UNIVERSIDAD EUROPEA DEL ATLÁNTICO

GRADO EN

Ingeniería Informática

PROGRAMACIÓN II

PROYECTO FINAL

TRANSFORMACIÓN DIGITAL PARA CONTADORES DE GAS INTELIGENTES

Trabajo realizado por

DANIEL ROLDÁN RÁBAGO

CARLOS ALEXIS CASADO TAMAYO

CAYETANO CASTILLO RUIZ

Profesor

ELDER BOL

SANTANDER – 09, Junio, 2023

ÍNDICE

INTRODUCCIÓN

Este proyecto consta de un software que permite a los clientes de una comercializadora de gas acceder y registrar las lecturas de sus contadores de gas a través de fotografías. Dichas lecturas se asocian de manera automática al cliente y a un periodo mensual, facilitando el cálculo correspondiente al suministro consumido por periodo.

Para este producto mínimo viable (PMV) se ha considerado trabajar la simulación de la última etapa en el proceso, partiendo de que disponemos los datos sobre una base de datos y los datos son descargados en formato Json para manipularlos y hacer el software funcional con los requisitos requeridos.

Entre las diversas características que ofrece, destacan las siguientes:

- Lectura automatizada mediante archivos cargados en el sistema.
- Precio por unidad de consumo ajustable por la empresa suministradora.
- Obtención de los precios mediante un registro en línea (fallback).
- Comparativa de consumo entre diferentes rangos de tiempo; se destaca la hora y fecha de mayor consumo, generando consigo un patrón estadístico a la disponibilidad del usuario.

Todo esto, permite haber conseguido digital el TFG de un compañero de la universidad del grado de ingeniería organización industrial. Su misión era el poder hacer un contador de gas inteligente al detectar que podría solucionar reducción de costes para las comercializadoras y otros beneficios interrelacionados con el control de sus clientes.

OBJETIVO DEL PMV

Este proyecto realizado para asignatura de programación II tiene vertientes diferentes respecto a los objetivos de haberlo llevado a cabo, por un lado lo mencionado anteriormente en la introducción. Digitalizar la metodología o la lógica y la funcionalidad que solicita el estudiante del TFG para realizar una simulación.

El otro objetivo, adquirir por el equipo Gasify de la asignatura los conocimientos de programación orientada a objetos y desarrollar habilidades y competencias para realizar proyectos colaborativos en un entorno dinámico y bajo unos requisitos.

PUNTO DE PARTIDA

Nuestro punto de partida es la necesidad de transformar y modernizar el proceso de lectura y facturación de contadores de gas tradicionales. Hasta ahora, este proceso ha sido manual y propenso a errores, lo que genera ineficiencias y dificultades para las comercializadoras. Al integrar nuestra solución a los contadores existentes, las comercializadoras podrán acceder a una lectura precisa y en tiempo real de los consumos de gas, simplificando así el proceso de facturación.

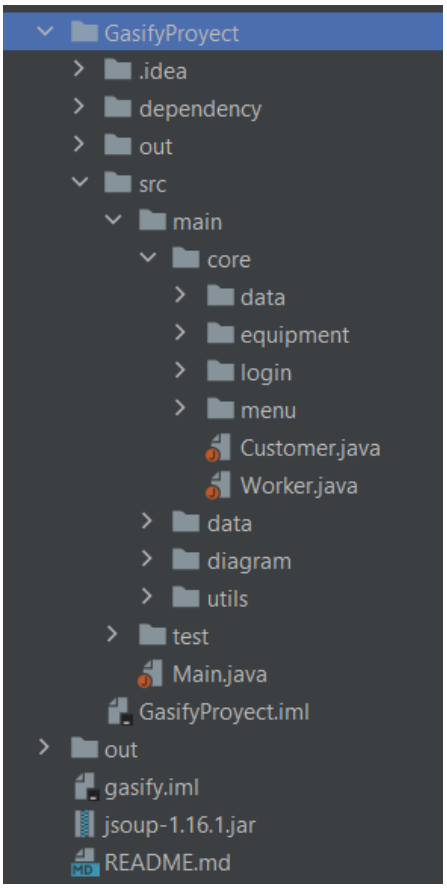
A continuación se muestra una fotografía de cómo proceden las comercializadoras a recopilar el dato de lectura, siendo tomada en un portal de Santander. Posteriormente, es facturado por contraste de esos valores con sus estimaciones estadísticas, no facturando nunca el consumo real del periodo considerado.

PROGRAMACIÓN ORIENTADA A OBJETOS

La Programación Orientada a Objetos (POO) es un paradigma de programación que se basa en la creación y manipulación de objetos para resolver problemas. En lugar de centrarse en las funciones o procedimientos, la POO se centra en los objetos, que son entidades que combinan datos y comportamiento.
La POO proporciona una forma estructurada y modular de desarrollar software, fomentando la reutilización de código, la modularidad, el mantenimiento y la flexibilidad. Al centrarse en los objetos y su interacción, la POO facilita la comprensión y el diseño de sistemas complejos, permitiendo un desarrollo eficiente y escalable.

ARQUITECTURA DEL CÓDIGO

Para que nuestro PMV sea claro y se encuentre bien estructurado dentro del proyecto, hemos procedido a realizar una arquitectura en base al código de la siguiente manera:



Dentro de src como se muestra en la figura anterior están los diversos paquetes que alojan

las clases que componen este proyecto. De modo que permite disponer de lo que llamamos arquitectura de código.

CLASES CORE

Las clases en este proyecto simulan a una plantilla que define las propiedades y comportamientos de cada objeto para su correcto funcionamiento, todos ellos integrados y relacionados entre sí, para conseguir el objetivo final.

Billing

La clase Billing es una clase que representa los datos de facturación de un cliente en el sistema. Esta clase pertenece al *paquete main.core.data* y se utiliza para almacenar y proporcionar acceso a información relevante para la facturación mensual.

Atributos:

- idReport: Un entero que representa el identificador del informe de facturación.
- idCustomer: Una cadena de texto que indica el identificador del cliente asociado a la facturación.
- idGasMeter: Una cadena de texto que indica el identificador del medidor de gas asociado a la facturación.
- idPlc: Una cadena de texto que indica el identificador del dispositivo PLC asociado a la facturación.
- idDataSim: Una cadena de texto que indica el identificador de la tarjeta SIM de datos asociada a la facturación.
- firstDate: Una cadena de texto que representa la fecha de inicio del período de facturación.
- secondDate: Una cadena de texto que representa la fecha de fin del período de facturación.
- firstValue: Un entero que indica el valor inicial para el período de facturación.
- secondValue: Un entero que indica el valor final para el período de facturación.

Constructores:

- Billing(int idReport, String idCustomer, String idGasMeter, String idPlc, String idDataSim, String firstDate, String secondDate, int firstValue, int secondValue): Este constructor crea un objeto Billing con los valores proporcionados para inicializar sus atributos.

Métodos de acceso:

- getIdReport(): Retorna el valor del identificador del informe de facturación.
- setIdReport(int idReport): Establece el valor del identificador del informe de facturación.
- getIdCustomer(): Retorna el identificador del cliente asociado a la facturación.
- setIdCustomer(String idCustomer): Establece el identificador del cliente asociado a la facturación.
- getIdGasMeter(): Retorna el identificador del medidor de gas asociado a la facturación.
- setIdGasMeter(String idGasMeter): Establece el identificador del medidor de gas asociado a la facturación.
- getIdPlc(): Retorna el identificador del dispositivo PLC asociado a la facturación.
- setIdPlc(String idPlc): Establece el identificador del dispositivo PLC asociado a la facturación.
- getIdDataSim(): Retorna el identificador de la tarjeta SIM de datos asociada a la facturación.
- setIdDataSim(String idDataSim): Establece el identificador de la tarjeta SIM de datos asociada a la facturación.
- getFirstDate(): Retorna la fecha de inicio del período de facturación.
- setFirstDate(String firstDate): Establece la fecha de inicio del período de facturación.
- getSecondDate(): Retorna la fecha de fin del período de facturación.

- ➢ setSecondDate(String secondDate): Establece la fecha de fin del período de facturación.

- ➢ getFirstValue(): Retorna el valor inicial para el período de facturación.

- ➢ setFirstValue(int firstValue): Establece el valor inicial para el período de facturación.

- ➢ getSecondValue(): Retorna el valor final para el período de facturación.

- ➢ setSecondValue(int secondValue): Establece el valor final para el período de facturación.

La clase Billing es responsable de almacenar y gestionar la información relacionada con la facturación mensual de un cliente en el sistema. Los atributos y métodos de acceso proporcionan las funcionalidades necesarias para interactuar con estos datos.

Data

La clase Data es una clase que pertenece al *paquete main.core.data* y se utiliza para realizar búsquedas y filtrar datos relacionados con clientes, trabajadores y facturación en el sistema. Esta clase también hereda de la clase ParseData e implementa la interfaz Finders, lo que le permite realizar análisis y buscar datos específicos.

Atributos:

- ➢ fullDate: Un objeto DateValidator utilizado para validar y formatear fechas completas en el formato "dd/MM/yyyy".

- ➢ monthYear: Un objeto DateValidator utilizado para validar y formatear fechas de mes y año en el formato "MM/yyyy".

- ➢ format: Un objeto DateValidator utilizado para formatear fechas.

Constructores:

- ➢ Data(): Constructor que inicializa los objetos fullDate y monthYear con los formatos de fecha correspondientes.

Métodos de acceso y búsqueda:

- ➢ findSimDataByID(String customerId): Busca y devuelve una lista de objetos DataSim filtrados por el identificador del cliente proporcionado.

- ➢ findWorkerByID(String id): Busca y devuelve una lista de objetos Worker filtrados por el identificador proporcionado.

- ➢ findWorkerByUser(String user): Busca y devuelve una lista de objetos Worker filtrados por el nombre de usuario proporcionado.
- ➢ findWorkerByEmail(String email): Busca y devuelve una lista de objetos Worker filtrados por el correo electrónico proporcionado.

- ➢ findPriceByDate(String period): Busca y devuelve el valor del precio correspondiente a un período de facturación especificado.

- ➢ findBillingByUser(String customerID): Busca y devuelve una lista de objetos Billing filtrados por el identificador del cliente proporcionado.

- ➢ findGasMeter(String gasMeterId): Busca y devuelve una lista de objetos GasMater filtrados por el identificador del medidor de gas proporcionado.

- ➢ findSim(String simId): Busca y devuelve una lista de objetos DataSim filtrados por el identificador de la tarjeta SIM proporcionada.

- ➢ findPlc(String plcId): Busca y devuelve una lista de objetos Plc filtrados por el identificador del dispositivo PLC proporcionado.

- ➢ findCustomerByName(String name): Busca y devuelve una lista de objetos Customer filtrados por el nombre proporcionado.

- ➢ findCustomerByID(String ID): Busca y devuelve una lista de objetos Customer filtrados por el identificador proporcionado.

- ➢ findCustomerByNumber(String numberPhone): Busca y devuelve una lista de objetos Customer filtrados por el número de teléfono proporcionado.

La clase Data proporciona métodos para buscar y filtrar datos relacionados con clientes, trabajadores y facturación en el sistema. Estos métodos utilizan los datos almacenados en diferentes listas, como customerData(), workerData(), billingData(), etc., para realizar las búsquedas y devolver los resultados filtrados.

Finders

La interfaz Finders es una interfaz definida en el *paquete main.core.data* y proporciona una lista de métodos para buscar y filtrar datos en el sistema. Los métodos de esta interfaz toman ciertos parámetros y devuelven listas de objetos específicos. Aquí está la descripción de los métodos de la interfaz Finders:

- ➢ findSimDataByID(String customerId): Busca y devuelve una lista de objetos DataSim filtrados por el identificador del cliente proporcionado.

- ➢ findWorkerByID(String id): Busca y devuelve una lista de objetos Worker filtrados por el identificador proporcionado.

- ➢ findWorkerByUser(String user): Busca y devuelve una lista de objetos Worker filtrados por el nombre de usuario proporcionado.

- ➢ findWorkerByEmail(String email): Busca y devuelve una lista de objetos Worker filtrados por el correo electrónico proporcionado.

- ➢ findPriceByDate(String period): Busca y devuelve el valor del precio correspondiente a un período de facturación especificado.

- ➢ findBillingByUser(String customerID): Busca y devuelve una lista de objetos Billing filtrados por el identificador del cliente proporcionado.

- ➢ findGasMeter(String gasMeterId): Busca y devuelve una lista de objetos GasMater filtrados por el identificador del medidor de gas proporcionado.

- ➢ findSim(String simId): Busca y devuelve una lista de objetos DataSim filtrados por el identificador de la tarjeta SIM proporcionada.

- ➢ findPlc(String plcId): Busca y devuelve una lista de objetos Plc filtrados por el identificador del dispositivo PLC proporcionado.

- ➢ findCustomerByName(String name): Busca y devuelve una lista de objetos Customer filtrados por el nombre proporcionado.

- ➢ findCustomerByID(String ID): Busca y devuelve una lista de objetos Customer filtrados por el identificador proporcionado.

- ➢ findCustomerByNumber(String numberPhone): Busca y devuelve una lista de objetos Customer filtrados por el número de teléfono proporcionado.

Estos métodos son utilizados para buscar y filtrar datos en el sistema, devolviendo listas de objetos que cumplen con los criterios de búsqueda especificados. La implementación de estos métodos se realiza en las clases que implementan la interfaz Finders, como la clase Data que hemos revisado previamente.

Prices

La clase Prices del *paquete main.core.data* representa los precios para un período específico. Esta clase contiene información sobre el año, período y valor de los precios.

Constructor:

- ➢ Prices(int year, String period, int value): Crea una nueva instancia de la clase Prices con los parámetros especificados. Toma el año, período y valor de los precios.

Métodos:

- ➢ int getYear(): Obtiene el año de los precios.
- ➢ void setYear(int year): Establece el año de los precios.
- ➢ String getPeriod(): Obtiene el período de los precios.
- ➢ void setPeriod(String period): Establece el período de los precios.
- ➢ float getValue(): Obtiene el valor de los precios.
- ➢ void setValue(int value): Establece el valor de los precios.

DataSim

La clase DataSim del *paquete main.core.equipment* es una representación de una tarjeta SIM de datos utilizada en equipos de comunicación. Esta clase contiene tres atributos:

Atributos:

- ➢ idDataSim: una cadena que representa el identificador único de la tarjeta SIM.
- ➢ numberDataSim: una cadena que representa el número de la tarjeta SIM.
- ➢ serviceCompany: una cadena que indica la empresa de servicios asociada a la tarjeta SIM.

Constructor:

- ➢ DataSim: constructor que permite crear una nueva instancia de la clase DataSim especificando el identificador, el número de la tarjeta y la empresa de servicios.

Métodos:

- ➢ getIdDataSim: método que devuelve el identificador de la tarjeta SIM.
- ➢ setIdDataSim: método que establece el identificador de la tarjeta SIM.
- ➢ getNumberDataSim: método que devuelve el número de la tarjeta SIM.

- ➢ setNumberDataSim: método que establece el número de la tarjeta SIM.

- ➢ getServiceCompany: método que devuelve la empresa de servicios asociada a la tarjeta SIM.
- ➢ setServiceCompany: método que establece la empresa de servicios asociada a la tarjeta SIM.

- ➢ toStringDataSim: método que devuelve una representación en forma de cadena de la instancia de la clase DataSim, incluyendo el identificador, el número de la tarjeta y la empresa de servicios.

Esta clase es utilizada para gestionar la información relacionada con las tarjetas SIM de datos en equipos de comunicación, permitiendo su creación, acceso y modificación.

Customer

La clase Customer es una clase pública perteneciente al paquete main.core que representa a un cliente en el sistema.

Atributos:

- ➢ idCustomer: una cadena que representa el identificador del cliente.

- ➢ nameCustomer: una cadena que representa el nombre del cliente.

- ➢ numberPhone: una cadena que representa el número de teléfono del cliente.
- ➢ address: una cadena que representa la dirección del cliente.

- ➢ plcId: una cadena que representa el identificador del PLC (Controlador Lógico Programable) asociado al cliente.

- ➢ simId: una cadena que representa el identificador de la tarjeta SIM asociada al cliente.

- ➢ gasMaterID: una cadena que representa el identificador del medidor de gas asociado al cliente.

Constructores:

- ➢ public Customer(String idCustomer, String nameCustomer, String numberPhone, String address): constructor que recibe los datos del cliente y los asigna a los atributos correspondientes.

- ➢ public Customer(): constructor sin argumentos.

Métodos:

- public void setNameCustomer(String nameCustomer): método que establece el nombre del cliente.
- public String getNameCustomer(): método que devuelve el nombre del cliente.
- public String getIdCustomer(): método que devuelve el identificador del cliente.
- public void setIdCustomer(String idCustomer): método que establece el identificador del cliente.
- public String getNumberPhone(): método que devuelve el número de teléfono del cliente.
- public void setNumberPhone(String numberPhone): método que establece el número de teléfono del cliente.
- public String getAddress(): método que devuelve la dirección del cliente.
- public void setAddress(String address): método que establece la dirección del cliente.
- public String getPlcId(): método que devuelve el identificador del PLC asociado al cliente.
- public void setPlcId(String plcId): método que establece el identificador del PLC asociado al cliente.
- public String getSimId(): método que devuelve el identificador de la tarjeta SIM asociada al cliente.
- public void setSimId(String simId): método que establece el identificador de la tarjeta SIM asociada al cliente.
- public String getGasMaterID(): método que devuelve el identificador del medidor de gas asociado al cliente.
- public void setGasMaterID(String gasMaterID): método que establece el identificador del medidor de gas asociado al cliente.
- @Override public String toString(): método que devuelve una representación de cadena de texto de la instancia de la clase Customer, mostrando los valores de todos los atributos.

La clase Customer representa la información de un cliente y proporciona métodos para acceder y modificar los datos de dicho cliente. Además, también incluye una implementación del método toString() para obtener una representación legible del objeto Customer.

Worker

La clase Worker es una subclase de la clase User y representa a un trabajador en el sistema. La clase pertenece al paquete principal (main).

Atributos:

- idEmployee: una cadena que representa el identificador del empleado.
- name: una cadena que representa el nombre del trabajador.
- nameCompany: una cadena que representa el nombre de la empresa a la que pertenece el trabajador.
- area: una cadena que representa el área en la que trabaja el empleado.

Constructores:

- public Worker(String idEmployee, String name, String nameCompany, String area): constructor que recibe los datos del trabajador y los asigna a los atributos correspondientes.
- public Worker(): constructor sin argumentos.

Métodos:

- @Override public String toString(): método que devuelve una representación de cadena de texto de la instancia de la clase Worker, mostrando los valores de los atributos idEmployee, email, nameCompany y department heredados de la clase User.
- public String getIdEmployee(): método que devuelve el identificador del empleado.
- public void setIdEmployee(String idEmployee): método que establece el identificador del empleado.
- public String getNameCompany(): método que devuelve el nombre de la empresa a la que pertenece el trabajador.

- ➢ public void setNameCompany(String nameCompany): método que establece el nombre de la empresa a la que pertenece el trabajador.

- ➢ public String getArea(): método que devuelve el área en la que trabaja el empleado.

- ➢ public void setArea(String area): método que establece el área en la que trabaja el empleado.

La clase Worker hereda los atributos y métodos de la clase User, que probablemente represente a un usuario genérico en el sistema. Al heredar de la clase User, la clase Worker también tiene acceso a los atributos email y department definidos en la clase User, y los incluye en la representación de cadena de texto devuelta por el método toString().

GasMater

La clase GasMater del *paquete main.core.equipment* representa un contador de gas.

Atributos:

- ➢ idGasMeter: una cadena que representa el identificador único del medidor de gas.

- ➢ brand: una cadena que indica la marca del medidor de gas.

- ➢ ref: una cadena que representa la referencia del medidor de gas.

- ➢ ean13: una cadena que indica el código EAN13 del medidor de gas.

- ➢ serie: una cadena que representa la serie del medidor de gas.

- ➢ nameModel: una cadena que indica el nombre del modelo del medidor de gas.

- ➢ readInstallationDay: un entero que representa el día de instalación del medidor de gas.

- ➢ plcId: una cadena que indica el identificador PLC asociado al medidor de gas.

Constructor:

- ➢ GasMater: constructor que permite crear una nueva instancia de la clase GasMater especificando el identificador, la marca, la referencia, el código EAN13, la serie, el día de instalación y el identificador PLC.

Métodos:

- getIdGasMater: método que devuelve el identificador del medidor de gas.
- setIdGasMater: método que establece el identificador del medidor de gas.
- getBrand: método que devuelve la marca del medidor de gas.
- setBrand: método que establece la marca del medidor de gas.
- getRef: método que devuelve la referencia del medidor de gas.
- setRef: método que establece la referencia del medidor de gas.
- getEan13: método que devuelve el código EAN13 del medidor de gas.
- setEan13: método que establece el código EAN13 del medidor de gas.
- getSerie: método que devuelve la serie del medidor de gas.
- setSerie: método que establece la serie del medidor de gas.
- setNameModel: método que establece el nombre del modelo del medidor de gas.
- getReadInstallationDay: método que devuelve el día de instalación del medidor de gas.
- setReadInstallationDay: método que establece el día de instalación del medidor de gas.
- toStringGasMater: método que devuelve una representación en forma de cadena de la instancia de la clase GasMater, incluyendo el identificador, la marca, la referencia, el código EAN13, la serie y el día de instalación.

Esta clase es utilizada para gestionar la información relacionada con los medidores de gas en equipos de equipamiento, permitiendo su creación, acceso y modificación.

Plc

La clase Plc representa un controlador lógico programable (PLC, por sus siglas en inglés) utilizado en equipos de equipamiento.

Atributos:

- ➢ idPlc: una cadena que representa el identificador único del PLC.
- ➢ maker: una cadena que indica el fabricante del PLC.
- ➢ model: una cadena que representa el modelo del PLC.
- ➢ idDataSim: una cadena que indica el identificador de la tarjeta SIM asociada al PLC.

Constructor:

- ➢ Plc: constructor que permite crear una nueva instancia de la clase Plc especificando el identificador, el fabricante, el modelo y el identificador de la tarjeta SIM.

Métodos:

- ➢ getMaker: método que devuelve el fabricante del PLC.
- ➢ setMaker: método que establece el fabricante del PLC.
- ➢ getModel: método que devuelve el modelo del PLC.
- ➢ setModel: método que establece el modelo del PLC.
- ➢ getIdPlc: método que devuelve el identificador del PLC.
- ➢ setIdPlc: método que establece el identificador del PLC.
- ➢ toStringPlc: método que devuelve una representación en forma de cadena de la instancia de la clase Plc, incluyendo el identificador, el fabricante y el modelo.

Esta clase se utiliza para gestionar la información relacionada con los controladores lógicos programables en equipos de equipamiento, permitiendo su creación, acceso y modificación.

Manager

La clase Manager del paquete main.core.login extiende la clase FileWriter y representa un administrador en el sistema de inicio de sesión.

Atributos:

- employee: un objeto de tipo Worker que representa al empleado asociado al administrador.
- data: un objeto de tipo Data utilizado para acceder y gestionar los datos de los empleados.

Constructor:

- Manager: constructor que crea una nueva instancia de la clase Manager y también inicializa el objeto data.

Métodos:

- newIdEmployee: método privado que genera un nuevo identificador único para un empleado.

- userCreate: método que crea un nuevo usuario registrando un empleado con un nombre de usuario, una contraseña, un correo electrónico y un departamento específico. Este método utiliza el método writeWorkersData() para escribir los datos del empleado en el archivo correspondiente.

- login: método que verifica las credenciales de inicio de sesión de un usuario. Recibe un nombre de usuario y una contraseña como parámetros y busca en los datos de los empleados si existe un empleado con esas credenciales. Si se encuentra una coincidencia, el método devuelve true, de lo contrario, devuelve false.

La clase Manager se utiliza para gestionar las operaciones de registro de usuarios y verificación de credenciales en el sistema de inicio de sesión. Además, tiene la capacidad de generar identificadores únicos para nuevos empleados y almacenar los datos correspondientes en el archivo adecuado.

User

La clase User del paquete main.core.login es una clase abstracta que representa un usuario genérico en el sistema de inicio de sesión.

Atributos:

- name: una cadena que representa el nombre del usuario.

- password: una cadena que almacena la contraseña del usuario.

Constructor:

- User: constructor que crea una nueva instancia de la clase User con un nombre y una contraseña especificados.

- ➢ User: constructor vacío utilizado para la creación de instancias sin parámetros.

Métodos:

- ➢ getName: método que devuelve el nombre del usuario.
- ➢ setName: método que establece el nombre del usuario.
- ➢ getPassword: método que devuelve la contraseña del usuario.
- ➢ setPassword: método que establece la contraseña del usuario.

La clase User es una clase abstracta, lo que significa que no se puede instanciar directamente, sino que debe ser subclase de una clase concreta que proporciona implementaciones específicas de los métodos abstractos definidos en User. Esta clase sirve como una base para la definición de diferentes tipos de usuarios en el sistema de inicio de sesión, como administradores, empleados u otros roles específicos.
Al heredar de la clase User, las subclases pueden agregar funcionalidad adicional y personalizada según sus necesidades, como métodos específicos del rol de usuario o atributos adicionales.

MenuOptions

La clase MenuOptions, perteneciente al *paquete main.core.menu*, proporciona opciones de menú y funcionalidades adicionales para la interacción con el sistema. Esta clase extiende la clase Menus y se encarga de manejar las opciones seleccionadas por el usuario.

Atributos:

- ➢ userInput: un objeto Scanner utilizado para capturar la entrada del usuario.
- ➢ search: un objeto Data utilizado para realizar búsquedas de datos.
- ➢ find: un objeto ProcessData utilizado para procesar los datos encontrados.
- ➢ register: un objeto Manager utilizado para el registro y autenticación de usuarios.

Métodos:

- ➢ principalMenu: este método muestra el menú principal y maneja las opciones seleccionadas por el usuario. El usuario puede elegir iniciar sesión o registrarse.
- ➢ login: este método maneja el inicio de sesión del usuario. Solicita al usuario que ingrese su nombre de usuario y contraseña, y verifica si las credenciales son correctas.

- registerOption: este método maneja la opción de registro de usuario. Solicita al usuario que ingrese un nombre de usuario, correo electrónico, contraseña y departamento. Luego, verifica la información ingresada y crea un nuevo usuario si es válido.

- loginOption: este método muestra el menú de opciones después de iniciar sesión. El usuario puede realizar acciones como buscar por nombre, buscar por ID o trabajar con un cliente específico.

- searchByName: este método realiza una búsqueda de clientes por nombre y muestra los resultados si se encuentran coincidencias.

- searchByID: este método realiza una búsqueda de clientes por ID y muestra los resultados si se encuentran coincidencias.

- workWithUserOption: este método maneja la opción de trabajar con un cliente específico. Permite realizar acciones relacionadas con ese cliente, como buscar facturas por fecha o generar una factura.

- findBillingByDate: este método busca facturas de un cliente específico en un período de tiempo específico y muestra los resultados si se encuentran coincidencias.

- findBillingByRangeDate: este método busca facturas de un cliente específico en un rango de fechas y muestra los resultados si se encuentran coincidencias.

- generateInvoiceOption: este método genera una factura para un cliente específico en un período de tiempo específico.

Menus

La clase Menus es una clase abstracta perteneciente al paquete main.core.menu que proporciona métodos para mostrar diferentes menús y presentar información de manera tabular en el sistema.

Atributos:

- table: un objeto de la clase TableList utilizado para representar los datos en forma de tabla.

- find: un objeto ProcessData utilizado para procesar datos y realizar cálculos.

- decimalFix: un objeto BigDecimal utilizado para formatear números decimales.

La clase Menus proporciona los siguientes métodos protegidos:

- principal: este método muestra el menú principal con opciones numeradas, como "Login", "Register" y "Exit". Utiliza el objeto table para imprimir la tabla en la consola.

- registerCheck: este método muestra una tabla con los datos de registro, como el nombre de usuario, correo electrónico y departamento ingresados por el usuario.

- logged: este método muestra el menú de opciones después de iniciar sesión, como "Find Customer by Name", "Find Customer by ID", "Work with a Customer" y "Exit".

- actionsCustomer: este método muestra el menú de acciones para un cliente específico, como "Find billing by Date", "Find billing by range date", "Generate invoice" y "Exit".

- customerList: este método muestra una lista de clientes en forma de tabla. Recibe una lista de objetos Customer y utiliza el objeto table para imprimir la tabla en la consola.

- billSpecificMonth: este método muestra los detalles de facturación de un cliente para un mes específico en forma de tabla. Recibe una lista de objetos Billing y los datos de identificación del cliente y el mes. Utiliza el objeto table para imprimir la tabla en la consola.

- billRangeMonth: este método muestra los detalles de facturación de un cliente en un rango de fechas en forma de tabla. Recibe una lista de cadenas info que contiene los datos de facturación. Calcula el consumo total y el importe total en el rango de fechas y los muestra en la tabla utilizando el objeto table.

Además, la clase Menus también tiene un método privado prettyDecimal que recibe un número decimal como argumento y devuelve un objeto BigDecimal formateado con dos decimales utilizando el objeto decimalFix.

CLASES UTILS

DateValidator

La clase DateValidator es una clase del paquete main.utils y se utiliza para validar fechas y realizar operaciones relacionadas con el formato de fecha.

Atributo:

- dateFormat: una cadena que representa el formato de fecha utilizado para validar y formatear fechas.

Constructores:

- public DateValidator(String dateFormat): constructor que recibe el formato de fecha y lo asigna al atributo dateFormat.

- public boolean isValid(String dateStr): método que recibe una cadena de texto que representa una fecha y verifica si es válida según el formato de fecha especificado. El método utiliza la clase DateFormat y SimpleDateFormat para realizar la validación. Si la fecha es válida, devuelve true; de lo contrario, devuelve false.

- public String[] dateFormat(String date): método que recibe una cadena de texto que representa una fecha y la divide en partes utilizando "/" como delimitador. El método devuelve un arreglo de cadenas que contiene las partes de la fecha dividida.

La clase DateValidator utiliza el formato de fecha proporcionado durante la inicialización para realizar las validaciones y operaciones de formato de fecha necesarias. Es importante destacar que la clase utiliza la biblioteca java.text.SimpleDateFormat para analizar y formatear las fechas.

FileReader

La clase FileReader es una clase abstracta del paquete main.utils y se utiliza para leer archivos y obtener su contenido como una cadena de texto. La clase tiene los siguientes elementos:

Constructor protegido: la clase tiene un constructor protegido que no acepta argumentos. Esto significa que no se pueden crear instancias directas de la clase FileReader, sino que se espera que se herede y se utilice a través de subclases.

Método:

public String readAsString(String path) throws IOException: este método recibe una cadena de texto que representa la ruta de un archivo y lee su contenido como una cadena de texto. Utiliza la clase Files y Paths de la biblioteca estándar de Java para leer los bytes del archivo

y luego los convierte en una cadena utilizando el constructor de String. Si ocurre algún error durante la lectura del archivo, se lanza una excepción IOException.
La clase FileReader se utiliza para leer archivos y obtener su contenido como una cadena de texto.

FileWriter

La clase FileWriter es una clase abstracta del paquete main.utils que extiende de la clase FileReader e implementa la interfaz Writer. Se utiliza para escribir datos en archivos JSON. La clase tiene los siguientes elementos:

Atributos privados:

- gson: un objeto de la clase Gson que se utiliza para serializar y deserializar objetos JSON.
- data: una cadena de texto que almacena el contenido del archivo JSON.
- writer: un objeto de la clase java.io.FileWriter que se utiliza para escribir en el archivo JSON.

Constructor:

El constructor de la clase FileWriter no acepta argumentos y se encarga de inicializar el atributo gson utilizando un objeto de la clase Gson con formato de impresión mejorado.

Métodos para escribir datos en archivos JSON:

- public void writeBillingData(Billing bill): este método recibe un objeto Billing y escribe los datos en el archivo JSON CustomerBilling.json. Primero lee el contenido actual del archivo utilizando el método readAsString heredado de la clase FileReader. A continuación, utiliza la biblioteca Gson para convertir la cadena JSON en una lista de objetos Billing. Agrega el nuevo objeto bill a la lista, obtiene un objeto java.io.FileWriter para el archivo y utiliza gson.toJson para escribir la lista actualizada en el archivo. Finalmente, cierra el escritor.

Métodos similares: writeCustomerData(), writeSimData(), writeGasMeterData(), writePLCData(), writeWorkersData () y writePriceData () son métodos similares al método writeBillingData (), pero cada uno se utiliza para escribir datos en un archivo JSON específico.

InvoiceGeneratorHTTP

La clase InvoiceGeneratorHTTP es una clase del paquete main.utils que se utiliza para generar una factura utilizando una solicitud HTTP.

Atributos:

- API: una cadena de texto que representa la URL de la API utilizada para generar la factura.
- url: un objeto de la clase URL que representa la dirección URL de la API.
- connection: un objeto de la clase HttpURLConnection que se utiliza para establecer la conexión con la API.
- body: una matriz de bytes que almacena los datos del cuerpo de la solicitud HTTP.
- base: un objeto de la clase StringBuilder que se utiliza para construir el JSON base de la factura.
- responseBody: un objeto de la clase StringBuilder que almacena la respuesta de la API.

Constructor:

- El constructor de la clase InvoiceGeneratorHTTP no acepta argumentos. En su implementación, inicializa los atributos url y connection con la URL proporcionada en el atributo API. También inicializa los atributos responseBody, base con objetos StringBuilder vacíos.

Métodos:

- public String generateInvoice(String... info): este método recibe una serie de información y genera una factura. El método itera sobre los elementos de la matriz info y construye un JSON base utilizando los números impares como títulos de las propiedades y los números pares como los resultados de las propiedades. El resultado se envía al método connectHTTP para enviar una solicitud HTTP a la API y obtener la respuesta.
- public String connectHTTP(String input): este método se encarga de establecer una conexión HTTP con la API y enviar la solicitud. Utiliza la configuración adecuada para una solicitud POST y establece el tipo de contenido como "application/json". El cuerpo de la solicitud se obtiene como bytes a partir del parámetro input. Luego, se envía la solicitud y se obtiene la respuesta de la API. La respuesta se almacena en el objeto responseBody y se retorna como una cadena de texto.

Es importante mencionar que cualquier excepción que ocurra durante la conexión HTTP se imprime en la salida estándar.

ParseData

La clase ParseData es una clase abstracta que extiende de FileReader e implementa la interfaz Parser. Pertenece al paquete main.utils y se utiliza para recopilar archivos JSON y analizarlos en una lista de objetos.

Atributos:

- gson: un objeto de la clase Gson que se utiliza para analizar y convertir los archivos JSON.

- data: una cadena de texto que almacena los datos leídos de los archivos JSON.

Constructor:

- El constructor de la clase ParseData no acepta argumentos. En su implementación, inicializa el atributo gson con un nuevo objeto Gson.

Métodos:

- public List<Billing> billingData(): este método lee el archivo JSON "CustomerBilling.json" y lo analiza en una lista de objetos Billing. Retorna la lista resultante. Si ocurre alguna excepción durante la lectura o el análisis del archivo, se imprime un mensaje de error.

- public List<Customer> customerData(): este método lee el archivo JSON "customers.json" y lo analiza en una lista de objetos Customer. Retorna la lista resultante. Si ocurre alguna excepción durante la lectura o el análisis del archivo, se imprime un mensaje de error.

- public List<DataSim> simData(): este método lee el archivo JSON "dataSims.json" y lo analiza en una lista de objetos DataSim. Retorna la lista resultante. Si ocurre alguna excepción durante la lectura o el análisis del archivo, se imprime un mensaje de error.

- public List<GasMater> gasMaterData(): este método lee el archivo JSON "gasMaters.json" y lo analiza en una lista de objetos GasMater. Retorna la lista resultante. Si ocurre alguna excepción durante la lectura o el análisis del archivo, se imprime un mensaje de error.

- ➢ public List<Plc> plcData() throws IOException: este método lee el archivo JSON "Plcs.json" y lo analiza en una lista de objetos Plc. Retorna la lista resultante. Si ocurre alguna excepción durante la lectura o el análisis del archivo, se propaga una excepción de tipo IOException.

- ➢ public List<Worker> workerData(): este método lee el archivo JSON "workers.json" y lo analiza en una lista de objetos Worker. Retorna la lista resultante. Si ocurre alguna excepción durante la lectura o el análisis del archivo, se imprime un mensaje de error.

- ➢ public List<Prices> pricesData(): este método lee el archivo JSON "Prices.json" y lo analiza en una lista de objetos Prices. Retorna la lista resultante. Si ocurre alguna excepción durante la lectura o el análisis del archivo, se imprime un mensaje de error.

Es importante mencionar que cualquier excepción que ocurra durante la lectura o el análisis de los archivos JSON se imprime en la salida estándar.

Parser

La interfaz Parser define un conjunto de métodos para obtener datos de diferentes tipos a partir de archivos JSON. Pertenece al paquete main.utils

Métodos:

- ➢ List<Billing> billingData(): Este método retorna una lista de objetos Billing obtenidos a partir de un archivo JSON.

- ➢ List<Customer> customerData(): Este método retorna una lista de objetos Customer obtenidos a partir de un archivo JSON.

- ➢ List<DataSim> simData(): Este método retorna una lista de objetos DataSim obtenidos a partir de un archivo JSON.

- ➢ List<GasMater> gasMaterData(): Este método retorna una lista de objetos GasMater obtenidos a partir de un archivo JSON.

- ➢ List<Plc> plcData(): Este método retorna una lista de objetos Plc obtenidos a partir de un archivo JSON.

- ➢ List<Worker> workerData(): Este método retorna una lista de objetos Worker obtenidos a partir de un archivo JSON.

- ➢ List<Prices> pricesData(): Este método retorna una lista de objetos Prices obtenidos a partir de un archivo JSON.

La interfaz Parser proporciona un contrato para obtener datos específicos a partir de archivos JSON. Cualquier clase que implemente esta interfaz debe proporcionar una implementación de estos métodos.

ProcessData

La clase ProcessData es una subclase de la clase Data. Pertenece al paquete main.process

Atributos:

- ➢ DateValidator fullDate: Un objeto DateValidator utilizado para validar fechas en formato "dd/MM/yyyy".

- ➢ DateValidator monthYear: Un objeto DateValidator utilizado para validar fechas en formato "MM/yyyy".

Constructores:

- ➢ ProcessData(): Constructor que inicializa los objetos fullDate y monthYear con los formatos de fecha correspondientes.

Métodos:

- ➢ List<String> allUsage(String customerID, String firstDate, String secondDate): Este método recibe un ID de cliente, una primera fecha y una segunda fecha como parámetros. Verifica si las fechas están en el formato correcto y luego filtra las facturas mensuales correspondientes al cliente y al rango de fechas especificado. A partir de los resultados obtenidos, se genera una lista de cadenas que representan el uso, incluyendo el mes, el ID del cliente, el ID del contador de gas, el cálculo del precio y el valor del mes. Si las fechas no son válidas, se imprime un mensaje de error y se devuelve null.

- ➢ List<Billing> findSpecificMonthBill(String customerId, String date): Este método busca las facturas correspondientes a un cliente y un mes específico. Recibe como parámetros el ID del cliente y la fecha en formato "MM/yyyy". El método valida el formato de la fecha y luego busca las facturas que coincidan con el cliente y la fecha proporcionados. Devuelve una lista de facturas correspondientes a los criterios de búsqueda. Si no se encuentran facturas, se imprime un mensaje de error y se devuelve null.

- ➢ List<Billing> monthlyBill(String customerId, String firstDate, String secondDate): Este método filtra las facturas mensuales de un cliente en un rango de fechas específico. Recibe como parámetros el ID del cliente, la primera fecha y la segunda fecha en formato "dd/MM/yyyy". El método valida el formato de las fechas y luego filtra las

facturas que pertenecen al cliente y que tienen una fecha de inicio dentro del rango especificado. Devuelve una lista de facturas que cumplen con los criterios de búsqueda. Si no se encuentran facturas, se imprime un mensaje de error y se devuelve null.

- ➢ float priceCalc(String customerId, String period): Este método calcula el precio de una factura mensual para un cliente y un período específicos. Recibe como parámetros el ID del cliente y el período en formato "MM/yyyy". El método utiliza el método findSpecificMonthBill para obtener las facturas correspondientes al cliente y al período especificados. Luego, calcula el precio basado en la diferencia de valores y el precio encontrado. Devuelve el precio calculado. Si no se encuentra el precio o no se encuentran facturas, se devuelve -1.

- ➢ float diffValues(String customerID, String date): Este método calcula la diferencia de valores entre una factura mensual para un cliente y un mes específicos. Recibe como parámetros el ID del cliente y la fecha en formato "MM/yyyy". El método utiliza el método findBillingByUser para obtener las facturas correspondientes al cliente. Luego, busca la factura que coincida con el cliente y la fecha especificados y calcula la diferencia entre los valores de inicio y fin. Devuelve la diferencia de valores. Si la fecha no es válida, se imprime un mensaje de error y se devuelve -1.

TableList

La clase TableList es una implementación de una tabla en Java que permite organizar y mostrar datos en forma tabular. La tabla se compone de columnas y cada columna tiene una descripción. A continuación, se describen las principales características y métodos de la clase:

- La clase tiene constantes que definen los caracteres especiales utilizados para dibujar la tabla, como las líneas horizontales y verticales, esquinas y cruces.

- La clase almacena las descripciones de las columnas en un arreglo de cadenas llamado descriptions.

- La tabla en sí se representa mediante una lista de arreglos de cadenas, donde cada arreglo representa una fila y cada elemento del arreglo representa el valor de una celda.

- La clase mantiene un arreglo llamado tableSizes que almacena la longitud máxima de cada columna para asegurar un formato adecuado.

- Se pueden aplicar diferentes opciones de formato a la tabla. Por ejemplo, se puede especificar un comparador para ordenar los datos por una columna determinada,

alinear el contenido de una columna específica y ajustar el espaciado entre columnas.

- La clase proporciona métodos para agregar filas a la tabla, filtrar los datos según un patrón de búsqueda y habilitar el uso de caracteres Unicode para una representación visual más sofisticada.

- El método print() se utiliza para mostrar la tabla en la consola. Este método formatea los datos y utiliza los caracteres especiales definidos para dibujar los bordes y las líneas de la tabla.

En resumen, la clase TableList brinda una funcionalidad completa para crear, manipular y mostrar tablas de datos en Java, con opciones de formato personalizables.

Writer

La interfaz Writer define un conjunto de métodos que se utilizan para escribir datos en diferentes tipos de objetos en un archivo. A continuación, se presenta una descripción de cada uno de los ***métodos***:

- ➢ writeBillingData(Billing bill): Este método se encarga de escribir los datos de una factura (Billing) en un archivo. Puede lanzar una excepción de tipo IOException si ocurre algún error durante el proceso de escritura.

- ➢ writeCustomerData(Customer customer): Este método se utiliza para escribir los datos de un cliente (Customer) en un archivo. Puede lanzar una excepción de tipo IOException si hay problemas durante la escritura.

- ➢ writeSimData(DataSim sim): Este método se encarga de escribir los datos de una tarjeta SIM (DataSim) en un archivo. Puede lanzar una excepción de tipo IOException si ocurre algún error durante la escritura.

- ➢ writeGasMeterData(GasMeter gas): Este método se utiliza para escribir los datos de un medidor de gas (GasMeter) en un archivo. Puede lanzar una excepción de tipo IOException si hay problemas durante la escritura.

- ➢ writePLCData(Plc plc): Este método se encarga de escribir los datos de un controlador lógico programable (Plc) en un archivo. Puede lanzar una excepción de tipo IOException si ocurre algún error durante la escritura.

- ➢ writeWorkersData(Worker employee): Este método se utiliza para escribir los datos de un trabajador (Worker) en un archivo. Puede lanzar una excepción de tipo IOException si hay problemas durante la escritura.

- ➢ writePriceData(Prices price): Este método se encarga de escribir los datos de un precio (Prices) en un archivo. Puede lanzar una excepción de tipo IOException si ocurre algún error durante la escritura.

En resumen, la interfaz Writer define una serie de métodos que deben ser implementados por clases concretas para realizar la escritura de diferentes tipos de datos en un archivo. Cada método se encarga de escribir los datos específicos de un objeto en particular y puede lanzar una excepción de tipo IOException en caso de fallos en la escritura.

EXCEPCIONES

Se han creado excepciones propias para gestionar posibles errores en los códigos anteriores. Estas excepciones se han diseñado para encapsular y representar situaciones excepcionales específicas que pueden ocurrir durante la ejecución de los métodos.

Pueden verse en el diagrama UML.

TEST UNITARIOS

Se han creado pruebas unitarias utilizando JUnit 5.8 para validar los métodos creados y garantizar la confiabilidad del código.
Esto ayuda a identificar posibles errores y asegurar que el código funcione correctamente.

DATA

En este paquete está subdividido en dos subpaquetes:

- img : se dispone de la imagen de lo que es un contador y muestra la forma de ofrecer la lectura por sistema tradicional.
- json: recoge los archivos que se cargan al sistema por medio de la clase ParseData

Los archivos que se encuentran en el paquete json son los siguientes:

- ➢ CustomerBilling: simula el archivo Json que podría ser descargado de la base de datos donde quedarían registradas las lecturas por medio del PLC y la DataSim sobre el contador haciendo uso de un periférico como podría ser una cámara y un led para generar un entorno de luz ideal para su posterior tratamiento de la imagen.

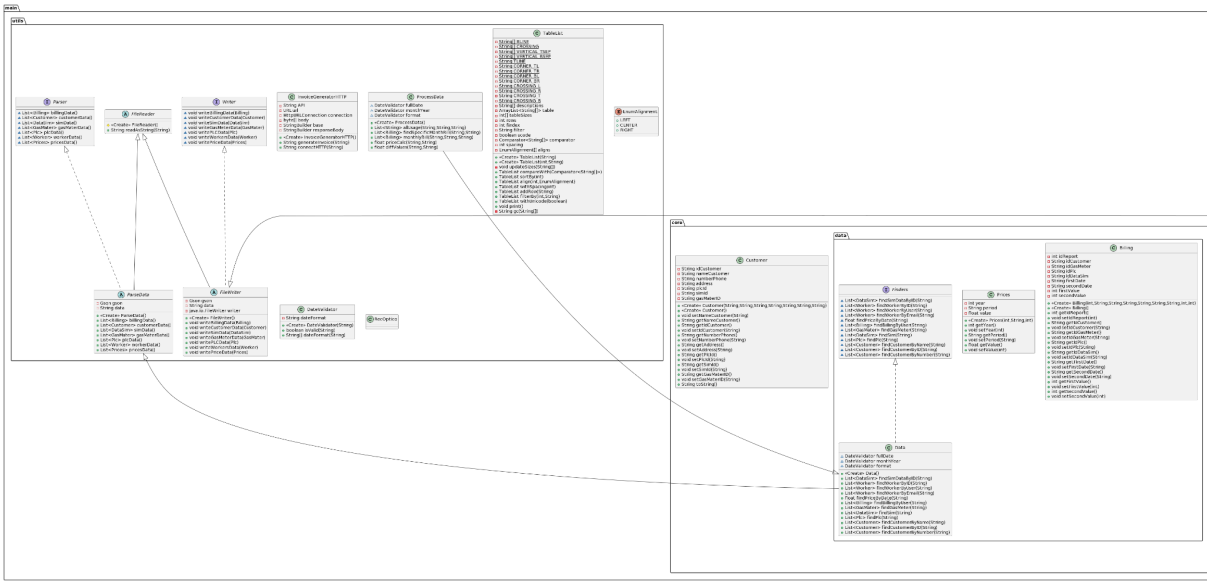
- ➢ customer: carga unos clientes predefinidos al sistema.

- ➢ dataSim: carga tarjetas sim al sistema.

- ➢ gasMater: carga los contadores al sistema.

- Plc: carga los controladores del contador al sistema.
- Prices: carga los precios al sistema siendo obtenidos en el promedio de una base de datos open source.
- workers: carga trabajadores de la compañia comercializadora al sistema.

DIAGRAMA UML



EJECUTAR EL PMV EN CONSOLA

Para ello, la clase Main nos sirve de archivo que lanza el programa por medio de la instancia de la clase MenuOptions y haciendo uso del método de la misma, principalMenu().