

FACULTAD DE INGENIERÍA

CARRERA DE INGENIERÍA DE SISTEMAS COMPUTACIONALES

INFORME PROYECTO

**Autores:**

Ramos Zevallos, Benjamin Franklin

Azaña Tejada Marcial-

Tarazona Mallqui Anibal-

Castillo Tineo Santiago Junior-

Campoverde Martínez Martin Fabricio-

**Curso:**

Técnicas de Programación Orientada a Objetos

**Docente del Curso:**

Jorge Chicana Aspajo

Esther Tarmeño Juscamaita

Trujillo – Perú

2025-

Forma

El contenido generado por IA puede ser incorrecto.Desarrollar guía de primeros pasos con Git

Crear repositorio local

Para iniciar un nuevo proyecto versionado con Git, se debe crear una carpeta en el sistema de archivos y luego inicializar un repositorio en su interior.

Texto

El contenido generado por IA puede ser incorrecto.

clonar un repositorio de GitHub a la  
máquina local

Texto

El contenido generado por IA puede ser incorrecto.Si se desea trabajar con un repositorio ya existente en GitHub, es posible clonar su contenido:

Este comando descarga el repositorio remoto en la computadora y crea una carpeta con el mismo nombre.

Navegación básica por el repositorio

Git ofrece comandos para revisar el estado del proyecto y la rama actual:

Ver el estado de los archivos

git status

Conocer en qué rama se está trabajando

git branch

Consultar el historial resumido de commits

git log –oneline

Realizar cambios en el código:

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Utilizar comandos git add y git commit

Si se realizan modificaciones en los archivos, se deben volver a añadir y confirmar:

Ver cambios pendientes

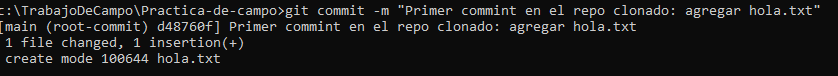
Texto

El contenido generado por IA puede ser incorrecto.git status

Añadir el archivo al área de preparación

Por ejemplo, si tienes un archivo hola.txt:

Hacer el commit

Con un mensaje que explique lo que hiciste:

**Visualizar historial de commits**

Ver todos los commits hechos en el repositorio:

git log

Texto

El contenido generado por IA puede ser incorrecto.

**Manejo de branches y merges**

Creamos una rama:



Ingresamos a la rama:

Una captura de pantalla de un celular con texto e imagen

El contenido generado por IA puede ser incorrecto.

Realizar cambios en una rama

Texto

El contenido generado por IA puede ser incorrecto.En la rama secundaria (trabajo2), se pueden realizar modificaciones como crear o editar archivos.  
Ejemplo:

**Uso de git merge:**

El comando git merge permite **combinar ramas**. Es el mecanismo más común para integrar cambios desarrollados en ramas secundarias hacia la rama principal.  
Ejemplo:

Fusión de ramas (merge)

Una vez realizados los cambios en la rama secundaria, se puede regresar a la rama principal y fusionar los cambios:

Imagen que contiene naranja, cerca, firmar, hombre

El contenido generado por IA puede ser incorrecto.

Fusionar los cambios:

Cambiamos a la rama principal

Texto

El contenido generado por IA puede ser incorrecto.

Si no hay conflictos, la fusión se realiza automáticamente y los cambios de desarrollo se incorporan en main.

**Resolución de conflictos**

En algunos casos, los mismos archivos pueden haber sido modificados en ambas ramas, lo que genera un conflicto. Git marcará el archivo afectado con delimitadores

El usuario debe editar el archivo, decidir qué contenido conservar y luego ejecutar:

Imagen de la pantalla de un video juego

El contenido generado por IA puede ser incorrecto.

**Conclusión**

El manejo de ramas y fusiones en Git es esencial para trabajar de forma ordenada y colaborativa. Permite experimentar sin comprometer el código principal, integrar cambios de manera controlada y resolver discrepancias cuando varios colaboradores trabajan en el mismo archivo.

**Colaboración en GitHub**

GitHub no solo sirve como almacenamiento de proyectos, sino también como una plataforma para **trabajo colaborativo**, permitiendo que varios usuarios contribuyan en un mismo proyecto de manera organizada y controlada.

Invitar a compañeros a colaborar en el repositorio

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

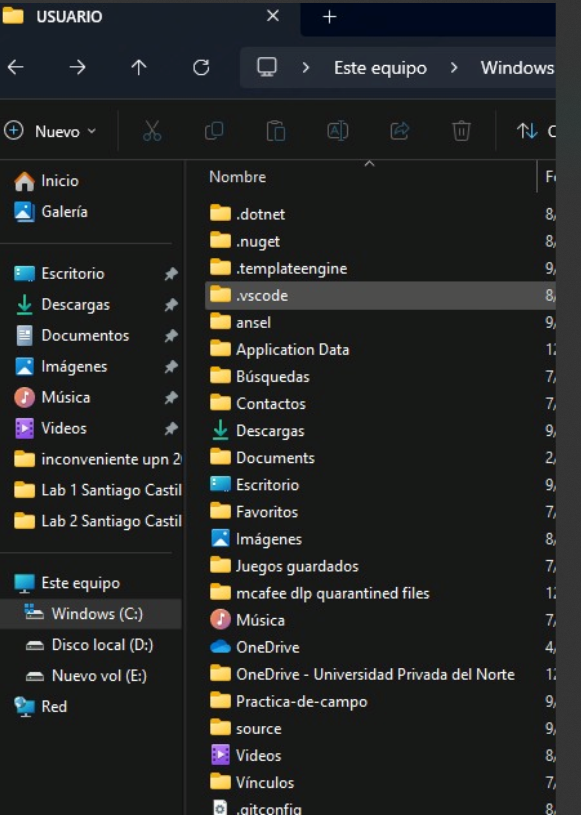
Esto permite que todos los miembros del grupo trabajen en la misma base de código sin interferir con los cambios de otros.

Clonar un repositorio compartido

Una vez que un compañero ha creado o compartido un repositorio, los demás pueden **copiarlo a su máquina local** para trabajar en él:

Texto

El contenido generado por IA puede ser incorrecto.



Pull Request (PR):

Un **Pull Request** permite **proponer cambios realizados en una rama** para que sean revisados antes de fusionarse con la rama principal (main):

Texto

El contenido generado por IA puede ser incorrecto.

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

Captura de pantalla con la imagen de una pantalla

El contenido generado por IA puede ser incorrecto.

Evidenciar revisión de código:

El Pull Request permite que el equipo o el profesor **evidencie la revisión del código**, ya que GitHub registra:

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

**Conclusión:**  
El uso de GitHub para colaboración permite que los equipos trabajen de forma eficiente, segura y documentada, con control de versiones, revisión de cambios y posibilidad de revertir errores de manera sencilla.

**Mejores prácticas y comandos avanzados en Git**

Git es una herramienta potente de control de versiones que permite **gestionar proyectos de forma eficiente**, especialmente cuando se trabaja en equipo. Para aprovechar todo su potencial, es recomendable seguir ciertas **mejores prácticas** y conocer comandos avanzados.

**Convenciones de nombres y mensajes de commit**

**Ramas**: usar nombres claros y consistentes facilita la comprensión del proyecto.

Ejemplo: main, desarrollo, feature/nueva-funcionalidad, bugfix/correccion-errores.

**mensajes de commit**: deben ser concisos, claros y descriptivos.

Buena práctica:

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

**Estrategias de ramificación**

* **Git Flow**: es un modelo de ramificación popular que organiza el desarrollo en varias ramas:
  + main → versión estable del proyecto.
  + develop → rama de integración donde se prueban nuevas funcionalidades.
  + feature/\* → ramas para desarrollar nuevas funcionalidades.
  + release/\* → ramas para preparar nuevas versiones.
  + hotfix/\* → ramas para correcciones urgentes en producción.
* Este modelo permite mantener un **flujo de trabajo organizado**, donde cada tipo de rama tiene un propósito claro y controlado.

**Comandos avanzados:**

**git stash**

Permite **guardar temporalmente cambios no confirmados** y limpiar el área de trabajo para cambiar de rama sin perder lo que estás haciendo.

**DESARROLLO**

1. **Diseñar los diagramas de clases del proyecto (5 puntos).**

Para el desarrollo del proyecto Sistema de Biblioteca, se elaboró un diagrama de clases UML utilizando la herramienta Draw.io, con el objetivo de representar las estructuras y relaciones entre las principales clases del sistema.

El diagrama incluye las clases Autor, Libro, Usuario, GestorArchivos y Principal.

Cada clase cuenta con sus atributos y métodos, reflejando los principios de la programación orientada a objetos, como la encapsulación y la modularidad.

La clase **Autor** define la información del escritor (nombre y nacionalidad).

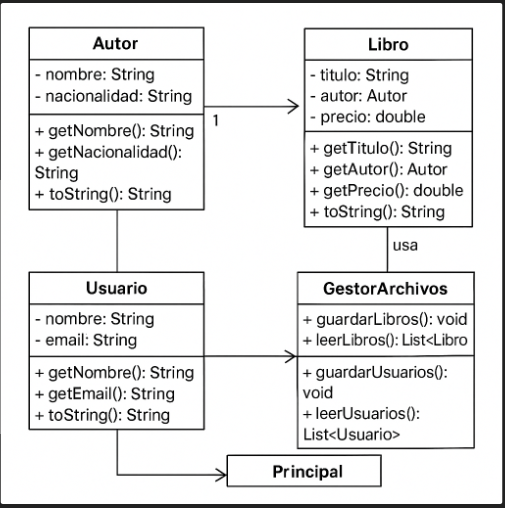
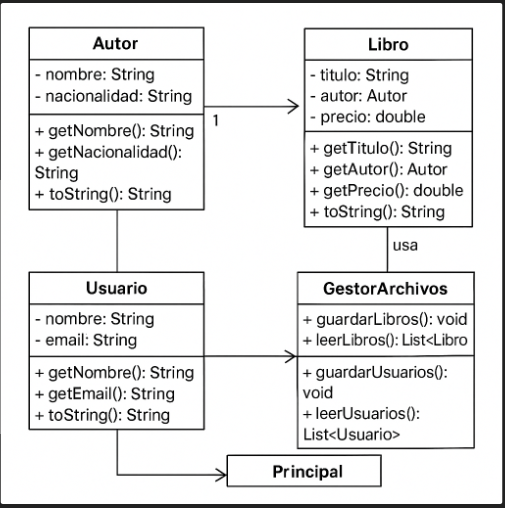
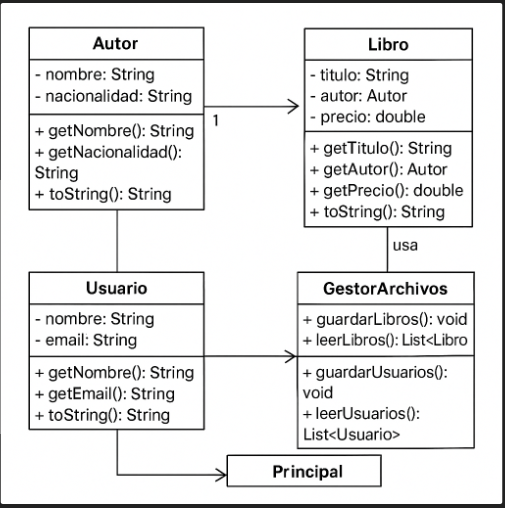
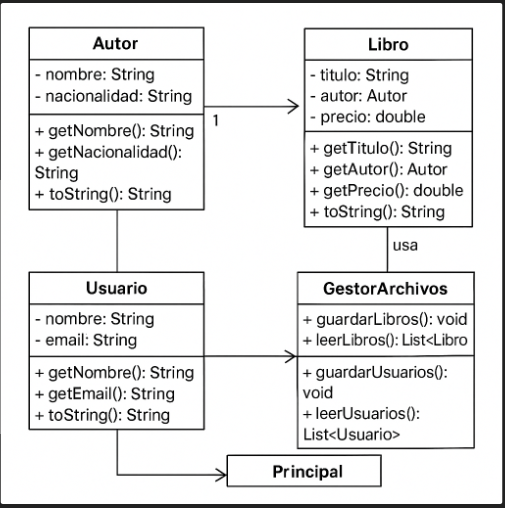
La clase **Libro** representa un libro e incluye atributos como título, precio y su relación con un Autor.

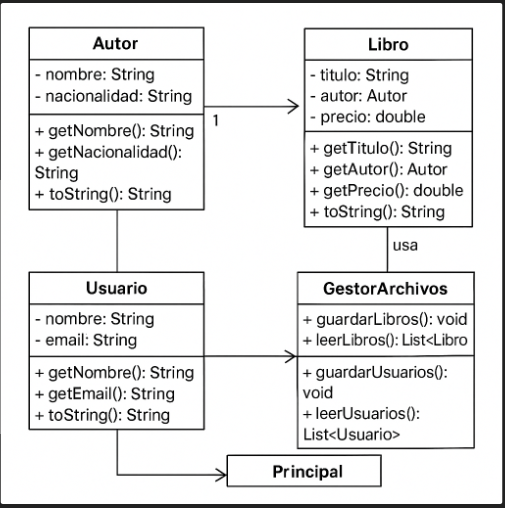
La clase **Usuario** almacena datos del usuario que interactúa con el sistema.

La clase **GestorArchivos** permite guardar y leer los registros desde archivos .txt.

Finalmente, **Principal** contiene el método main() que ejecuta el flujo general del programa.

Este diseño facilita la comprensión del sistema y su correcta implementación en código.

****

****

1. **Relacionar los diagramas de clase creados con la programación implementada del proyecto. (5 puntos).**

Las relaciones entre las clases se implementaron utilizando objetos como atributos dentro de otras clases.

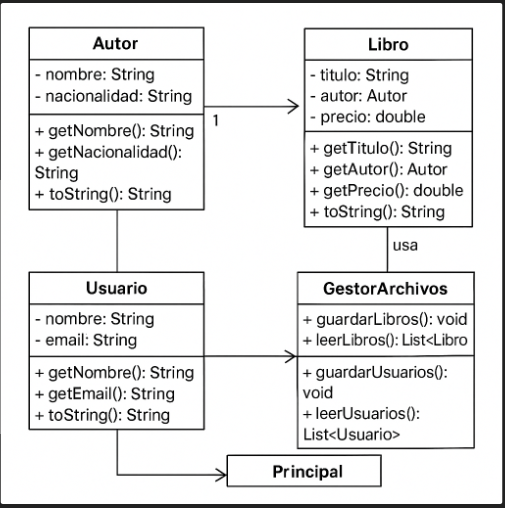
Por ejemplo, la clase **Libro** tiene un atributo de tipo **Autor**, lo que permite vincular directamente a cada libro con su escritor.

En el método **agregarLibro()**, se crea un nuevo objeto de tipo **Libro** y se le asigna un autor existente o recién creado mediante el método **agregarAutor().**

El método **mostrarLibros()** recorre una lista de objetos **Libro** y muestra sus datos en consola, accediendo a la información del autor a través del objeto que está asociado.

Esta estructura permite mantener la relación uno a muchos entre **Autor** y **Libro,** además de facilitar la lectura de datos completos sin duplicar información.

Todo el manejo se apoya en la clase **GestorArchivos,** que guarda y recupera la información de manera persistente en los archivos **.txt.**

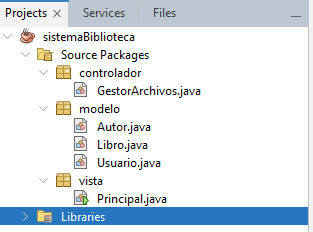
****

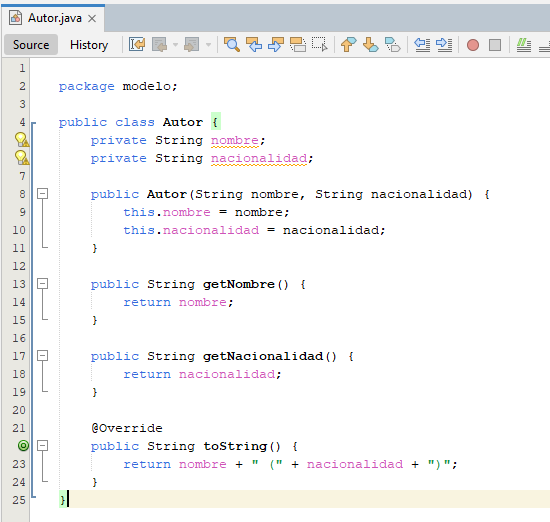
1. **Implementación de gestión de archivos. (5 puntos).**

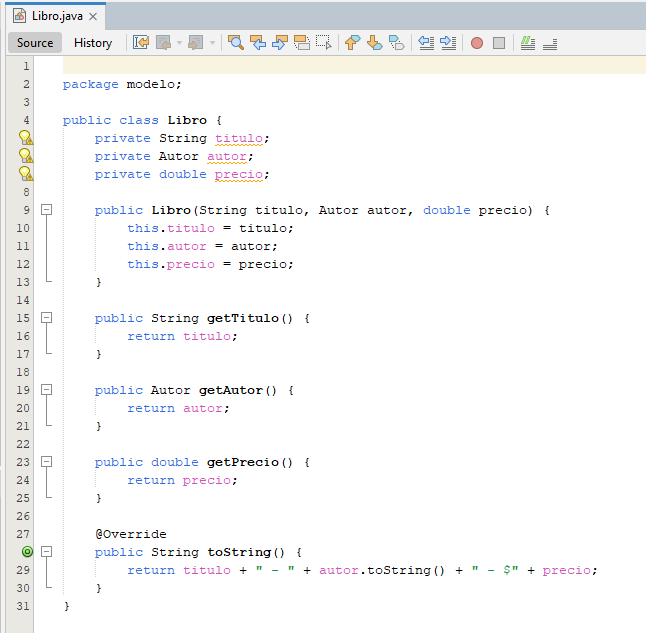
Para mantener la persistencia de los datos, se implementó una gestión de archivos mediante la clase **GestorArchivos.**

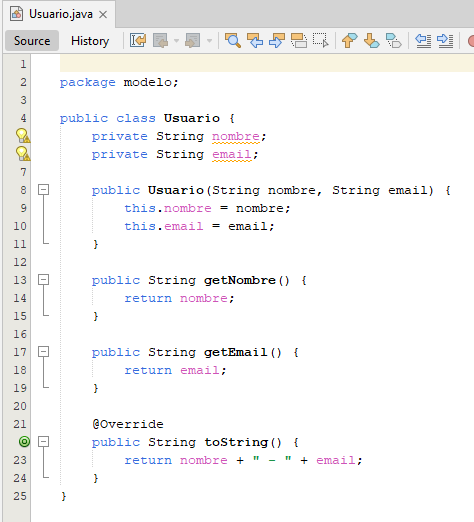
Esta clase permite guardar y leer información de libros y usuarios en archivos de texto plano **(libros.txt** y **usuarios.txt**), ubicados en la carpeta raíz del proyecto.

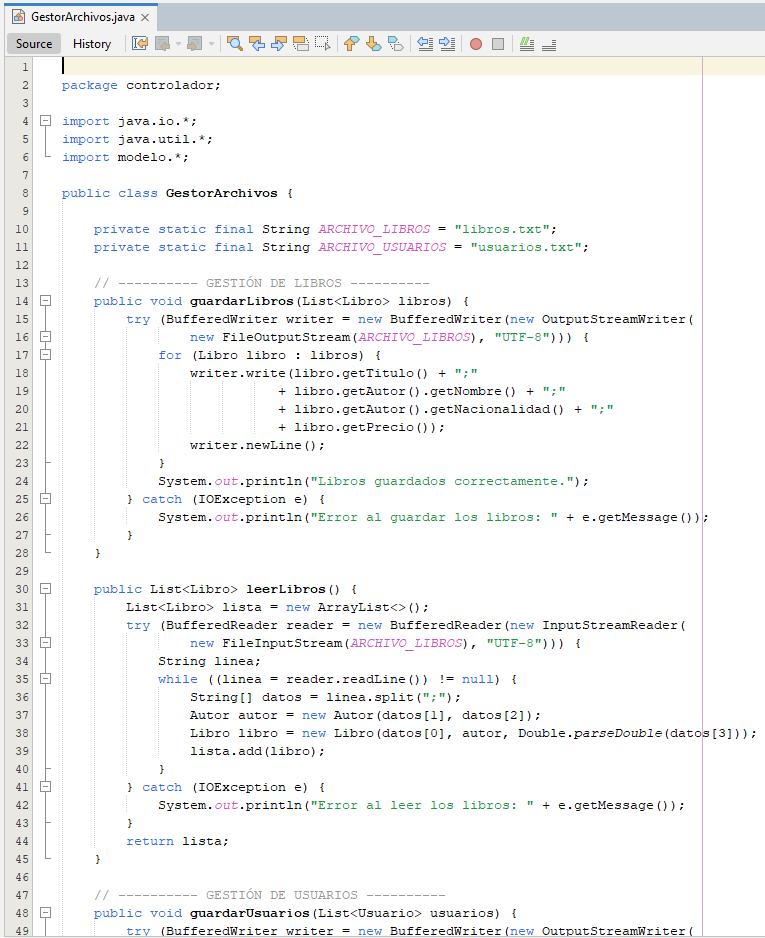
El proceso de escritura se realiza con la clase **FileWriter,** mientras que la lectura utiliza **BufferedReader,** permitiendo mostrar los datos en consola.

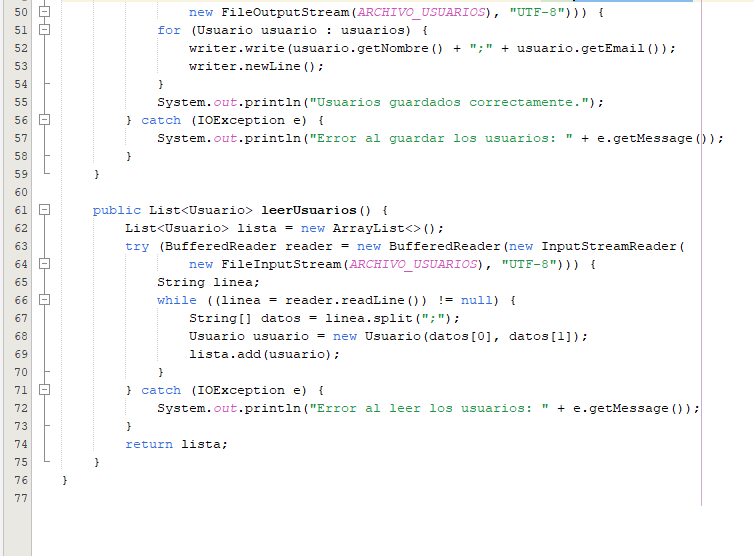






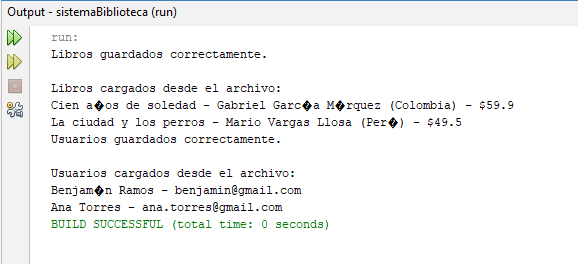








Ejemplo de funcionamiento:



Esta funcionalidad asegura que los registros no se pierdan al cerrar el programa y puedan ser reutilizados en futuras ejecuciones, cumpliendo el objetivo de persistencia del sistema.

1. **Control de estado de requerimientos funcionales implementados en relación con las historias de usuario. (5 puntos).**

