1 Repositorios en GitHub

Un repositorio es el elemento más básico de GitHub. Es un lugar donde se puede almacenar códigos, archivos y el historial de revisiones de cada archivo. Los repositorios pueden contar con múltiples colaboradores y pueden ser públicos como privados. Los repositorios en las organizaciones que utilizan GitHub Enterprise Cloud y le pertenecen a una cuenta empresarial también pueden crearse con visibilidad interna.

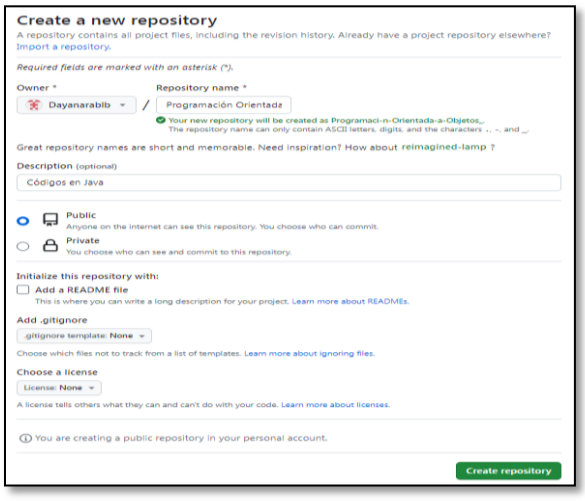


Ilustración 2. Creacion de repositorio público

Los archivos que agregues a un repositorio mediante un navegador están limitados a 25 MiB por archivo. Se Puede agregar archivos más grandes, de hasta 100 MiB cada uno, mediante la línea de comando. (GitHub, 2023) En GitHub se puede subir y gestionar una variedad de tipos de archivos, la plataforma está diseñada principalmente para el control de versiones de código fuente, pero admite muchos tipos de archivos, incluyendo:

- Código Fuente: Archivos de código en diversos lenguajes de programación como Java, Python, JavaScript, C++, entre otros.
- Documentación: Archivos Markdown, archivos de texto, documentos en formato PDF, y otros formatos de documentación.
- Imágenes y Gráficos: Archivos de imágenes en formatos como JPG, PNG, GIF, SVG, etc.
- Archivos de Configuración: Archivos de configuración del proyecto, como archivos YAML, JSON, XML, etc.
- Archivos de Datos: Conjuntos de datos en formatos como CSV, JSON, XML, entre otros.
- Archivos de Estilo y Plantillas: Archivos de hojas de estilo (CSS), plantillas HTML, y otros relacionados con la presentación.
- Archivos Binarios: Aunque GitHub está optimizado para archivos de texto, también puede gestionar archivos binarios como imágenes, archivos de audio, archivos ejecutables, etc.
- Archivos de Proyectos: Archivos específicos del proyecto, como archivos de configuración de compilación, scripts de automatización, etc.

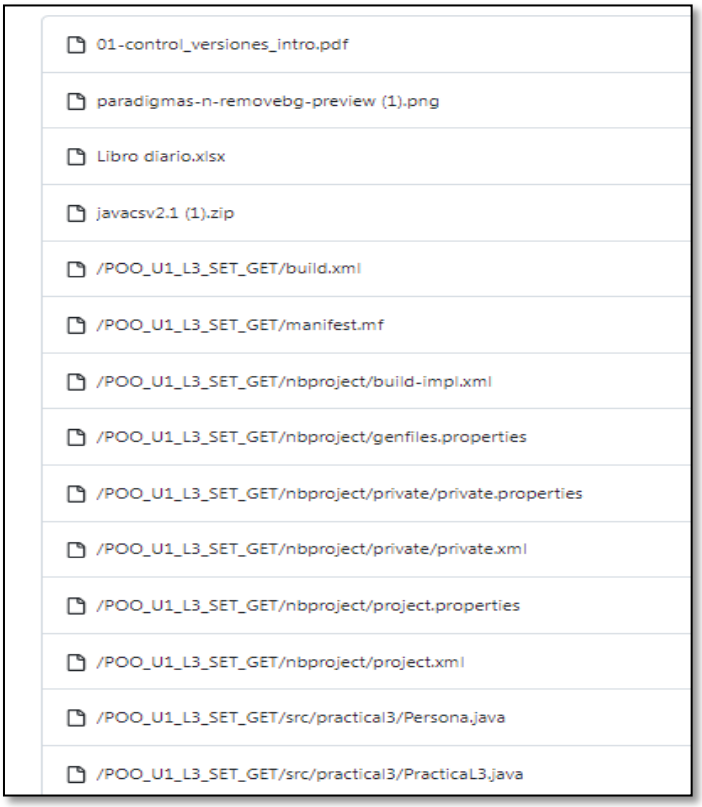


Ilustración 3. Tipos de Archivos que se pueden subir en GitHub

3.1.3 Paradigmas de Programación

Los paradigmas de la programación son estilos que se siguen a la hora de programar un software. Como estilo podemos referirnos a muchas cosas, por ejemplo las estructuras de control que vengamos utilizando, o la manera de estructurar los programas, incluso las piezas elementales que tendrán éstos.

A lo largo de la historia del software se han incorporado diferentes paradigmas que iban agregando mejores posibilidades de organización del código, aportando mayor claridad y la posibilidad de crear el software componiéndolo con piezas más pequeñas y por lo tanto menos complejas. Los lenguajes de programación a lo largo del tiempo han nacido con el paradigma que estaba más establecido en el momento de su creación, aunque muchos han ido evolucionando e incorporando la posibilidad que los programadores usasen otros estilos o paradigmas de la programación si lo deseaban. Es así como actualmente la mayoría de los lenguajes implementan diversos tipos de programación, lo que se conoce a veces como multi-paradigma. (Rodriguez, 2011)

Los tipos o técnicas de programación son bastante variados, aunque puede que muchos de los lectores sólo conozcan un método para realizar los programas. En la mayoría de los casos, las técnicas se centran en programación modular y programación estructurada, pero existen otros tipos de programación.

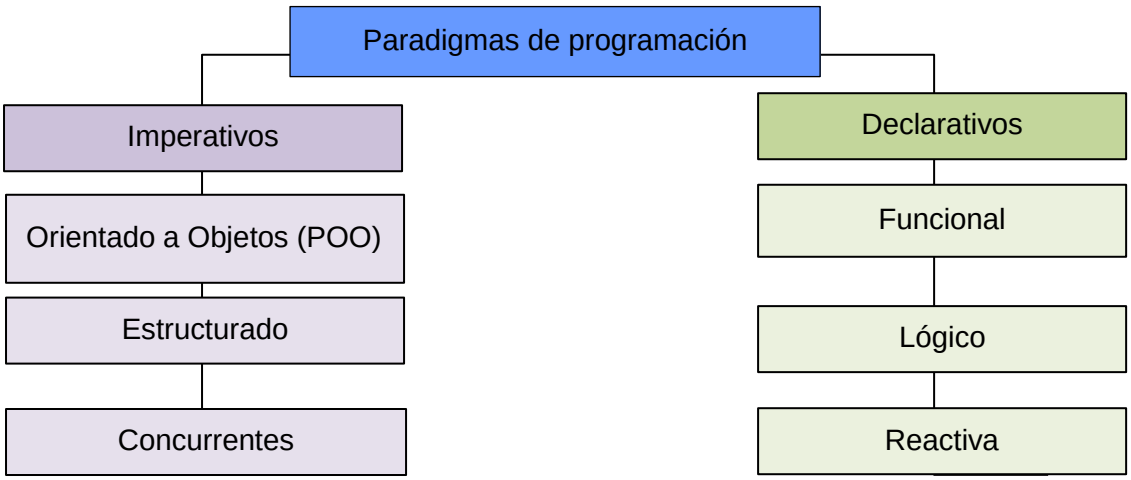


Ilustración 4. Paradigmas de Programación

3.1.3.1 Paradigma Imperativo

En programación, el paradigma imperativo se centra en explicar paso a paso cómo lograr un resultado. En este enfoque, se envían instrucciones a la computadora sobre cómo realizar una tarea específica, demostrando el flujo de control y manipulando el estado del programa mediante cambios directos en las variables y la memoria. Las declaraciones imperativas suelen consistir en asignaciones, bucles y condicionales. Este paradigma es más similar a la arquitectura de máquinas y se centra en la ejecución de acciones. (Palacios, 2021)

Orientado a Objetos: El paradigma orientado a objetos se basa en la idea de que todo es un objeto, así como todo lo que nos rodea en el mundo real es un objeto. En este paradigma, una computadora, una persona o incluso el aire “todo se comporta como un objeto”. Estos objetos a su vez tienen “atributos”, que son básicamente los diferenciadores que tienen estos objetos de otros objetos, tal como son; su color, altura o espesor.

Estructurado: La programación estructurada es un paradigma de programación orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa utilizando únicamente subrutinas o funciones y tres estructuras: secuencial, condicional y repetitiva.

Estructuras de Control

- Estructura secuencial. Está formada por una secuencia de llamadas a instrucciones del lenguaje o funciones del programador.
- Estructura condicional. Es aquella que ejecuta una estructura si se cumple una condición booleana.
- Estructura iterativa con condición. Es aquella que ejecuta una estructura una y otra vez si se cumple una condición booleana.

Concurrentes: El paradigma concurrente en programación se refiere al diseño y la implementación de sistemas que pueden ejecutar múltiples tareas de manera simultánea. La concurrencia se refiere a la ejecución de varios procesos de manera independiente, donde cada uno avanza en su propio ritmo.

3.1.3.2 Paradigma Declarativo

El paradigma declarativo se centra en describir lo que se quiere lograr sin detallar los pasos exactos para lograr ese resultado. En lugar de dar instrucciones específicas a una computadora, los programadores declaran propiedades, relaciones o restricciones que deben cumplirse. Este enfoque intenta abstraer la lógica subyacente y simplificar la comprensión del problema. (Palacios, 2021)

Funcional: En el paradigma funcional, la programación se basa en el concepto de funciones puramente matemáticas. Se centra en la evaluación de expresiones y evita cambios de estado y variables mutables. Las funciones en este paradigma no tienen efectos secundarios, lo que significa que siempre producen el mismo resultado para una entrada determinada.

Reactivo: En el paradigma reactivo, el énfasis está en la propagación automática del cambio. Se utiliza en situaciones en las que los datos pueden cambiar con el tiempo y usted necesita responder a estos cambios de manera eficiente. Se usa comúnmente en interfaces de usuario, sistemas de eventos y aplicaciones que requieren actualizaciones en tiempo real.

Lógico: Se basa en la lógica matemática y las reglas formales. En este enfoque, el programador declara relaciones y hechos lógicos y el sistema de ejecución obtiene las respuestas mediante inferencia lógica.

3.1.4 Entorno de Desarrollo

Un entorno de desarrollo es un conjunto de procedimientos y herramientas que se utilizan para desarrollar código fuente o programas. Un entorno de desarrollo suele tener tres niveles de servidores, clasificados como desarrollo, ensayo y producción. Estos tres niveles generalmente se denominan DSP.

El servidor de desarrollo es en donde los desarrolladores prueban el código y verifican si la aplicación se ejecuta correctamente con el código. Una vez que se prueba la implementación y el desarrollador considera que el código funciona correctamente, la aplicación se mueve al servidor de prueba. Un servidor de integración es un entorno que se crea para verse exactamente igual que un entorno de servidor de producción. La aplicación se prueba en el servidor de prueba para determinar su confiabilidad y garantizar que la aplicación no falle en el servidor de producción real. Este tipo de prueba en un servidor de prueba es el paso final antes de implementar una aplicación en un servidor de producción. Una vez que se completa la aprobación la aplicación pasa a formar parte de este servidor. Algunos ejemplos de entornos de desarrollo integrados populares son NetBeans, Microsoft Visual Studio, Adobe Flex Builder y Eclipse. (Arimetrics, s/f)

3.1.5 Características e Instalación

Para la correcta instalación de los entornos de desarrollo integrados (IDE) NetBeans, Visual Studio Code y Eclipse, se necesita un equipo que cumpla con ciertos requisitos mínimos.

Se recomienda un procesador mínimo de 2 GHz para un rendimiento óptimo, pero se prefiere un procesador más rápido.Tener al menos 4 GB de RAM para un rendimiento adecuado. Sin embargo, sería beneficioso tener más memoria, especialmente cuando se trabaja en proyectos grandes.

Cada IDE y su instalación serán diferentes, pero se debe tener al menos 2 GB de espacio libre en disco para cada IDE, así como espacio adicional para el proyecto que estás creando. Verificar la compatibilidad de cada IDE con su sistema operativo. Este IDE es multiplataforma, pero asegurarse de descargar la versión adecuada para su sistema operativo (Windows, macOS o Linux).

Requisitos específicos para cada IDE:

NetBeans: Antes de instalar NetBeans, debe instalar JDK en su sistema, porque NetBeans es una plataforma Java.

Visual Studio Code: Asegurarse de tener un sistema operativo que admita Visual Studio Code. Funciona en Windows, macOS y Linux. Algunas extensiones o funciones pueden requerir la instalación de JRE.

Eclipse: Eclipse también es una aplicación Java, por lo que necesita instalar el JDK en su sistema. Asegúrese de tener una conexión a Internet estable para descargar e instalar este IDE, así como actualizaciones y extensiones.

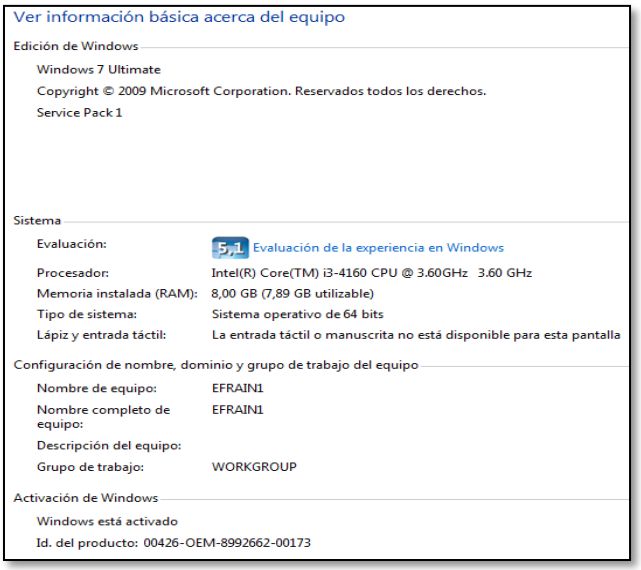


Ilustración 5. Propiedades del Sistema

3.1.6 Revisión de conceptos generales de la POO

La programación orientada a objetos es un paradigma de programación que se centra en crear objetos e interactuar con ellos para resolver problemas de software. En POO, los objetos representan entidades del mundo real, como personas, animales o vehículos, y están organizados en clases que definen sus propiedades y comportamientos.

3.1.7 Principios Básicos de la Programación Orientada a Objetos

Encapsulación: La encapsulación le permite restringir el acceso a los datos y la información de los objetos, protegerlos y ocultar detalles de los datos en sí.

Herencia: La herencia en programación orientada a objetos permite definir una jerarquía entre clases y compartir atributos y métodos comunes que se pueden reutilizar.

Polimorfismo: El polimorfismo nos permite diseñar objetos que pueden exhibir diferentes comportamientos, lo que nos permite procesar objetos de diferentes maneras.

Abstracción: La abstracción se basa en extraer información esencial de objetos y cosas simples para representar la complejidad. Por lo tanto, estos objetos, que forman parte de un sistema, representan el código subyacente y ocultan detalles complejos al usuario. (Cuenca, 2021)

3.1.8 Definición de Clases, Objetos, Atributos y Métodos

Clases

Las clases son uno de los subconjuntos de los objetos y son uno de los elementos más importantes para programar en este lenguaje. Estos elementos sirven para crear plantillas que pueden ser replicados para categorizar objetos con atributos similares. (Durán, 2023)

Tipos de clases en Java. Los tipos de clases en Java son los siguientes:

- Públicas
- Privadas
- Finales
- Abstractas

Objetos

Un objeto en Java es una entidad que representa información sobre un objeto en el código de un programa. Así, los objetos en este lenguaje son instancias o miembros de una clase particular que tienen propiedades, atributos y características que los distinguen de otros, al igual que los objetos del mundo real. (Durán, 2023)

Atributos

Los atributos son características individuales que distinguen un objeto de otro y determinan su apariencia, condición u otras cualidades. Los atributos se almacenan en variables llamadas instancias y cada objeto específico puede tener valores diferentes para estas variables. (Durán, 2023)

Métodos

Un método en Java es un conjunto de instrucciones definidas dentro de una clase, que realizan una determinada tarea y a las que podemos invocar mediante un nombre. Un método es una abstracción de una operación que puede hacer o realizarse con un objeto. Una clase puede declarar cualquier número de métodos que lleven a cabo operaciones de lo más variado con los objetos. En esta sección los métodos se clasifican en dos grupos: los métodos de instancia y los métodos de clase. (Facil, 20203)

3.1.9 Modelamiento de Clases y Objetos

Modelar clases y objetos es un paso crítico en el desarrollo de software orientado a objetos. Este proceso implica representar visual y conceptualmente entidades y sus interacciones en un sistema. Las clases y los objetos son conceptos fundamentales en la programación orientada a objetos (POO). Las clases son plantillas para crear objetos que son instancias concretas de esas clases. (Peñalvo & Aguilar, 2022) El modelado ayuda a comprender la estructura de un sistema, las relaciones entre entidades y su comportamiento.

3.1.10 UML Diagramas de Casos de Usos

Los diagramas de casos de uso son una parte importante del modelado de sistemas en UML (lenguaje de modelado unificado). Estos diagramas representan las interacciones entre el sistema y los actores externos, mostrando cómo responde el sistema a ciertos estímulos. Los casos de uso describen funciones específicas del sistema desde el punto de vista del usuario. (Grau & Segura, 2011)

3.1.11 UML Diagramas de Clases

Los diagramas de clases en UML proporcionan una representación estática de la estructura de un sistema, mostrando clases, sus atributos, métodos y relaciones entre ellos. Estos diagramas son esenciales para comprender la organización y arquitectura del sistema. (Grau & Segura, 2011)

3.1.12 Identificación de Clases de un Sistema, Uso Correcto de Identificadores.

La identificación de clases implica reconocer objetos relevantes en el sistema y darles nombres significativos. El uso adecuado de identificadores es fundamental para la claridad y comprensibilidad del código. Por ejemplo, en un sistema bancario, las clases pueden identificarse como Cuenta bancaria, Cliente y Transacción, utilizando nombres descriptivos que reflejen su función en el sistema. (Programacion en Java, s/f)

3.1.13 Modificadores de Acceso

Los modificadores de acceso en la programación orientada a objetos determinan la visibilidad y el alcance de las clases, métodos y atributos. Los principales modificadores son "públicos", “privados” y “protegidos". (Mundogeek, 2009) Por ejemplo, si en un sistema de comercio electrónico queremos ocultar ciertos detalles de implementación de la clase Producto, podemos declarar sus atributos como “privados” para limitar el acceso externo directo.

3.1.14 Implementación de clases

La implementación de clases implica traducir el diseño conceptual de clases y objetos en código concreto. Esto incluye definir atributos, métodos y gestionar relaciones entre clases. En el sistema de reservas de una aerolínea, la implementación de la clase de vuelo puede incluir métodos para reservar asientos, calcular tarifas y gestionar la disponibilidad de vuelos. La implementación garantiza que el diseño conceptual se traduzca exitosamente en una funcionalidad ejecutable. (IBM, 2021)

4. Desarrollo

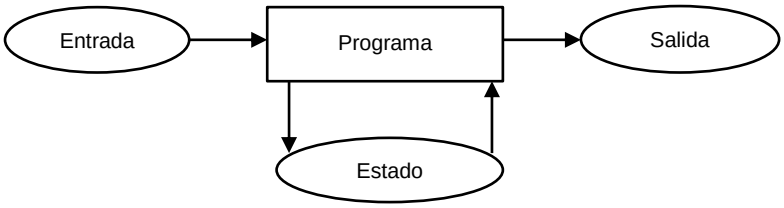


Ilustración 6. Modelo Computacional del paradigma Imperativo

El modelo informático del paradigma imperativo se basa en la idea de que un programa es una serie de comandos que cambian el estado de un sistema. El control de flujo se logra mediante instrucciones secuenciales y estructuras de flujo de control, y las variables y asignaciones son fundamentales para gestionar el estado del programa.

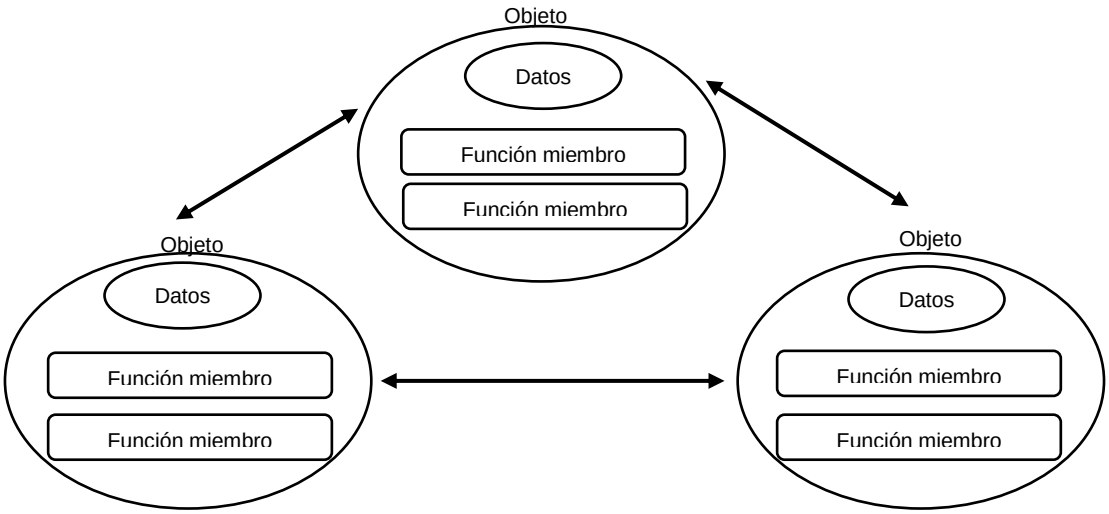
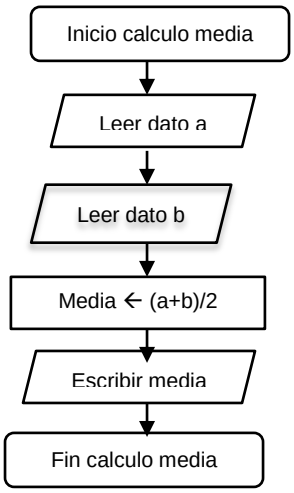


Ilustración 7. Paradigma Orientado a Objetos

La idea fundamental de la orientación a objetos y de los lenguajes que implementan este paradigma de programación, es combinar (encapsular) en una sola unidad tanto los datos como las funciones que operan (manipulan) sobre los datos. Esta característica permite modelar los objetos del mundo real de un modo mucho más eficiente que con funciones y datos. Esta unidad de programación se denomina objeto.



La estructura secuencial en programación imperativa implica la ejecución lineal y secuencial de instrucciones, una tras otra, sin bifurcaciones ni repeticiones. Cada instrucción se completa antes de pasar a la siguiente, siguiendo un flujo de control predecible. Esta estructura define un punto de inicio y fin claro en el código.

Ilustración 8. Ejemplo de Estructura secuencial

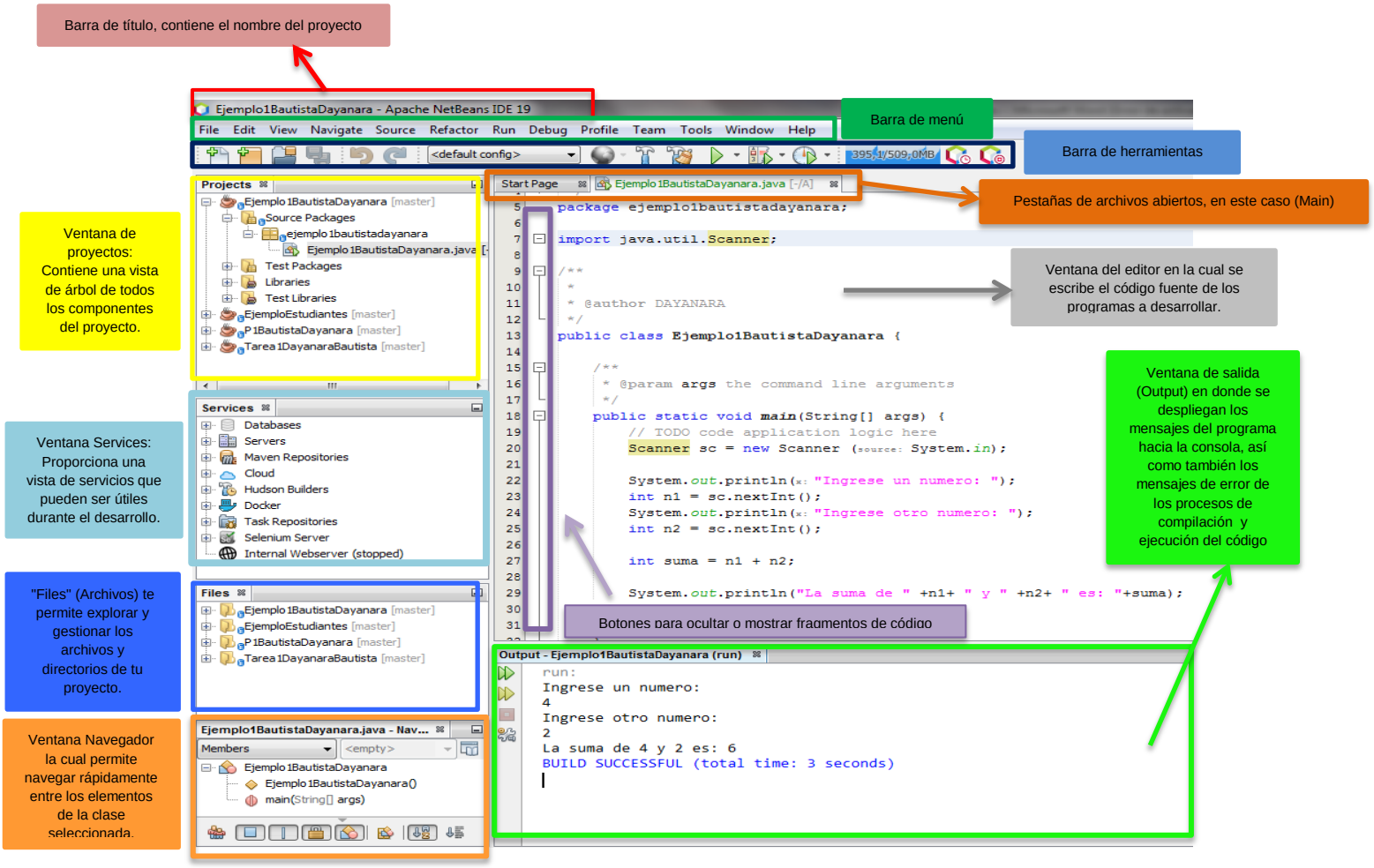


Ilustración 9. Área de trabajo en Netbeans

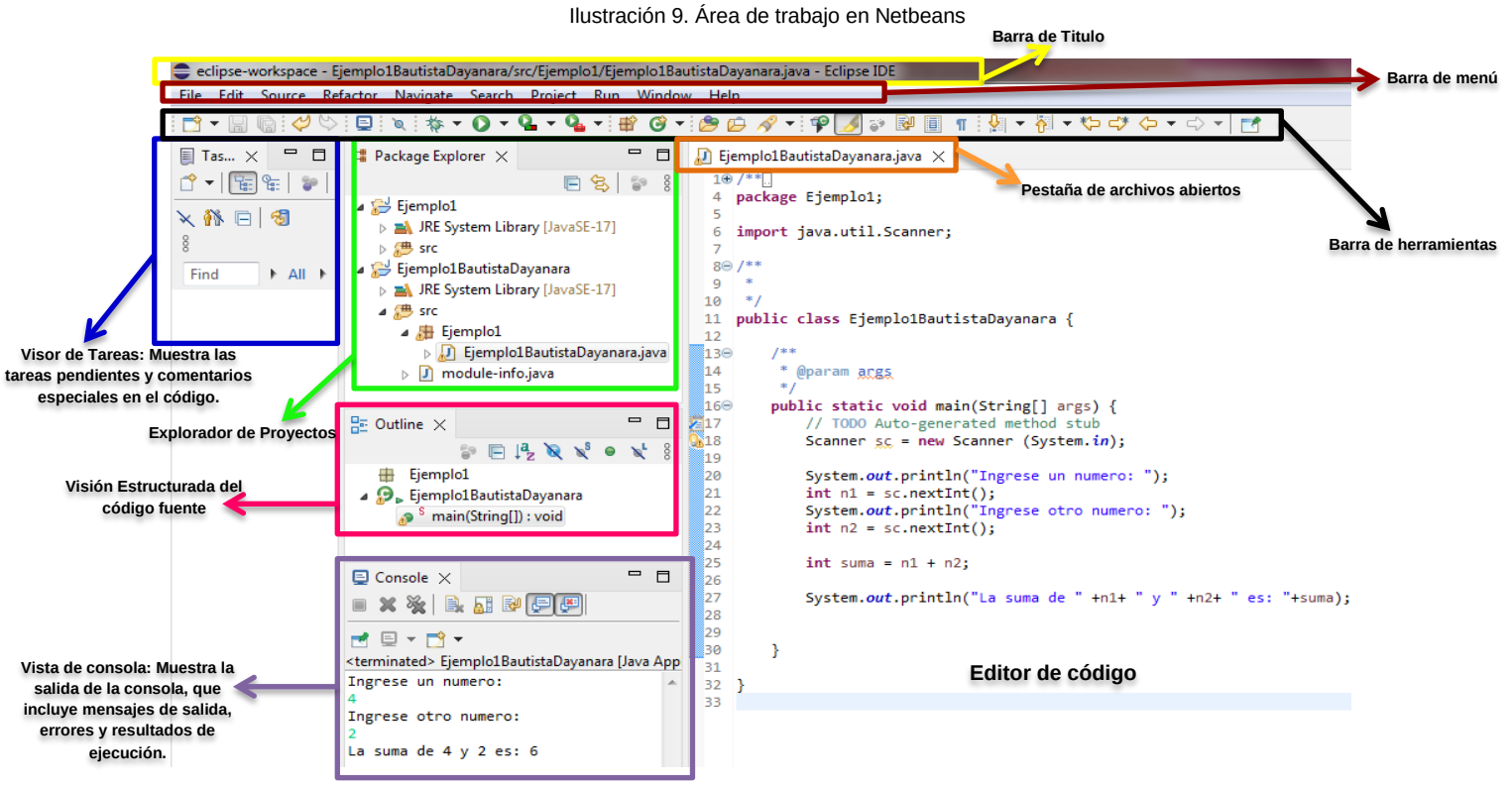


Ilustración 10. Área de Trabajo en Eclipse

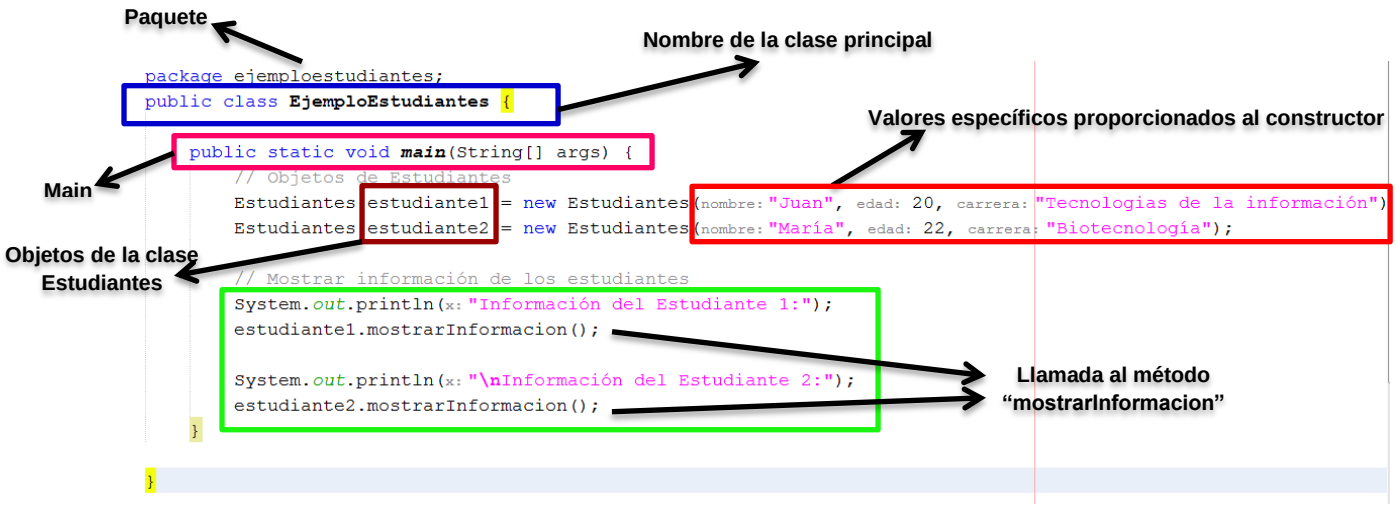


Ilustración 11. Ejemplo Clase principal (Objetos)

La clase Estudiantes tiene atributos que son "nombre", "edad" y "carrera", después tenemos un constructor de la clase Estudiantes toma tres parámetros "nombre, edad, carrera) en donde se los utiliza para inicializar los atributos de los objetos de la clase Estudiante. La clase Estudiantes tiene el método mostrar información la cual aparecerá al momento de compilar el código, en la clase "EjemploEstudiante" se crean dos objetos de la clase Estudiante (estudiante1, estudiante2) y se llama al método "mostrarInformacion".

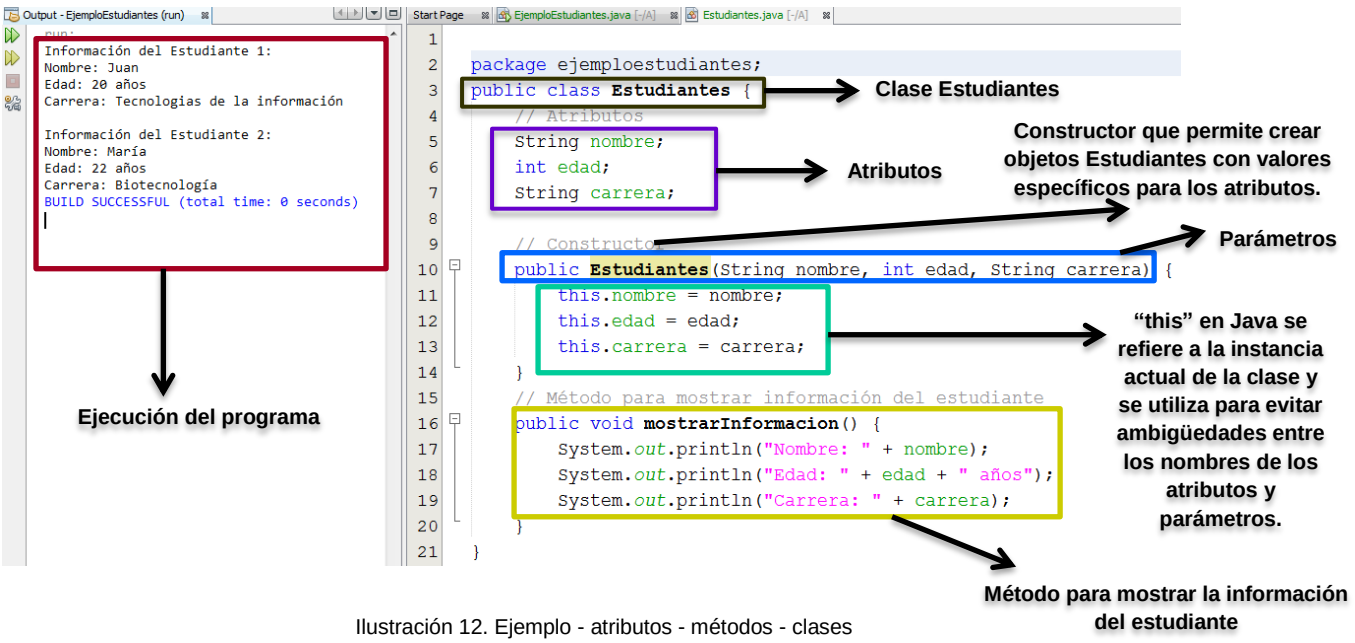


Ilustración 12. Ejemplo - atributos - métodos - clases

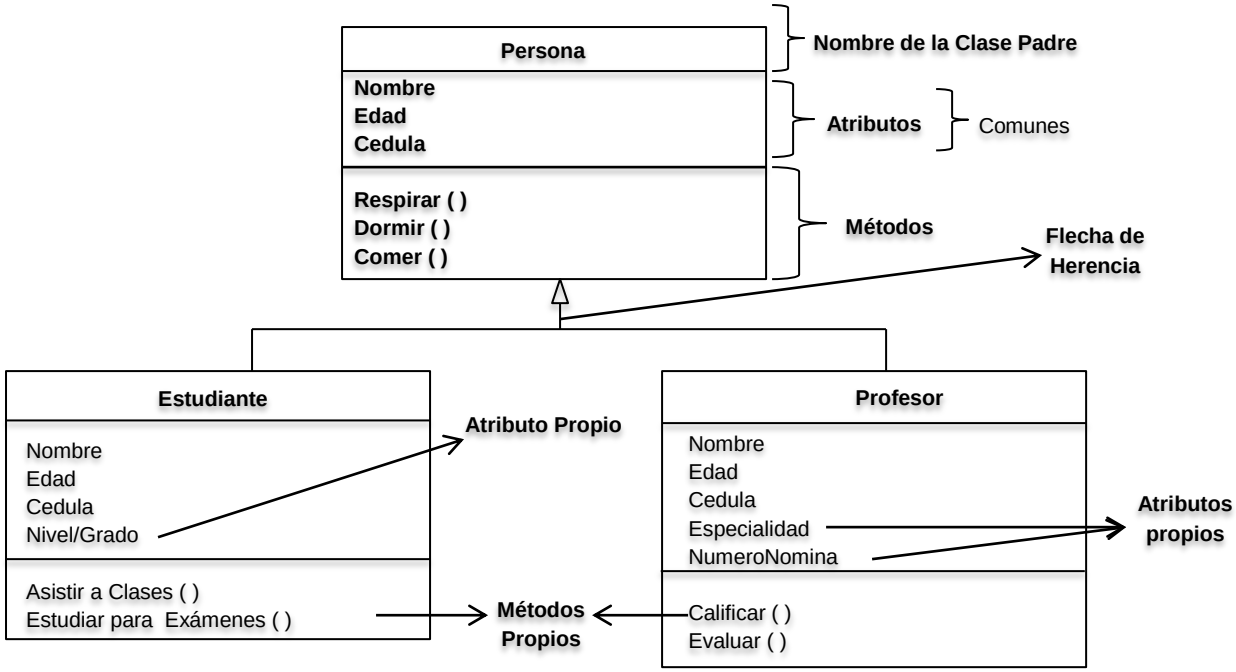


Ilustración 13. Ejemplo de Diagrama de clases -UML

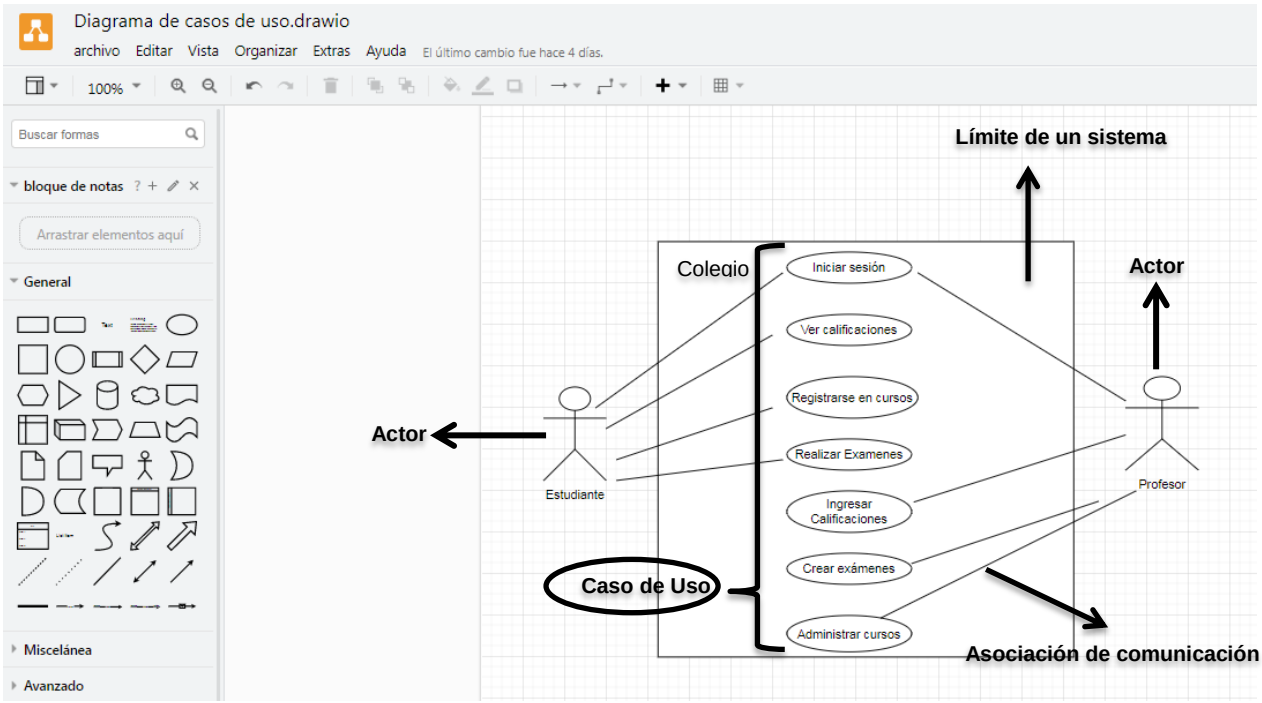


Ilustración 14. Caso de uso en software Draw.io

En el diagrama de casos de uso tenemos dos actores que son Estudiante y Profesor y los casos de uso para el estudiante son iniciar sesión, ver calificaciones, registrarse en cursos y realizar exámenes, para el profesor iniciar sesión, ingresar calificaciones, crear exámenes y administrar cursos, la relación entre estos dos actores es que ambos iniciar sesión para ingresar al sistema.

5. Conclusiones

- La programación orientada a objetos (POO) proporciona un enfoque estructurado y modular para el desarrollo de software, lo que facilita la creación de sistemas más mantenibles y reutilizables.

- Los sistemas de control de versiones (VCS) como Git y GitHub son esenciales para rastrear y gestionar eficazmente los cambios en el código fuente, mejorando la colaboración dentro de los equipos de desarrollo.

- UML es una herramienta poderosa para visualizar y comunicar la estructura y el diseño de sistemas de software complejos, facilitando la comprensión entre los miembros del equipo y las partes interesadas.

6. Recomendaciones

- Es fundamental integrar y utilizar sistemas de control de versiones en proyectos de software. Se recomienda que se familiarice con los comandos básicos de Git y utilice plataformas como GitHub para la colaboración remota.

- Se recomienda comprender a fondo los principios de la programación orientada a objetos y practicar su aplicación en proyectos del mundo real para maximizar los beneficios de la modularidad y la reutilización del código.

7. Bibliografía/ Referencias

Colectiva, N. (2023). Que es GitHub, Historia y otros Detalles. Obtenido de https://blog.nubecolectiva.com/que-es-github-historia-y-otros-detalles/

GitHub. (2023). Agregar un archivo a un repositorio. Obtenido de https://docs.github.com/es/repositories/working-with-files/managing-files/adding-a-file-to-a-repository

Palacios, D. (20 de Febrero de 2021). Styde. Obtenido de https://styde.net/programacion-declarativa-vs-imperativa/

Piedra, N. A., Santos, J. A., & Santos, A. M. (2022). La Importancia de los Sistemas de Control de Versiones en La Gestión de Liberación de Sistemas Web. SCIÉNDO, 333 - 340.

Platziteam. (2019). Platzi. Obtenido de https://platzi.com/blog/github-vs-gitlab/

Arimetrics. (s/f). Entorno de Desarrollo. Obtenido de https://www.arimetrics.com/glosario-digital/entorno-de-desarrollo

Canelo, M. M. (02 de Noviembre de 2020). Profile. Obtenido de https://profile.es/blog/que-es-la-programacion-orientada-a-objetos/

Cuenca, J. (2021). Entendiendo los principios de la Programación orientada a objetos. Obtenido de https://joseantoniocuenca.com/entendiendo-los-principios-de-la-programacion-orientada-a-objetos

Durán, M. (16 de mayo de 2023). Clases en Java. Obtenido de https://blog.hubspot.es/website/que-es-clase-en-java

Facil, P. (13 de Enero de 20203). Los métodos de clase en Java. Obtenido de https://programacionfacil.org/blog/los-metodos-de-clase-en-java/#google_vignette

Grau, X. F., & Segura, M. I. (2011). Desarrollo Orientado a Objetos con UML. Obtenido de https://www.uv.mx/personal/maymendez/files/2011/05/umltotal.pdf

IBM. (2021). Implementacion de clases Java. Obtenido de https://www.ibm.com/docs/es/mfci/7.6.2?topic=implementation-java-class

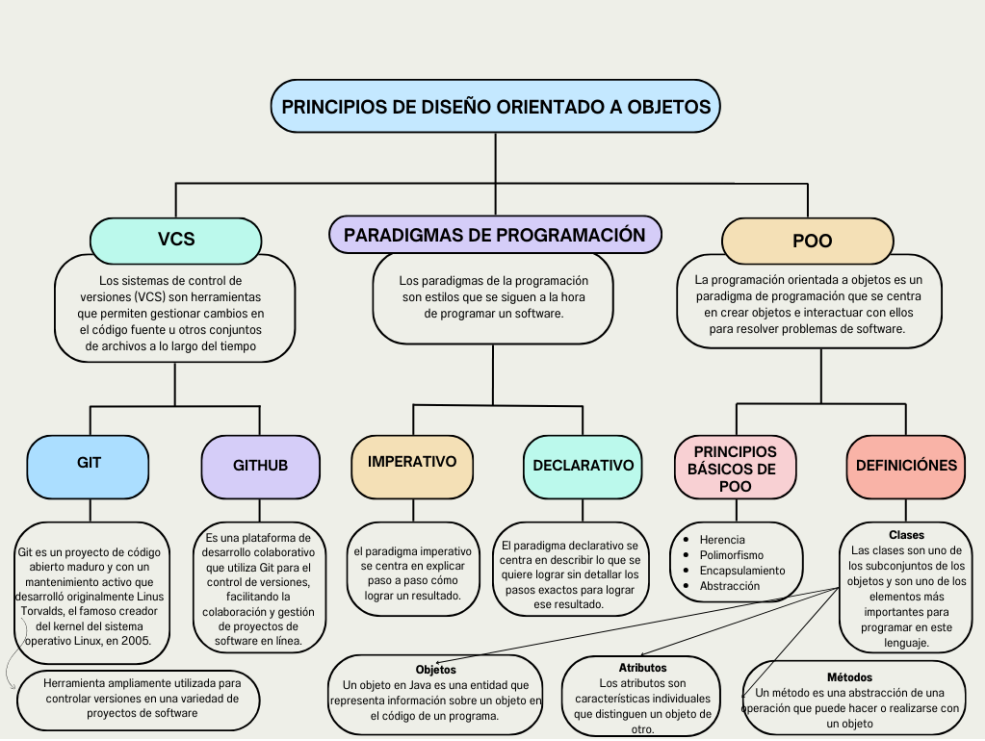
Leal, E. T., R., C. M., & Leal, D. A. (2012). Revisión de los sistemas de control de versiones utilizados en el desarrollo de software. Dialnet, 75-76.

Mundogeek. (30 de Marzo de 2009). Modificadores en Java. Obtenido de http://mundogeek.net/archivos/2009/03/30/modificadores-en-java/

Peñalvo, F. J., & Aguilar, C. P. (2022). Diagramas de Clase en UML. Obtenido de https://repositorio.grial.eu/bitstream/grial/353/1/DClase.pdf

Rodriguez, C. V. (2011). Paradigmas de Programacion. Obtenido de https://www.slideserve.com/adolph/paradigmas-de-programaci-n#google_vignette

8. Anexos



Anexo 1: Mapa Conceptual realizado en canva

Enlace al mapa conceptual: https://www.canva.com/design/DAF0S4BI77c/KZ6OxsQtuUkfCq-cWbfxWQ/edit?utm_content=DAF0S4BI77c&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

9. Legalización de documento

Nombres y Apellidos: Bautista Bravo Dayanara Lisseth

CI: 2350083057

Firma: Imagen de su firma