Universidad Tecnológica de Tijuana

TSU Desarrollo de Software Multiplataforma

Materias:

Programación Estructurada

Sistemas Operativos

Conmutación y Enrutamiento de Redes

Actividad:

Proyecto Final (Cinema World)

Alumnos:

Jimenez Cruz Obed de Jesus

Urquidez Arredondo Misael

Nava Pineda Eduardo

Profesores:

Dr. Marylu Lara Lagunes

Dr. Emer Ignacio Bernal Jimenez

Ing. Humberto Vejar Polanco

Fecha de entrega:

Martes 12 de Agosto del 2025

Cuatrimestre: 2 **Grupo:** 2-C

Índice

Índice de Ilustraciones

1.Introducción

Este documento presenta de manera integral el desarrollo, implementación y documentación de un proyecto de programación realizado en Java utilizando un entorno de trabajo remoto basado en Ubuntu. Se siguieron las prácticas recomendadas por el estándar IEEE 830 para la redacción de requisitos de software, integrando aspectos solicitados tanto en la materia de Programación como en la de Sistemas Operativos y, en menor medida, en Redes de Computadoras. El trabajo se ejecutó mediante la instalación y configuración del lenguaje Java desde la terminal de Ubuntu, aprovechando herramientas de administración remota como X2Go para acceder al entorno de trabajo. La gestión de versiones y colaboración se realizó a través de GitHub, asegurando un control ordenado y documentado del código fuente. La parte de redes se limitó a la verificación básica de conectividad mediante el uso de **ping**, cumpliendo con los requisitos solicitados. La documentación que aquí se presenta detalla cada una de las fases y elementos involucrados, desde el propósito del sistema hasta las restricciones y requisitos técnicos, con un enfoque formal y orientado a la reproducibilidad del proceso.

1.1 Propósito

El propósito de este documento es definir y detallar exhaustivamente los requisitos funcionales y no funcionales del sistema desarrollado, facilitando la comprensión clara y precisa de las funcionalidades que la aplicación en Java debe cumplir. Se busca que este documento sirva como una guía completa tanto para desarrolladores que deseen implementar o mejorar el sistema, como para evaluadores y docentes que requieran verificar el cumplimiento de los objetivos académicos planteados. El proyecto está diseñado para simular un entorno real de trabajo, donde se aplican conocimientos de programación, administración de sistemas y redes, mediante la integración de herramientas modernas y estándares de calidad reconocidos internacionalmente. Además, esta documentación pretende unificar las mejores prácticas de desarrollo de software con el uso de sistemas operativos robustos, control de versiones mediante GitHub, y protocolos básicos de redes, como el ping para verificación de conectividad. De esta manera, se establece una base sólida y estructurada que permita futuras ampliaciones o integraciones con sistemas más complejos, garantizando que el software desarrollado pueda ser entendido, reproducido, mantenido y evolucionado por usuarios con conocimientos básicos o intermedios en programación y administración de sistemas.

1.2 Ámbito del Sistema

El sistema descrito en este proyecto es una aplicación de consola desarrollada en Java, que funciona en un ambiente Linux, específicamente en una distribución Ubuntu configurada para permitir acceso remoto mediante X2Go. Esta aplicación tiene como función principal facilitar la captura y validación de datos relacionados con métodos de pago, tales como tarjetas de crédito o débito, PayPal y transferencias bancarias, mostrando posteriormente un resumen seguro y protegido de la información ingresada sin almacenar datos confidenciales. El ámbito del sistema se restringe a una aplicación local sin integración con bases de datos o servicios externos, enfocada en la práctica de conceptos fundamentales de programación, administración de sistemas y gestión de redes básicas. Se plantea así como una plataforma educativa que combina el aprendizaje teórico con la aplicación práctica, en un entorno controlado que simula condiciones reales de trabajo profesional. El sistema está diseñado para ser independiente y portable, con una arquitectura modular que permite una fácil extensión hacia futuras funcionalidades, como interfaces gráficas, bases de datos o comunicación en red. Este alcance delimitado permite que el usuario se concentre en aspectos esenciales del desarrollo de software y administración remota, sin complicaciones derivadas de integraciones complejas o dependencias externas. En resumen, el sistema funciona como una herramienta práctica y didáctica que cubre múltiples áreas de conocimiento, ofreciendo un entorno accesible y eficiente para el aprendizaje integral y la aplicación de estándares profesionales en el desarrollo de software.

1.3 Definiciones, Acrónimos y Abreviaturas

Este apartado agrupa y explica los términos técnicos, acrónimos y abreviaturas usados a lo largo del documento, para facilitar la comprensión de lectores con distintos niveles de conocimiento:

- Java: Lenguaje de programación orientado a objetos, portable y robusto, utilizado para desarrollar la aplicación principal del proyecto.
- Ubuntu: Distribución de Linux basada en Debian, usada como sistema operativo del servidor para el desarrollo y pruebas del software.
- X2Go: Software que permite el acceso remoto a entornos gráficos en sistemas Linux de forma segura y eficiente, facilitando el trabajo colaborativo y la administración remota
- GitHub: Plataforma web para la gestión de repositorios Git, que facilita el control de versiones, colaboración en proyectos y seguimiento histórico de cambios.
- Ping: Herramienta de red para verificar la conectividad entre equipos en una red IP, enviando paquetes ICMP y midiendo tiempos de respuesta.
- IEEE 830: Estándar internacional que establece las recomendaciones para la especificación de requisitos de software, que guía la estructura y contenido de este documento.
- CVV: Currículo
- C.W: Cinema world
- JDK (Java Development Kit): Conjunto de herramientas necesarias para desarrollar, compilar y ejecutar aplicaciones Java.
- CLI (Command Line Interface): Interfaz de línea de comandos, por la cual el usuario interactúa con el sistema operativo o aplicaciones.
- VPN (Virtual Private Network): Tecnología que permite establecer conexiones seguras y privadas a través de redes públicas.

Estas definiciones aseguran que cualquier lector pueda entender sin ambigüedades los términos técnicos y siglas presentes en la documentación, facilitando su lectura y aplicación.

1.3 Referencias

La realización de este proyecto y la elaboración de esta documentación se apoyaron en diversas fuentes confiables y documentos técnicos especializados, que garantizan la fundamentación adecuada y el cumplimiento de estándares reconocidos:

- Oracle Java Documentation: La documentación oficial de Oracle sobre el lenguaje Java, que abarca desde conceptos básicos hasta aspectos avanzados del lenguaje y su ecosistema.
- Ubuntu Official Documentation: Manuales y guías oficiales para la administración, instalación y configuración del sistema operativo Ubuntu, utilizados para establecer el entorno de desarrollo.
- GitHub Docs: Documentación oficial sobre el uso de GitHub para control de versiones, gestión de repositorios y colaboración en proyectos de software.
- IEEE Standard 830-1998: Estándar internacional que regula la especificación de requisitos de software, proporcionando las directrices para la correcta elaboración de documentos técnicos como el presente.
- Tutoriales y guías en línea sobre la instalación y configuración de X2Go, que facilitaron la implementación del acceso remoto al servidor.
- Manuales y referencias sobre comandos básicos de Linux para administración de sistemas y redes, esenciales para las operaciones realizadas en el entorno Ubuntu.

Estas fuentes constituyen la base técnica y metodológica que respalda la calidad y confiabilidad del desarrollo y la documentación del proyecto.

1.5 Visión General del Documento

Este documento se estructura en cuatro secciones principales que permiten al lector navegar de forma lógica y eficiente por el contenido, facilitando tanto la comprensión general como el acceso a detalles técnicos específicos:

- La Introducción proporciona el contexto, objetivos y alcance del proyecto, ofreciendo una visión general clara sobre la finalidad del trabajo y los estándares empleados.

- La Descripción General detalla el entorno en el que se desarrolla el producto, sus funcionalidades básicas, el perfil de los usuarios previstos, así como las restricciones y dependencias que condicionan su uso y evolución.

- La sección de Requisitos Específicos profundiza en las funcionalidades concretas que el sistema debe cumplir, abarcando interfaces, procesos, requisitos de rendimiento, atributos de calidad y limitaciones de diseño.

- Finalmente, los Apéndices contienen información complementaria y soporte técnico relevante, como fragmentos de código, instrucciones de instalación y configuración, que facilitan la replicabilidad y mantenimiento del proyecto.

La organización busca equilibrar la profundidad técnica con la accesibilidad, siendo útil tanto para desarrolladores y administradores, como para evaluadores académicos y profesionales del área.

2. Descripción General

2.1 Perspectiva del Producto

El producto desarrollado es una aplicación independiente que se ejecuta en un entorno Linux (Ubuntu) mediante línea de comandos, orientada a la captura y validación de datos para métodos de pago, simulando procesos comunes en aplicaciones comerciales y financieras. Esta independencia tecnológica y operativa le confiere portabilidad y flexibilidad para ser adaptado a otros sistemas o integrarse a soluciones mayores.BEl proyecto está concebido como una herramienta educativa que integra tres áreas fundamentales: programación en Java, administración de sistemas operativos y conceptos básicos de redes. Esta integración facilita una comprensión multidisciplinaria, permitiendo al usuario no solo programar, sino también configurar entornos remotos seguros y verificar la conectividad básica entre equipos. Desde el punto de vista arquitectónico, el producto está diseñado con modularidad y claridad, implementando cada funcionalidad como métodos separados y evitando dependencias externas o tecnologías complejas. Esto facilita su mantenimiento, ampliación y reutilización en futuros proyectos o contextos más avanzados. El producto no cuenta con interfaz gráfica ni almacenamiento persistente, limitándose a la ejecución local en consola y la interacción directa con el usuario. Sin embargo, esta simplicidad es intencional, priorizando la comprensión y dominio de conceptos fundamentales que sirven de base para desarrollos posteriores más complejos.

2.2 Funciones del Producto

El sistema realiza un conjunto definido de funciones que cumplen con el propósito de simular la captura y procesamiento básico de métodos de pago de forma segura y sencilla. Estas funciones se describen a continuación de manera detallada y secuencial:

- Presentación del menú de métodos de pago: El sistema despliega un menú claro y sencillo con opciones enumeradas para que el usuario seleccione el método de pago deseado, validando que la opción elegida sea válida y, en caso contrario, solicitando reingreso sin interrumpir la ejecución.

- Captura de datos del método de pago: Dependiendo de la opción seleccionada, el sistema solicita información específica como nombre del titular, últimos dígitos de la tarjeta, mes y año de expiración y código CVV. Se incluyen validaciones para asegurar que los datos tengan un formato correcto y estén dentro de rangos aceptables, evitando la captura de datos erróneos o malformados.

- Validación y procesamiento interno**: La aplicación realiza comprobaciones básicas para verificar que la información ingresada sea coherente (por ejemplo, que el mes esté entre 1 y 12, que el año no sea anterior al actual), y gestiona internamente estos datos evitando la exposición de información sensible o almacenamiento no autorizado.

- Visualización de resumen seguro: El sistema muestra un resumen con los datos relevantes para el usuario, ocultando información sensible como los dígitos completos de la tarjeta o el código CVV, garantizando así la privacidad y seguridad del usuario durante la interacción.

- Gestión modular de funciones: Cada funcionalidad está implementada como un método o procedimiento independiente, facilitando la legibilidad del código, su mantenimiento y futuras ampliaciones sin afectar el conjunto global del sistema.

- Control de errores y robustez: Se implementan mecanismos que permiten al sistema manejar entradas inválidas o errores de usuario sin que se produzcan fallas o cierres abruptos, mostrando mensajes claros y solicitando correcciones oportunas.

Estas funciones configuran un flujo operativo sencillo pero completo, que asegura una experiencia de usuario coherente y un código bien estructurado acorde con buenas prácticas de desarrollo.

2.3 Características de los Usuarios

Los usuarios previstos para este sistema son principalmente estudiantes y desarrolladores en formación interesados en adquirir habilidades prácticas en programación en Java, administración de sistemas Linux y fundamentos básicos de redes. Estos usuarios poseen conocimientos elementales a intermedios en programación, manejo de terminal y conceptos básicos de redes, y buscan consolidar estos conocimientos mediante la aplicación práctica en un entorno controlado y realista. Asimismo, el sistema está orientado para ser utilizado por docentes y evaluadores que requieren una herramienta clara y estructurada para enseñar, demostrar y evaluar competencias técnicas relacionadas con la ingeniería de software, administración de sistemas y comunicación en redes. Los usuarios valoran la simplicidad y claridad en la interacción, así como la modularidad y documentación del código, que les permite entender fácilmente la estructura del programa y modificarlo para experimentar con nuevas funcionalidades. Este perfil de usuario también implica que el sistema debe ser tolerante a errores y brindar mensajes claros para facilitar el aprendizaje, evitando sobrecargar con complejidades innecesarias y manteniendo un enfoque educativo práctico y efectivo.

2.4 Restricciones

El desarrollo y operación del sistema están sujetos a una serie de restricciones técnicas y funcionales que delimitan su diseño y uso, buscando mantener la simplicidad y efectividad del proyecto:

- Interacción exclusivamente en consola: No se implementó interfaz gráfica, por lo que todas las operaciones deben realizarse mediante terminal o consola, limitando la experiencia visual pero facilitando la portabilidad y compatibilidad.

- Ausencia de almacenamiento persistente: No se utiliza ninguna base de datos ni archivos para guardar información, por lo que todos los datos son temporales y se manejan exclusivamente en memoria durante la ejecución del programa.

- Dependencia del entorno Java y Linux: El sistema requiere que el JDK esté correctamente instalado y configurado en Ubuntu, condicionando su funcionamiento a la disponibilidad y configuración adecuada de este entorno.

- Seguridad limitada: Al tratarse de un proyecto académico, no se aplican mecanismos avanzados de cifrado ni autenticación, por lo que la seguridad de los datos depende principalmente de la correcta interacción del usuario y del entorno cerrado donde se ejecuta.

- Red limitada a verificación básica: Las funcionalidades de red se reducen a la comprobación de conectividad mediante el comando ping y acceso remoto seguro vía X2Go, sin implementación de protocolos o servicios de comunicación complejos.

- Recursos limitados del servidor: El entorno de desarrollo dispone de recursos hardware limitados, lo que puede impactar el rendimiento bajo cargas elevadas o escenarios más complejos.

Estas restricciones aseguran que el proyecto se mantenga en un ámbito controlado, propicio para el aprendizaje y la práctica sin complicaciones innecesarias.

2.5 Suposiciones y Dependencias

Para garantizar el correcto desarrollo y operación del sistema, se asumen ciertas condiciones y dependencias que el entorno y los usuarios deben cumplir:

- Conectividad estable para acceso remoto: Se asume que los usuarios cuentan con una conexión de red confiable para utilizar X2Go y acceder remotamente al entorno gráfico del servidor Ubuntu.

- Instalación correcta de Java: Se requiere que la versión del JDK esté correctamente instalada y configurada en el sistema operativo, permitiendo la compilación y ejecución sin errores.

- Dominio básico de comandos de terminal: Se supone que los usuarios tienen conocimientos mínimos para operar la terminal Linux, incluyendo compilación de código Java, navegación de directorios y ejecución de comandos básicos.

- Acceso y permisos en GitHub: El repositorio del proyecto debe ser accesible y contar con permisos adecuados para clonar, actualizar y gestionar versiones del código fuente.

- Conocimientos fundamentales de programación: Se espera que los usuarios comprendan conceptos básicos de programación estructurada, uso de variables, condicionales y métodos en Java, facilitando la comprensión y modificación del código.

Estas suposiciones permiten enfocar el proyecto en aspectos técnicos sin abordar problemas derivados de configuraciones incorrectas o falta de habilidades básicas.

2.6 Requisitos Futuros

El proyecto está concebido con una visión de evolución y ampliación que contempla múltiples áreas de mejora para incrementar su funcionalidad, usabilidad y robustez:

- Implementación de interfaces gráficas: Añadir interfaces gráficas utilizando tecnologías como Java Swing o JavaFX para mejorar la experiencia del usuario y facilitar la interacción.
- Integración de bases de datos: Incorporar sistemas de almacenamiento persistente mediante bases de datos SQL o NoSQL para gestionar usuarios, métodos de pago y registros históricos.
- Comunicación en red avanzada: Desarrollar funcionalidades cliente-servidor que permitan la interacción en red mediante sockets o servicios web REST, ampliando el alcance del sistema.
- Seguridad reforzada: Implementar mecanismos de cifrado, autenticación y validación avanzada para proteger la información sensible y garantizar la integridad de las transacciones.
- Portabilidad multiplataforma mejorada: Automatizar despliegues y configuraciones para que el sistema funcione de forma nativa en diferentes sistemas operativos con mínima intervención.
- Automatización de pruebas: Desarrollar pruebas unitarias, de integración y funcionales automatizadas para asegurar la calidad y estabilidad del software en el tiempo.
- Documentación continua: Mantener la documentación actualizada y enriquecida, incluyendo manuales de usuario, guías de instalación y planes de mantenimiento.

Estas mejoras plantean un camino claro para la evolución del sistema hacia aplicaciones más complejas y profesionales, facilitando su adopción en contextos reales y comerciales.

3. Requisitos Específicos

3.1 Interfaces Externas

El sistema interactúa con diversos componentes externos, cuya correcta configuración es fundamental para garantizar un funcionamiento adecuado y seguro:

- Interfaz de usuario (UI): La interacción se realiza a través de la consola de comandos del sistema operativo Ubuntu, accesible localmente o mediante sesiones remotas X2Go con entorno gráfico XFCE, que proporcionan una experiencia ligera y eficiente.

- Herramientas de desarrollo y gestión: El entorno requiere la instalación y configuración del JDK para la compilación y ejecución del código Java, así como el uso de Git y GitHub para el control de versiones, permitiendo colaboración y seguimiento del proyecto.

- Servicios de red y seguridad: El servidor debe permitir conexiones SSH mediante el puerto 22, con firewalls configurados para garantizar un acceso seguro, además del uso de herramientas como ping para verificación de conectividad.

- Clientes remotos: Los usuarios pueden acceder al servidor desde diferentes sistemas operativos (Windows, Linux, macOS) utilizando clientes compatibles con X2Go, asegurando flexibilidad y accesibilidad.

- Utilidades de red básicas: Se emplean comandos estándar del sistema Linux para tareas como comprobación de red, gestión de usuarios y configuración del entorno.

Estas interfaces garantizan un entorno integrado que soporta las necesidades funcionales y operativas del sistema.

3.2 Funciones

El sistema debe cumplir con las siguientes funciones operativas y técnicas, detalladas para asegurar claridad en su desarrollo e implementación:

1. Mostrar menú interactivo para selección del método de pago:

- Desplegar opciones numeradas para que el usuario seleccione el método deseado.
- Validar que la opción seleccionada corresponda a un valor válido, solicitando reingreso en caso contrario.
- Permitir la opción de salida del programa de forma controlada.

2. Capturar datos específicos según método de pago:

- Solicitar nombre completo del titular de la cuenta o tarjeta.
- Capturar últimos cuatro dígitos del número de tarjeta, asegurando que sean numéricos y de longitud correcta.
- Pedir mes y año de expiración, validando rangos apropiados (mes 1-12, año no menor al actual).
- Solicitar código CVV, validando formato y longitud adecuada.

3. Validar entradas y controlar errores:

- Comprobar que cada dato cumple con el formato y rango esperado.
- En caso de error, mostrar mensajes claros y solicitar corrección sin terminar abruptamente la ejecución.

4. Mostrar resumen seguro de la información:

- Presentar el nombre del titular y tipo de método de pago seleccionado.
- Mostrar número de tarjeta parcialmente oculto, revelando solo los últimos cuatro dígitos.
- Mostrar fecha de expiración en formato legible.
- Ocultar el código CVV para proteger la privacidad del usuario.

5. Permitir repetir operaciones o salir:

- Ofrecer al usuario la posibilidad de realizar una nueva captura o finalizar el programa.

6. Estructura modular y código limpio:

- Implementar cada función en métodos separados, con nombres descriptivos y comentarios explicativos, facilitando mantenimiento y escalabilidad.
- Estas funciones configuran el comportamiento completo del sistema, asegurando funcionalidad, usabilidad y robustez.

7. cw.java

- Clase principal del programa. Gestiona el flujo inicial de ejecución, mostrando el menú para registrar usuarios, iniciar sesión o salir.
- Determina si el usuario es administrador o cliente normal y redirige a Admin.java o Reproductor.java según el caso.
- También conecta con Suscripcion.java para manejo de pagos y membresías.

8. Suscripcion.java

- Encargada de procesar pagos y administrar las suscripciones de los usuarios.
- Controla fechas de inicio y fin de membresía, tipos de suscripción, métodos de pago y puntos asociados.
- Valida si un usuario tiene acceso activo al contenido.

9. Reproductor.java

- Maneja la reproducción de contenido. Presenta opciones de listas, reproduce elementos y administra puntos o recompensas para el usuario.
- Regresa al menú principal después de finalizar la reproducción.

10. Admin.java

- Proporciona el menú de administrador para gestionar usuarios. Permite ver todos los registros de usuarios y eliminar cuentas, exceptuando el administrador principal. Lee y escribe datos en el archivo 'usuarios.txt'

3.3 Requisitos de Rendimiento

Para garantizar una experiencia de usuario óptima y un funcionamiento eficiente, el sistema debe cumplir con ciertos parámetros de rendimiento:

- La respuesta a las entradas del usuario debe ser inmediata, sin retardos perceptibles, asegurando una interacción fluida y sin interrupciones.

- La compilación y ejecución del programa Java desde terminal deben completarse en menos de un segundo en entornos con recursos mínimos, optimizando el ciclo de desarrollo y pruebas.

- El acceso remoto mediante X2Go debe mantener latencias bajas, garantizando que las acciones del usuario se reflejen rápidamente en el entorno remoto para evitar frustración o errores por retrasos.

- El consumo de recursos del sistema, incluyendo CPU y memoria RAM, debe ser mínimo y eficiente, evitando que la aplicación afecte el rendimiento general del servidor o cause bloqueos.

Estos requisitos aseguran que el sistema sea práctico para su uso en entornos educativos y de desarrollo, manteniendo una buena experiencia y estabilidad operativa.

3.4 Restricciones de Diseño

El diseño del sistema debe respetar ciertas restricciones que garantizan coherencia, mantenibilidad y cumplimiento de los objetivos del proyecto:

- La aplicación debe ser modular, con funciones implementadas en métodos independientes, usando nomenclatura clara y consistente, y código comentado para facilitar la comprensión.
- Se debe emplear exclusivamente Java estándar, evitando librerías externas o dependencias que compliquen la instalación o ejecución.
- No se debe implementar interfaz gráfica, limitando la interacción a la consola para mantener la simplicidad y enfoque académico.
- Las validaciones deben asegurar la integridad y coherencia de los datos, evitando errores que provoquen fallos o comportamientos inesperados.
- El código debe ser portable y funcionar en cualquier distribución Linux que cuente con JDK instalado y configurado adecuadamente.
- Se debe evitar el almacenamiento persistente, garantizando que la información sensible no quede registrada en disco o sistemas externos.

Estas restricciones promueven la calidad y robustez del desarrollo, facilitando la adaptación y uso en diferentes contextos.

3.5 Atributos del Sistema

El sistema desarrollado presenta atributos clave que definen su calidad y valor para los usuarios y administradores:

- Portabilidad: Puede ejecutarse en cualquier sistema Linux con Java configurado, permitiendo su uso en diversos entornos sin modificaciones significativas.
- Seguridad: Protege la privacidad del usuario al no almacenar ni mostrar datos completos sensibles, siguiendo buenas prácticas para evitar filtraciones.
- Usabilidad: Ofrece una interacción sencilla y clara mediante mensajes comprensibles y un flujo lógico, facilitando su uso por usuarios con conocimientos básicos.
- Mantenibilidad: Su código está organizado, comentado y estructurado en módulos, facilitando futuras modificaciones y ampliaciones sin afectar la estabilidad.
- Fiabilidad: Maneja entradas erróneas de forma controlada, evitando cierres inesperados o errores críticos durante la ejecución.
- Escalabilidad: Su arquitectura modular permite incorporar nuevas funcionalidades y mejoras de forma ordenada y eficiente.

Estos atributos aseguran que el sistema sea una herramienta confiable, accesible y durable para fines educativos y de desarrollo.

3.6 Otros Requisitos

Además de los requisitos funcionales y técnicos, el proyecto debe cumplir con aspectos adicionales importantes para su correcta implementación y uso:

- La documentación debe seguir estándares internacionales (IEEE 830), ofreciendo claridad, orden y precisión para facilitar la comprensión y evaluación.

- El repositorio GitHub debe contener el código fuente completo, junto con un historial de versiones, archivos de configuración y documentación asociada, permitiendo la colaboración y seguimiento.

- Se debe garantizar que el sistema permita acceso remoto seguro y eficiente mediante herramientas como X2Go, asegurando que los usuarios puedan operar y administrar el entorno sin complicaciones.

- Los usuarios deben disponer de guías o tutoriales básicos para la instalación y configuración del entorno, facilitando la puesta en marcha del sistema.

- Se recomienda mantener prácticas de seguridad en el servidor Ubuntu, como configuración de firewalls y actualizaciones, para proteger el entorno donde se ejecuta la aplicación.

Estos aspectos complementan la funcionalidad técnica, asegurando una experiencia integral y profesional.

4. Apéndices

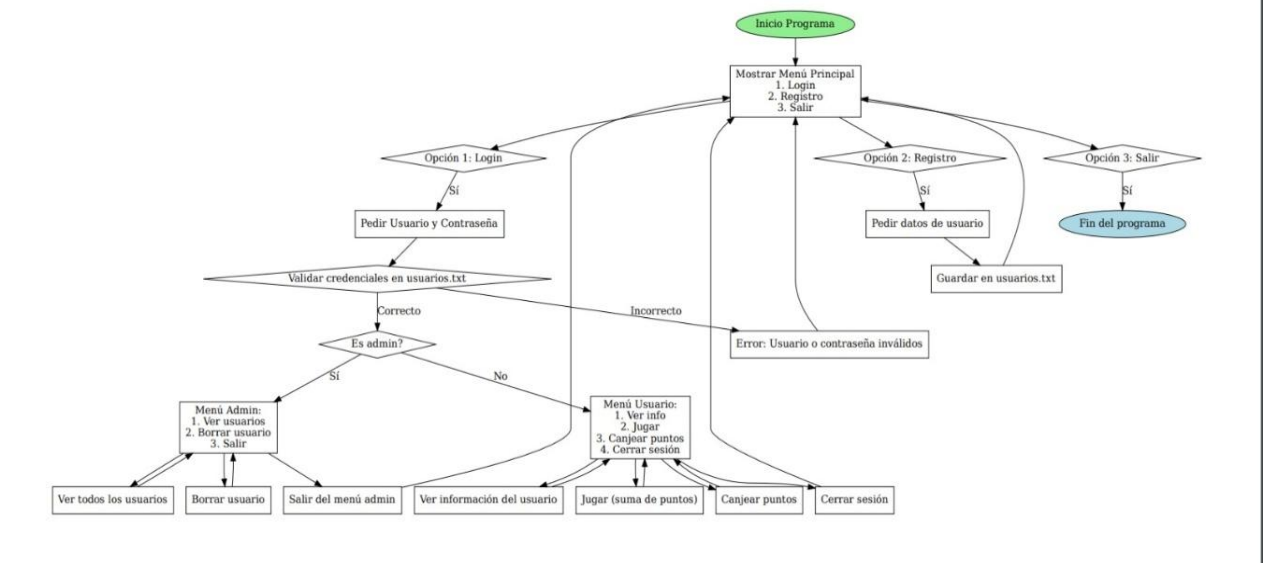


Ilustración 1. DIagrama de flujo inicio de programa

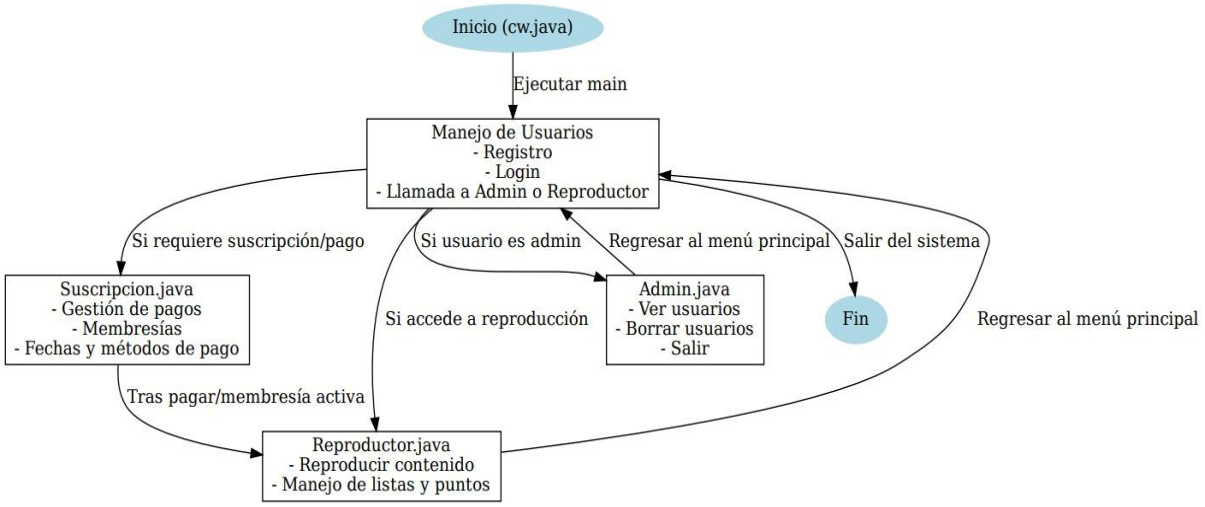


Ilustración 2. diagrama de flujo Inicio (C.W)

metodoDepago.java 9

metodoDepago.java > Java > metodoDepago > solicitarDatosPago()

```java
import java.util.Scanner;

public class metodoDepago {
    static Scanner sc = new Scanner(System.in);

    // Método que devuelve una cadena con resumen del método de pago
    public static String solicitarDatosPago() {
        System.out.println(x:"--Elija su método de pago--");
        System.out.println(x:"1. Tarjeta de débito");
        System.out.println(x:"2. Tarjeta de crédito");
        System.out.println(x:"3. Tarjeta prepagada");
        String tipoPago = "";
        int metodoPago = 0;
        while (true) {
            try {
                metodoPago = Integer.parseInt(sc.nextLine());
                if (metodoPago >= 1 && metodoPago <= 3) break;
                else System.out.println(x:"Opción no válida, ingresa un número entre 1 y 3.");
            } catch (NumberFormatException e) {
                System.out.println(x:"Entrada inválida, ingresa un número.");
            }
        }

        switch (metodoPago) {
            case 1: tipoPago = "Tarjeta de débito"; break;
            case 2: tipoPago = "Tarjeta de crédito"; break;
            case 3: tipoPago = "Tarjeta prepagada"; break;
        }

        System.out.println(x:"--Ingrese los datos de la tarjeta--");

        System.out.print(s:"Nombre del titular: ");
        String nombreTitular = sc.nextLine();

        System.out.print(s:"Número de la tarjeta: ");
        String numeroTarjeta = sc.nextLine();

        System.out.print(s:"Fecha de vencimiento (MM/AA): ");
        String fechaVencimiento = sc.nextLine();

        System.out.print(s:"Número de seguridad (CVV): ");
        String cvv = sc.nextLine();

        System.out.print(s:"Código postal: ");
        String codigoPostal = sc.nextLine();

        System.out.println(x:"--Datos ingresados correctamente--");

        // Solo guardamos tipo de pago y nombre titular para simplificar y evitar guardar datos sensibles
        return tipoPago + " - Titular: " + nombreTitular;
    }
}
```

Ilustración 3. Código de método de pago

pagos.java 3, U

pagos.java > Java > pagos > main(String[] args)

```java
import java.util.Scanner;
public class pagos {
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        Scanner teclado = new Scanner(System.in);
        System.out.println(x:"¿Qué deseas hacer? (registrarse / iniciar seccion / renovar)");
        String accion = teclado.nextLine();
        if (accion.equalsIgnoreCase(anotherString:"iniciar seccion")) {
            System.out.println(x:"Elegiste iniciar sesión");
        } else if (accion.equalsIgnoreCase(anotherString:"registrarse")) {
            System.out.println(x:"Elegiste registrarte");
            System.out.println(x:"Ingresa tu nombre de usuario:");
            String usuario = teclado.nextLine();
            if (usuario.equalsIgnoreCase(anotherString:"Misael")) {
                System.out.println(x:"Usuario correcto, ahora ingresa la contraseña:");
                String contrasena = teclado.nextLine();
                if (contrasena.equals(anObject:"24108730")) {
                    System.out.println("Contraseña correcta. Bienvenido " + usuario);
                } else {
                    System.out.println(x:"Contraseña incorrecta");
                }
            } else {
                System.out.println(x:"Usuario incorrecto");
            }
        } else if (accion.equalsIgnoreCase(anotherString:"renovar")) {
            System.out.println(x:"Elegiste renovar membresía");
        } else {
            System.out.println(x:"No elegiste ninguna opción válida");
        }
        System.out.println(x:"¿Qué tipo de membresía deseas? (normal / vip)");
        String tipoMembresia = teclado.nextLine();
        if (tipoMembresia.equalsIgnoreCase(anotherString:"normal")) {
            System.out.println(x:"Elegiste membresía NORMAL");
            System.out.println(x:"Elige una opción:\n1. Un mes\n2. Tres meses\n3. Seis meses\n4. Un año");
            int opcion = teclado.nextInt();
            if (opcion == 1) {
                System.out.println(x:"Pagarás 200 $");
            } else if (opcion == 2) {
                System.out.println(x:"Pagarás 600 $");
            } else if (opcion == 3) {
                System.out.println(x:"Pagarás 900 $");
            } else if (opcion == 4) {
                System.out.println(x:"Pagarás 1400 $");
            } else {
                System.out.println(x:"Opción inválida");
            }
        } else if (tipoMembresia.equalsIgnoreCase(anotherString:"vip")) {
            System.out.println(x:"Elegiste membresía VIP");
            System.out.println(x:"Elige una opción:\n1. Un mes\n2. Tres meses\n3. Seis meses\n4. Un año");
            int opcion = teclado.nextInt();
            if (opcion == 1) {
                System.out.println(x:"Pagarás 299 $");
            } else if (opcion == 2) {
                System.out.println(x:"Pagarás 699 $");
            } else if (opcion == 3) {
                System.out.println(x:"Pagarás 999 $");
            } else if (opcion == 4) {
                System.out.println(x:"Pagarás 1499 $");
            } else {
                System.out.println(x:"Opción inválida");
            }
        } else {
            System.out.println(x:"No elegiste una membresia válida");
        }
        teclado.close();
    }
}
```

Ilustración 4. Código de pago

suscripcion.java 9+

suscripcion.java > Language Support for Java(TM) by Red Hat > suscripcion > gestionarNuevaCuenta(String, String)

```java
// Importa clases para manejar archivos, fechas y listas
import java.io.*;
// Importa clases para trabajar con archivos de forma moderna (leer, escribir)
import java.nio.file.Files;
import java.nio.file.Paths;
// Importa clases para trabajar con fechas (obtener fecha actual, formatear)
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
// Importa clase para manejar listas
import java.util.List;

// Declara la clase llamada suscripcion
public class suscripcion {
    // Declara una constante que guarda el nombre del archivo donde están los usuarios
    static final String ARCHIVO_USUARIOS = "usuarios.txt";
    // Crea un objeto Scanner para poder leer texto que escribe el usuario en consola
    static java.util.Scanner sc = new java.util.Scanner(System.in);

    // Método público que pregunta si el usuario quiere suscribirse ahora o no
    // Recibe el nombre de usuario y contraseña para luego usar
    // Devuelve el método de pago escogido o "sinmetodo" si no quiere suscribirse
    public static String gestionarNuevaCuenta(String usuario, String contrasena) {
        // Muestra el mensaje en pantalla y pide que el usuario responda
        System.out.print(s:"¿Quieres suscribirte ahora? (s/si/n): ");
        // Lee lo que el usuario escribió y lo guarda en una variable, quitando espacios y con minúsculas
        String respuesta = sc.nextLine().trim().toLowerCase();

        // Compara si la respuesta es "s" o "si" para saber si quiere suscribirse
        if (respuesta.equals(anObject:"s") || respuesta.equals(anObject:"si")) {
            // Si sí, llama al método que muestra las opciones de membresía
            return mostrarOpciones(usuario, contrasena);
        } else {
            // Si no, dice que puede suscribirse después y devuelve "sinmetodo"
            System.out.println(x:"Puedes suscribirte más tarde desde el menú.");
            return "sinmetodo";
        }
    }

    // Método privado que muestra las opciones de membresía y maneja la selección
    private static String mostrarOpciones(String usuario, String contrasena) {
        // Imprime el título de la sección
        System.out.println(x:"\n--- Opciones de Membresía ---");
        // Muestra las opciones disponibles con su costo
        System.out.println(x:"1. Membresía de 1 mes  - $299");
        System.out.println(x:"2. Membresía de 3 meses - $699");
        System.out.println(x:"3. Membresía de 6 meses - $999");
        System.out.println(x:"4. Membresía de 12 meses (1 año) - $1000");
        // Pide que el usuario elija una opción
        System.out.print(s:"Elige una opción (1-4): ");

        // Lee la opción que escribió el usuario
        String opcion = sc.nextLine().trim();
        // Variables que usarán para guardar datos según la opción elegida
        int meses = 0;       // Cantidad de meses que dura la membresía
        String tipo = "";    // Nombre de la membresía
        double precio = 0;   // Precio en pesos

        // Evalúa la opción que escribió el usuario
        switch (opcion) {
            case "1":
                meses = 1;                // Si eligió 1, la membresía dura 1 mes
                tipo = "Membresía 1 mes"; // Guardamos el nombre
                precio = 299;             // Guardamos el precio
                break;                    // Salimos del switch
            case "2":
                meses = 3;
                tipo = "Membresía 3 meses";
```

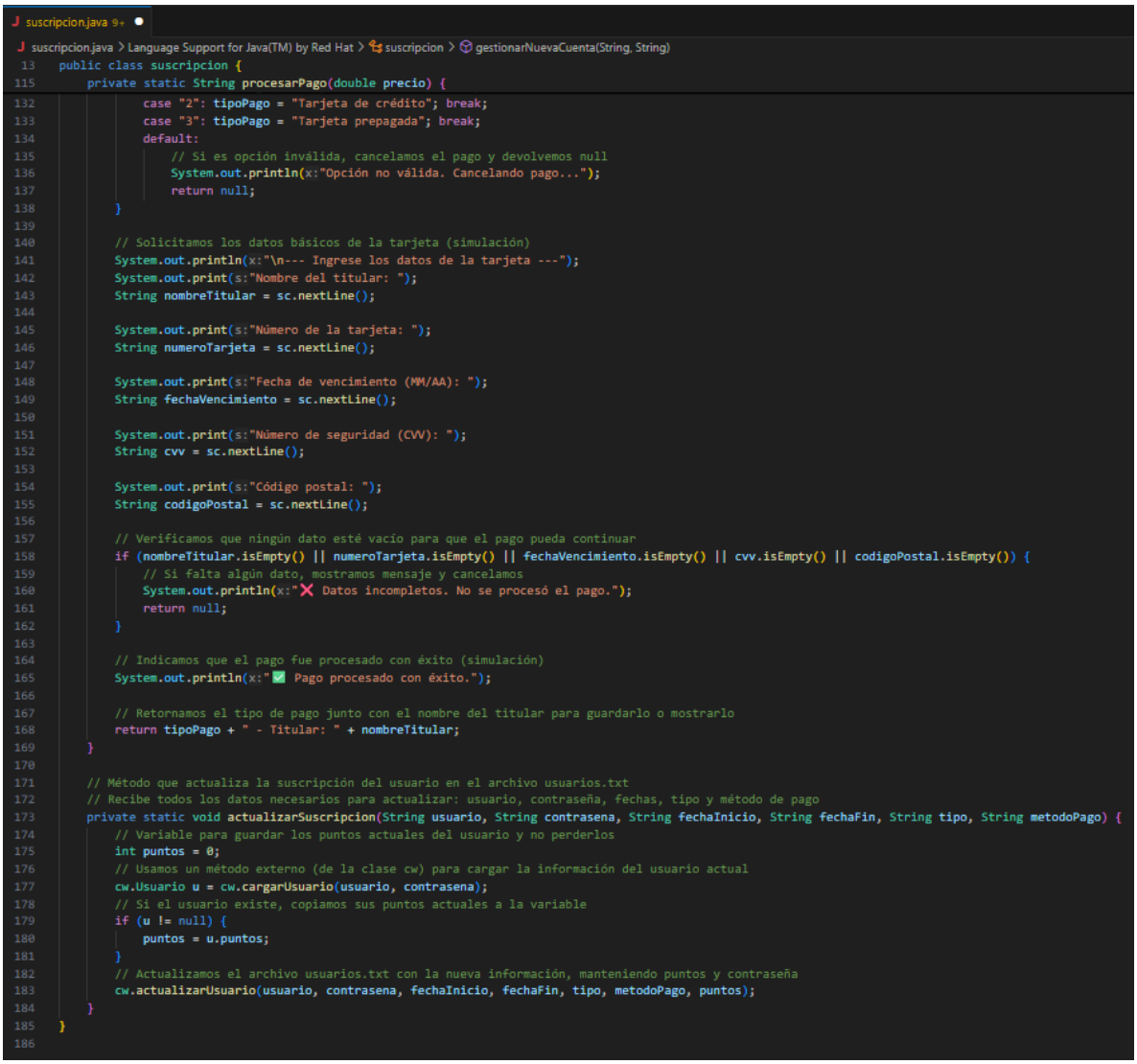
Ilustración 5. Código de suscripción parte 1

suscripcion.java 9+

suscripcion.java > Language Support for Java(TM) by Red Hat > suscripcion > gestionarNuevaCuenta(String, String)

```
13  public class suscripcion {
40      private static String mostrarOpciones(String usuario, String contrasena) {
67                  tipo = "Membresía 3 meses";
68                  precio = 699;
69                  break;
70              case "3":
71                  meses = 6;
72                  tipo = "Membresía 6 meses";
73                  precio = 999;
74                  break;
75              case "4":
76                  meses = 12;
77                  tipo = "Membresía 12 meses";
78                  precio = 1000;
79                  break;
80              default:
81                  // Si no es ninguna opción válida, mostramos mensaje y regresamos "sinmetodo"
82                  System.out.println(x:"Opción no válida. No se activó ninguna suscripción.");
83                  return "sinmetodo";
84          }
85
86          // Llama al método para procesar el pago, pasando el precio que calculamos
87          String metodoPago = procesarPago(precio);
88          // Si el pago no fue exitoso (metodoPago es null), indicamos que no se activó suscripción
89          if (metodoPago == null) {
90              System.out.println(x:"❌ El pago no se completó. No se activó la suscripción.");
91              return "sinmetodo";
92          }
93
94          // Obtenemos la fecha actual del sistema (hoy)
95          LocalDate hoy = LocalDate.now();
96          // Calculamos la fecha final sumando los meses de la membresía a la fecha de hoy
97          LocalDate fin = hoy.plusMonths(meses);
98          // Creamos un formato para mostrar fechas como "aaaa-mm-dd"
99          DateTimeFormatter fmt = DateTimeFormatter.ofPattern(pattern:"yyyy-MM-dd");
100
101         // Convertimos la fecha de inicio y la de fin a texto con el formato indicado
102         String fechaInicio = hoy.format(fmt);
103         String fechaFin = fin.format(fmt);
104
105         // Actualizamos el archivo de usuarios con la nueva información de suscripción
106         actualizarSuscripcion(usuario, contrasena, fechaInicio, fechaFin, tipo, metodoPago);
107         // Confirmamos al usuario que su membresía está activa y por cuánto tiempo
108         System.out.println("✅ " + tipo + " activada por $" + precio + ". Vigente hasta " + fechaFin);
109
110         // Devolvemos el método de pago elegido para que se pueda guardar o usar después
111         return metodoPago;
112     }
113
114     // Método que simula la parte de cobrar y pedir datos de la tarjeta al usuario
115     private static String procesarPago(double precio) {
116         // Mostramos un mensaje con el precio a pagar
117         System.out.println("\n💳 --- Pago de $" + precio + " ---");
118         // Mostramos las opciones de método de pago
119         System.out.println(x:"Seleccione método de pago:");
120         System.out.println(x:"1. Tarjeta de débito");
121         System.out.println(x:"2. Tarjeta de crédito");
122         System.out.println(x:"3. Tarjeta prepagada");
123         // Pedimos que el usuario elija una opción
124         System.out.print(s:"Opción: ");
125         // Leemos la opción elegida
126         String opcionPago = sc.nextLine().trim();
127
128         String tipoPago;
129         // Evaluamos la opción y guardamos el tipo de tarjeta elegida
130         switch (opcionPago) {
131             case "1": tipoPago = "Tarjeta de débito"; break;
```

Ilustración 6. Código de suscripción parte 2



```java
suscripcion.java 9+
suscripcion.java > Language Support for Java(TM) by Red Hat > suscripcion > gestionarNuevaCuenta(String, String)
13  public class suscripcion {
115     private static String procesarPago(double precio) {
132             case "2": tipoPago = "Tarjeta de crédito"; break;
133             case "3": tipoPago = "Tarjeta prepagada"; break;
134             default:
135                 // Si es opción inválida, cancelamos el pago y devolvemos null
136                 System.out.println(x:"Opción no válida. Cancelando pago...");
137                 return null;
138         }
139
140         // Solicitamos los datos básicos de la tarjeta (simulación)
141         System.out.println(x:"\n--- Ingrese los datos de la tarjeta ---");
142         System.out.print(s:"Nombre del titular: ");
143         String nombreTitular = sc.nextLine();
144
145         System.out.print(s:"Número de la tarjeta: ");
146         String numeroTarjeta = sc.nextLine();
147
148         System.out.print(s:"Fecha de vencimiento (MM/AA): ");
149         String fechaVencimiento = sc.nextLine();
150
151         System.out.print(s:"Número de seguridad (CVV): ");
152         String cvv = sc.nextLine();
153
154         System.out.print(s:"Código postal: ");
155         String codigoPostal = sc.nextLine();
156
157         // Verificamos que ningún dato esté vacío para que el pago pueda continuar
158         if (nombreTitular.isEmpty() || numeroTarjeta.isEmpty() || fechaVencimiento.isEmpty() || cvv.isEmpty() || codigoPostal.isEmpty()) {
159             // Si falta algún dato, mostramos mensaje y cancelamos
160             System.out.println(x:"❌ Datos incompletos. No se procesó el pago.");
161             return null;
162         }
163
164         // Indicamos que el pago fue procesado con éxito (simulación)
165         System.out.println(x:"✅ Pago procesado con éxito.");
166
167         // Retornamos el tipo de pago junto con el nombre del titular para guardarlo o mostrarlo
168         return tipoPago + " - Titular: " + nombreTitular;
169     }
170
171     // Método que actualiza la suscripción del usuario en el archivo usuarios.txt
172     // Recibe todos los datos necesarios para actualizar: usuario, contraseña, fechas, tipo y método de pago
173     private static void actualizarSuscripcion(String usuario, String contrasena, String fechaInicio, String fechaFin, String tipo, String metodoPago) {
174         // Variable para guardar los puntos actuales del usuario y no perderlos
175         int puntos = 0;
176         // Usamos un método externo (de la clase cw) para cargar la información del usuario actual
177         cw.Usuario u = cw.cargarUsuario(usuario, contrasena);
178         // Si el usuario existe, copiamos sus puntos actuales a la variable
179         if (u != null) {
180             puntos = u.puntos;
181         }
182         // Actualizamos el archivo usuarios.txt con la nueva información, manteniendo puntos y contraseña
183         cw.actualizarUsuario(usuario, contrasena, fechaInicio, fechaFin, tipo, metodoPago, puntos);
184     }
185 }
186
```

Ilustración 7. Código de suscripción parte 3

```java
// Importamos las clases necesarias para manejar archivos, fechas, entrada de usuario, etc.
import java.io.*; // Para manejar archivos, lectura y escritura
import java.nio.charset.StandardCharsets; // Para especificar codificación UTF-8 al leer/escribir archivos
import java.nio.file.*; // Para manejar rutas y operaciones avanzadas con archivos
import java.time.LocalDate; // Para manejar fechas sin tiempo (solo año, mes, día)
import java.time.format.DateTimeFormatter; // Para dar formato y parsear fechas
import java.util.List; // Para manejar listas (colecciones de líneas de archivos)
import java.util.Scanner; // Para leer entrada de usuario desde consola
// Declaramos la clase principal del programa
public class cw {
    // Constante con el nombre del archivo donde se guardan los usuarios
    static final String ARCHIVO_USUARIOS = "usuarios.txt";
    // Scanner para leer entrada de usuario desde consola
    static Scanner sc = new Scanner(System.in);
    // Clase interna para almacenar los datos de un usuario
    static class Usuario {
        String usuario;          // Nombre de usuario
        String contrasena;       // Contraseña del usuario
        String fechaInicio;      // Fecha de inicio de la suscripción
        String fechaFin;         // Fecha fin de la suscripción
        String tipoSuscripcion;  // Tipo de suscripción (ejemplo: "No suscrito", "Cancelada", etc)
        String metodoPago;       // Método de pago usado
        int puntos;              // Puntos acumulados del usuario (pueden ser usados para recompensas)
    }
    // Método principal que se ejecuta al iniciar el programa
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        // Bucle infinito para mostrar el menú hasta que el usuario decida salir
        while (true) {
            System.out.println("\n--- MENÚ ---");
            System.out.println("1. Crear usuario");    // Opción para registrar nuevo usuario
            System.out.println("2. Iniciar sesión");    // Opción para ingresar con usuario existente
            System.out.println("3. Salir");             // Opción para terminar programa
            System.out.print("Elige una opción: ");
            // Leemos la opción elegida y la evaluamos
            switch (sc.nextLine()) {
                case "1":
                    crearUsuario(); // Llama a método que crea un nuevo usuario
                    break;
                case "2":
                    iniciarSesion(); // Llama a método que permite iniciar sesión
                    break;
                case "3":
                    System.out.println("Saliendo del sistema...");
                    return; // Termina el programa
                default:
                    System.out.println("Opción no válida. Intenta de nuevo."); // Opción inválida
            }
        }
    }
    // Método para crear un nuevo usuario y guardarlo en archivo
    static void crearUsuario() {
        System.out.print("Ingresa un nombre de usuario: ");
        String usuario = sc.nextLine().trim(); // Leemos y limpiamos espacios
        if (usuario.isEmpty()) { // Validamos que no esté vacío
            System.out.println("❌ Debes poner algo en el nombre de usuario.");
            return; // Salimos si es inválido
        }
        System.out.print("Ingresa una contraseña: ");
        String contrasena = sc.nextLine().trim();
        if (contrasena.isEmpty()) { // Validamos que la contraseña no esté vacía
            System.out.println("❌ Debes poner algo en la contraseña.");
            return;
        }
        // Verificamos si el usuario ya existe para evitar duplicados
        if (usuarioExiste(usuario)) {
            System.out.println("❌ El usuario ya existe. Intenta con otro nombre.");
            return;
        }
        // Abrimos el archivo para agregar el nuevo usuario al final (modo append)
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(ARCHIVO_USUARIOS, append:true))) {
            // Guardamos datos separados por ";" en formato:
            // usuario;contraseña;fechaInicio;fechaFin;tipoSuscripcion;metodoPago;puntos
            // Al crear, las fechas quedan vacías y la suscripción como "No suscrito"
            writer.write(usuario + ";" + contrasena + ";;;" + "No suscrito" + ";" + "sinmetodo" + ";0");
            writer.newLine(); // Salto de línea para siguiente registro
            System.out.println("✅ Usuario creado correctamente. Estado: No suscrito.");
        } catch (IOException e) {
            System.out.println("❌ Error al guardar el usuario."); // Si hay problema con archivo
            return;
        }
```

Ilustración 8. Código de Cinema World parte 1

```java
cw.java > Java > cw > crearUsuario()
public class cw {
    static void crearUsuario() {
        // Aquí se llama a la gestión de suscripción (debes tener esa clase y método definidos)
        String metodoPago = suscripcion.gestionarNuevaCuenta(usuario, contrasena);
        if (metodoPago != null && !metodoPago.equalsIgnoreCase(anotherString:"sinmetodo")) {
            actualizarMetodoPago(usuario, metodoPago); // Actualiza método de pago en archivo
        }
    }
    // Método para verificar si un usuario ya existe en el archivo
    static boolean usuarioExiste(String usuarioBuscado) {
        try (BufferedReader reader = new BufferedReader(new FileReader(ARCHIVO_USUARIOS))) {
            String linea;
            while ((linea = reader.readLine()) != null) {
                String[] datos = linea.split(regex:";"); // Separa la línea en campos por ";"
                if (datos.length > 0 && datos[0].equals(usuarioBuscado)) {
                    return true; // Usuario encontrado
                }
            }
        } catch (IOException e) {
            // Si no existe el archivo, consideramos que no existe el usuario
        }
        return false; // Usuario no encontrado
    }
    // Método para iniciar sesión, validado usuario y contraseña
    static void iniciarSesion() {
        System.out.print(s:"Usuario: ");
        String usuario = sc.nextLine();
        System.out.print(s:"Contraseña: ");
        String contrasena = sc.nextLine();
        // Validación especial para administrador hardcodeado
        if (usuario.equals(anObject:"admin") && contrasena.equals(anObject:"2006")) {
            System.out.println(x:"¡Bienvenido Admin!");
            Admin.menuAdmin(); // Llama a menú especial de administrador (debes definir esta clase)
            return;
        }
        // Carga los datos del usuario si coincide usuario y contraseña
        Usuario u = cargarUsuario(usuario, contrasena);
        if (u == null) {
            System.out.println(x:"Usuario o contraseña incorrectos.");
            return;
        }
        LocalDate hoy = LocalDate.now(); // Fecha actual
        DateTimeFormatter fmt = DateTimeFormatter.ofPattern(pattern:"yyyy-MM-dd"); // Formato para fechas
        // Validamos que el usuario tenga fechas de suscripción válidas
        if (!u.fechaInicio.isEmpty() && !u.fechaFin.isEmpty()) {
            // Convertimos strings de fechas a objetos LocalDate para comparar
            LocalDate fechaInicio = LocalDate.parse(u.fechaInicio, fmt);
            LocalDate fechaFin = LocalDate.parse(u.fechaFin, fmt);
            // Detectamos si la fecha del sistema está adelantada (hoy < fechaInicio)
            if (hoy.isBefore(fechaInicio)) {
                System.out.println(x:"Error: La fecha del sistema está adelantada respecto a tu suscripción.");
                System.out.println(x:"Se eliminará tu cuenta para evitar fraudes o errores.");
                eliminarCuenta(u.usuario); // Borra cuenta por seguridad
                System.out.println(x:"Por favor, crea una nueva cuenta y suscríbete nuevamente.");
                return;
            }
            // Verificamos si la suscripción está activa (fechaInicio <= hoy <= fechaFin)
            boolean activo = (hoy.isAfter(fechaInicio) || hoy.isEqual(fechaInicio)) &&
                             (hoy.isBefore(fechaFin) || hoy.isEqual(fechaFin));
            if (!activo) { // Si no está activo, pedir renovar suscripción
                System.out.println(x:"Tu suscripción ha expirado o no es válida.");
                String nuevoMetodoPago = suscripcion.gestionarNuevaCuenta(u.usuario, u.contrasena);
                if (nuevoMetodoPago != null && !nuevoMetodoPago.equalsIgnoreCase(anotherString:"sinmetodo")) {
                    actualizarMetodoPago(u.usuario, nuevoMetodoPago);
                    System.out.println(x:"Suscripción activada, inicia sesión de nuevo.");
                }
                return;
            }
            // Suscripción válida: mostramos mensaje de bienvenida
            System.out.println("Inicio de sesión exitoso. ¡Bienvenido, " + u.usuario + "!");
            System.out.println("Tu suscripción está activa hasta: " + u.fechaFin + " (Tipo: " + u.tipoSuscripcion + ")");
            // Llamamos al reproductor de películas para iniciar sesión y actualizar puntos
            int nuevosPuntos = reproductorpeliculas.iniciar(u.usuario, u.puntos, u.fechaInicio, u.fechaFin, u.tipoSuscripcion, u.metodoPago, u.contrasena);
            // Actualizamos los datos del usuario en el archivo, incluyendo nuevos puntos
            actualizarUsuario(u.usuario, u.contrasena, u.fechaInicio, u.fechaFin, u.tipoSuscripcion, u.metodoPago, nuevosPuntos);
        } else {
            // Caso donde no hay fechas válidas
            if (u.tipoSuscripcion.equalsIgnoreCase(anotherString:"Cancelada") || u.tipoSuscripcion.equalsIgnoreCase(anotherString:"No suscrito")) {
                System.out.println("Tu suscripción está " + u.tipoSuscripcion + ". Debes suscribirte para usar el reproductor.");
                String nuevoMetodoPago = suscripcion.gestionarNuevaCuenta(u.usuario, u.contrasena);
                if (nuevoMetodoPago != null && !nuevoMetodoPago.equalsIgnoreCase(anotherString:"sinmetodo")) {
```

Ilustración 9.Código de Cinema World parte 2

```java
cw.java 1 ●
cw.java > Java > cw > iniciarSesion()
10  public class cw {
103     static void iniciarSesion() {
162                 }
163             } else {
164                 System.out.println(x:"Error en las fechas de suscripción. Por favor, contacta soporte.");
165             }
166         }
167     }
168     // Método para eliminar completamente la cuenta del usuario (lo borra del archivo)
169     static void eliminarCuenta(String usuario) {
170         Path path = Paths.get(ARCHIVO_USUARIOS);          // Ruta del archivo original
171         Path tempPath = Paths.get(first:"usuarios_temp.txt");     // Archivo temporal para reescribir
172         try {
173             // Leemos todas las líneas del archivo original
174             List<String> lineas = Files.readAllLines(path, StandardCharsets.UTF_8);
175             // Abrimos el archivo temporal para escribir
176             try (BufferedWriter bw = Files.newBufferedWriter(tempPath, StandardCharsets.UTF_8)) {
177                 for (String linea : lineas) {
178                     String[] datos = linea.split(regex:";"); // Separamos línea por campos
179                     // Solo copiamos al temporal las líneas que no coinciden con el usuario a borrar
180                     if (datos.length > 0 && !datos[0].equals(usuario)) {
181                         bw.write(linea);
182                         bw.newLine();
183                     }
184                 }
185             }
186             // Reemplazamos archivo original por el temporal (sin el usuario eliminado)
187             Files.move(tempPath, path, StandardCopyOption.REPLACE_EXISTING);
188             System.out.println(x:"Cuenta eliminada correctamente.");
189         } catch (IOException e) {
190             System.out.println("Error eliminando cuenta: " + e.getMessage());
191         }
192     }
193     // Método para cargar un usuario desde archivo si usuario y contraseña coinciden
194     static Usuario cargarUsuario(String usuarioBuscado, String contrasenaBuscado) {
195         try (BufferedReader br = new BufferedReader(new FileReader(ARCHIVO_USUARIOS))) {
196             String linea;
197             while ((linea = br.readLine()) != null) {
198                 System.out.println("Leyendo línea: '" + linea + "'"); // DEBUG: mostrar línea leída
199                 if (linea.trim().isEmpty()) continue; // Ignorar líneas vacías
200                 String[] datos = linea.split(regex:";"); // Dividir línea en campos separados por ";"
201                 if (datos.length < 2) {
202                     System.out.println("Línea ignorada por formato incorrecto (menos de 2 campos): " + linea);
203                     continue; // Saltar línea si formato no es correcto
204                 }
205                 // Si usuario y contraseña coinciden con lo buscado, creamos objeto Usuario
206                 if (datos[0].equals(usuarioBuscado) && datos[1].equals(contrasenaBuscado)) {
207                     Usuario u = new Usuario();
208                     u.usuario = datos[0];
209                     u.contrasena = datos[1];
210                     u.fechaInicio = datos.length > 2 ? datos[2] : "";
211                     u.fechaFin = datos.length > 3 ? datos[3] : "";
212                     u.tipoSuscripcion = datos.length > 4 ? datos[4] : "No suscrito";
213                     u.metodoPago = datos.length > 5 ? datos[5] : "sinmetodo";
214                     try {
215                         u.puntos = Integer.parseInt(datos.length > 6 ? datos[6] : "0");
216                     } catch (NumberFormatException e) {
217                         u.puntos = 0; // Si no es número, asignar 0 puntos
218                     }
219                     return u; // Retornamos usuario cargado
220                 }
221             }
222         } catch (IOException e) {
223             System.out.println("Error leyendo archivo usuarios: " + e.getMessage());
224         }
225         return null; // No se encontró usuario o hubo error
226     }
227     // Método para actualizar los datos de un usuario en el archivo
228     public static void actualizarUsuario(String usuario, String contrasena, String fechaInicio, String fechaFin,
229                                          String tipoSuscripcion, String metodoPago, int puntos) {
230         Path path = Paths.get(ARCHIVO_USUARIOS);        // Archivo original
231         Path tempPath = Paths.get(first:"usuarios_temp.txt");  // Archivo temporal para reescribir
232         boolean encontrado = false;                     // Para saber si usuario fue encontrado y actualizado
233         try {
234             List<String> lineas = Files.readAllLines(path, StandardCharsets.UTF_8);
235             // Abrimos temporal para escritura
236             try (BufferedWriter bw = Files.newBufferedWriter(tempPath, StandardCharsets.UTF_8)) {
237                 for (String linea : lineas) {
238                     String[] datos = linea.split(regex:";");
239                     // Si línea corresponde al usuario que queremos actualizar
240                     if (datos.length > 0 && datos[0].equals(usuario)) {
```

Ilustración 10.Código de Cinema World parte 3

```
cw.java 1 ●
cw.java > Java > cw > actualizarUsuario(String usuario, String contrasena, String fechaInicio, String fechaFin, String tipoSuscripcion, String metodoPago, int puntos)
10  public class cw {
228     public static void actualizarUsuario(String usuario, String contrasena, String fechaInicio, String fechaFin,
241                         // Formamos línea nueva con datos actualizados
242                         String nuevaLinea = usuario + ";" + contrasena + ";" + fechaInicio + ";" + fechaFin + ";" +
243                                 tipoSuscripcion + ";" + metodoPago + ";" + puntos;
244                         bw.write(nuevaLinea);
245                         encontrado = true;
246                     } else {
247                         // Copiamos línea sin cambio
248                         bw.write(linea);
249                     }
250                     bw.newLine();
251                 }
252             }
253             // Intentamos reemplazar el archivo original por el temporal
254             try {
255                 Files.move(tempPath, path, StandardCopyOption.REPLACE_EXISTING);
256             } catch (IOException e) {
257                 System.out.println("No se pudo reemplazar el archivo directamente: " + e.getMessage());
258                 System.out.println(x:"Intentando eliminar archivo original y mover temporal...");
259                 // Si falla, borramos archivo original y movemos temporal manualmente
260                 try {
261                     Files.delete(path);
262                     Files.move(tempPath, path);
263                     System.out.println(x:"Archivo actualizado correctamente después de eliminar el original.");
264                 } catch (IOException ex) {
265                     System.out.println("Error al eliminar o mover archivo: " + ex.getMessage());
266                 }
267             }
268             if (!encontrado) { // Si no encontró usuario para actualizar
269                 System.out.println(x:"Usuario no encontrado al actualizar.");
270                 Files.deleteIfExists(tempPath); // Borra archivo temporal si existe
271             }
272         } catch (IOException e) {
273             System.out.println("Error actualizando usuario: " + e.getMessage());
274             try {
275                 Files.deleteIfExists(tempPath);
276             } catch (IOException ex) {
277                 // Ignorar error al borrar temp
278             }
279         }
280     }
281     // Método para actualizar solo el método de pago de un usuario en el archivo
282     static void actualizarMetodoPago(String usuario, String metodoPago) {
283         String tempFile = "usuarios_temp.txt"; // Archivo temporal
284         try {
285             List<String> lineas = Files.readAllLines(Paths.get(ARCHIVO_USUARIOS));
286             // Abrimos archivo temporal para escribir las líneas actualizadas
287             try (BufferedWriter writer = new BufferedWriter(new FileWriter(tempFile))) {
288                 for (String linea : lineas) {
289                     String[] datos = linea.split(regex:";");
290                     // Si línea corresponde al usuario, actualizamos método de pago
291                     if (datos.length > 0 && datos[0].equals(usuario)) {
292                         String contrasena = datos.length > 1 ? datos[1] : "";
293                         String fechaInicio = datos.length > 2 ? datos[2] : "";
294                         String fechaFin = datos.length > 3 ? datos[3] : "";
295                         String tipoSuscripcion = datos.length > 4 ? datos[4] : "No suscrito";
296                         String puntos = datos.length > 6 ? datos[6] : "0";
297                         // Reescribimos línea con método de pago actualizado
298                         writer.write(usuario + ";" + contrasena + ";" + fechaInicio + ";" + fechaFin + ";" +
299                                 tipoSuscripcion + ";" + metodoPago + ";" + puntos);
300                     } else {
301                         // Copiamos línea sin cambio
302                         writer.write(linea);
303                     }
304                     writer.newLine();
305                 }
306             }
307             // Reemplazamos archivo original por el temporal con cambio
308             Files.delete(Paths.get(ARCHIVO_USUARIOS));
309             Files.move(Paths.get(tempFile), Paths.get(ARCHIVO_USUARIOS));
310         } catch (IOException e) {
311             System.out.println("Error actualizando método de pago: " + e.getMessage());
312         }
313     }
314 }
315
```

Ilustración 11. Código de Cinema World parte 4

5. Conclusión

En conclusión, este documento ha presentado una descripción completa y detallada del proyecto de programación desarrollado en Java sobre un entorno Ubuntu accesible remotamente, siguiendo los lineamientos y estándares establecidos por el IEEE 830 para especificación de requisitos de software. La aplicación desarrollada representa una integración efectiva de conceptos fundamentales de programación, administración de sistemas operativos y redes, enfocándose en la captura y validación segura de datos relacionados con métodos de pago, dentro de un contexto educativo y profesional. Se ha detallado el propósito del documento como una guía tanto para desarrolladores como para evaluadores, estableciendo un marco sólido para la comprensión, implementación y mantenimiento del software, con una arquitectura modular que facilita futuras ampliaciones. El ámbito del sistema queda definido como una aplicación de consola, portable y sencilla, que no depende de bases de datos ni interfaces gráficas, enfocándose en el manejo correcto de datos y la interacción segura con el usuario. Además, se identificaron y describieron en detalle los requisitos funcionales y no funcionales, las restricciones de diseño, las características de los usuarios previstos, y las condiciones de entorno necesarias para su funcionamiento, junto con un análisis de las posibles mejoras y ampliaciones que pueden implementarse para convertir el proyecto en una solución más robusta y completa. Finalmente, se establecieron aspectos técnicos complementarios, como la configuración de herramientas externas (X2Go, GitHub), protocolos de red básicos, y buenas prácticas de seguridad y mantenimiento del entorno, asegurando que el sistema pueda ser utilizado y evaluado de manera confiable y efectiva. Este trabajo, por tanto, constituye una base sólida para la formación y desarrollo profesional en las áreas de programación, administración de sistemas y redes, brindando un recurso integral y bien documentado que puede ser adaptado y expandido para múltiples propósitos y escenarios futuros.

6. Referencias apa

1. Canonical Ltd. (s.f.). Documentación oficial de Ubuntu. Recuperado de https://help.ubuntu.com/

2. Comunidad Ubuntu España. (s.f.). Documentación en español de Ubuntu. Recuperado de https://wiki.ubuntu.com/SpanishDocumentation

3. Oracle. (s.f.). Java: Tecnología para el desarrollo de aplicaciones. Recuperado de https://www.java.com/es/

4. JavaHispano. (s.f.). Documentación y tutoriales de Java. Recuperado de https://www.javahispano.org/documentacion/

5. Red Hat, Inc. (s.f.). Documentación de Red Hat Enterprise Linux 9. Recuperado de https://access.redhat.com/documentation/es-es/red_hat_enterprise_linux/9/

6. Mozilla Developer Network. (s.f.). Guía de seguridad en redes. Recuperado de https://developer.mozilla.org/es/docs/Web/Security

7. Cisco Systems, Inc. (s.f.). Fundamentos de redes. Recuperado de https://www.cisco.com/c/es_mx/solutions/small-business/resource-center/networking-basics.html

8. Microsoft Docs. (s.f.). Conceptos básicos de redes. Recuperado de https://learn.microsoft.com/es-es/windows-server/networking/

9. Instituto Nacional de Ciberseguridad (INCIBE). (s.f.). Guías y recursos sobre seguridad en redes. Recuperado de https://www.incibe.es/protege-tu-empresa/guias-y-recursos

10. Aulaclic. (s.f.). Curso básico de redes informáticas. Recuperado de https://www.aulaclic.es/redes/index.htm