## **Documentación del Sistema de Gestión de Gasolineras**

### **1. Descripción Detallada de Funcionalidades de Clases y Métodos**

A continuación, se detalla cada clase proporcionada:

#### **Paquete modelo**

##### **Clase Gasolinera**

* **Propósito de la Clase:** Representa una entidad de gasolinera individual, almacenando su información de ubicación y los precios de diferentes tipos de combustible. Esta clase es inmutable una vez creada (los precios y la ubicación no cambian después de la instanciación).
* **Atributos:**
  + - provincia: String: Provincia donde se ubica la gasolinera.
  + - municipio: String: Municipio donde se ubica la gasolinera.
  + - localidad: String: Localidad donde se ubica la gasolinera.
  + - cp: String: Código postal de la gasolinera.
  + - direccion: String: Dirección completa de la gasolinera.
  + - gasolina95: double: Precio de la Gasolina 95.
  + - gasolina98: double: Precio de la Gasolina 98.
  + - diesel: double: Precio del Diésel.
  + - dieselPlus: double: Precio del Diésel Plus.
* **Constructores:**
  + + Gasolinera(provincia: String, municipio: String, localidad: String, cp: String, direccion: String, gasolina95: double, gasolina98: double, diesel: double, dieselPlus: double)
    - **Propósito:** Construye una nueva instancia de Gasolinera.
    - **Parámetros:**
      * provincia: Nombre de la provincia (no nulo, no vacío).
      * municipio: Nombre del municipio (no nulo, no vacío).
      * localidad: Nombre de la localidad (no nulo, no vacío).
      * cp: Código postal (puede ser nulo o vacío si no es esencial).
      * direccion: Dirección de la gasolinera (no nula, no vacía).
      * gasolina95: Precio para Gasolina 95 (se espera double, Double.NaN si no disponible).
      * gasolina98: Precio para Gasolina 98 (se espera double, Double.NaN si no disponible).
      * diesel: Precio para Diésel (se espera double, Double.NaN si no disponible).
      * dieselPlus: Precio para Diésel Plus (se espera double, Double.NaN si no disponible).
    - **Pre-condiciones:** Los parámetros provincia, municipio, localidad y direccion no deben ser nulos ni cadenas vacías (después de trim()). Los precios deben ser valores numéricos válidos o Double.NaN.
    - **Post-condiciones:** Se crea un nuevo objeto Gasolinera con los datos proporcionados.
    - **Excepciones Thrown:** IllegalArgumentException si alguno de los parámetros de cadena obligatorios es nulo o vacío.
* **Métodos Públicos:**
  + + getProvincia(): String
  + + getMunicipio(): String
  + + getLocalidad(): String
  + + getCp(): String
  + + getDireccion(): String
  + + getGasolina95(): double
  + + getGasolina98(): double
  + + getDiesel(): double
  + + getDieselPlus(): double  
    - **Propósito:** Métodos de acceso para obtener los valores de los atributos correspondientes.
    - **Retorno:** El valor del atributo respectivo.
  + + getPrecioCombustible(tipo: String): double  
    - **Propósito:** Obtiene el precio de un tipo de combustible específico.
    - **Parámetros:**
      * tipo: El tipo de combustible (e.g., "gasolina95", "diesel"). Se normaliza a minúsculas internamente.
    - **Retorno:** El precio (double) del combustible especificado. Devuelve Double.NaN si el tipo es nulo, vacío, o no corresponde a ninguno de los tipos conocidos ("gasolina95", "gasolina98", "diesel", "dieselplus").
    - **Pre-condiciones:** Ninguna explícita más allá de la validez del objeto.
    - **Post-condiciones:** El precio del combustible es devuelto sin modificar el estado del objeto.
  + + toString(): String  
    - **Propósito:** Devuelve una representación en cadena del objeto Gasolinera, útil para depuración o visualización simple.
    - **Retorno:** Una cadena que incluye la dirección, localidad, provincia y los precios de todos los combustibles. Los precios no disponibles (Double.NaN) se muestran como "N/A".

##### **Clase LectorCSV**

* **Propósito de la Clase:** Proporciona funcionalidad para leer datos de gasolineras desde un archivo CSV y convertirlos en una lista de objetos Gasolinera. Es una clase de utilidad con métodos estáticos.
* **Métodos Públicos Estáticos:**
  + + leerGasolineras(rutaArchivo: String): List<Gasolinera>
    - **Propósito:** Lee un archivo CSV que contiene datos de gasolineras y los transforma en una lista de objetos Gasolinera.
    - **Parámetros:**
      1. rutaArchivo: La ruta completa al archivo CSV.
    - **Retorno:** Una List<Gasolinera> poblada con los datos del archivo. Devuelve una lista vacía si el archivo no contiene datos válidos o está vacío (después de la cabecera).
    - **Pre-condiciones:**
      1. rutaArchivo no debe ser nulo ni vacío.
      2. El archivo especificado por rutaArchivo debe existir y ser legible.
      3. El archivo CSV debe tener una línea de cabecera que se omite.
      4. Las líneas de datos deben seguir un formato esperado con un número mínimo de campos (al menos 17 para acceder a los índices de precios esperados).
      5. Los campos de precios deben ser parseables a double según la lógica de parsePrecio.
    - **Post-condiciones:** Se devuelve una lista de objetos Gasolinera. Las líneas con datos incorrectos o insuficientes se omiten y se registra una advertencia en System.err.
    - **Excepciones Thrown:**
      1. CSVLecturaException:
         * Si rutaArchivo es nulo o vacío.
         * Si ocurre FileNotFoundException al intentar abrir el archivo.
         * Si ocurre IOException durante la lectura del archivo.
         * Si una línea no tiene el número mínimo de campos esperados.
         * Si faltan datos esenciales para construir un objeto Gasolinera (e.g., provincia vacía).
    - **Algoritmo Overview:**
      1. Valida la ruta del archivo.
      2. Abre el archivo usando BufferedReader y FileReader.
      3. Lee y descarta la primera línea (cabecera).
      4. Itera sobre las líneas restantes: a. Divide la línea en campos usando una expresión regular que maneja comas dentro de campos entrecomillados. b. Verifica que haya suficientes campos. Si no, lanza CSVLecturaException. c. Extrae los campos correspondientes a provincia, municipio, localidad, CP, dirección y los diferentes precios de combustible. d. Convierte los campos de precio usando el método parsePrecio. e. Intenta crear un nuevo objeto Gasolinera. f. Si la creación es exitosa, lo añade a la lista. g. Si ocurre NumberFormatException o IllegalArgumentException al procesar una línea, imprime un mensaje de error y continúa con la siguiente línea (la línea problemática se omite).
      5. Devuelve la lista de gasolineras.
* **Métodos Privados Estáticos:**
  + - parsePrecio(campo: String, indice: int): double
    - **Propósito:** Convierte una representación de precio en formato String (proveniente del CSV) a un valor double. Maneja distintos formatos (con coma o punto decimal, o enteros representando milésimas).
    - **Parámetros:**
      1. campo: El valor del campo de precio como String.
      2. indice: El índice de la columna (usado para mensajes de error, aunque no en la implementación actual del println).
    - **Retorno:** El precio como double. Devuelve Double.NaN si el campo es nulo, vacío, "null", o si no se puede convertir a número.
    - **Algoritmo Overview:**
      1. Si campo es nulo, devuelve Double.NaN.
      2. Limpia el campo: quita espacios, comillas.
      3. Si el campo procesado está vacío o es "null", devuelve Double.NaN.
      4. Reemplaza comas (,) por puntos (.) para normalizar el separador decimal.
      5. Intenta convertir a double: a. Si contiene un punto (.), usa Double.parseDouble(). b. Si no contiene punto, asume que es un entero representando milésimas, lo parsea como long y divide por 1000.0. c. Si NumberFormatException ocurre, imprime una advertencia y devuelve Double.NaN.

##### **Clase CSVLecturaException**

* **Propósito de la Clase:** Una excepción personalizada que hereda de RuntimeException. Se utiliza para señalar errores específicos ocurridos durante la lectura o procesamiento de un archivo CSV. Al ser RuntimeException, no requiere declaración throws obligatoria, pero indica problemas serios en la operación de lectura.
* **Constructores:**
  + + CSVLecturaException(message: String)
    - **Propósito:** Crea una nueva excepción con un mensaje descriptivo.
    - **Parámetros:**
      * message: El mensaje de error.
  + + CSVLecturaException(message: String, cause: Throwable)
    - **Propósito:** Crea una nueva excepción con un mensaje descriptivo y una causa original.
    - **Parámetros:**
      * message: El mensaje de error.
      * cause: La excepción original que causó este error.

#### **Paquete logica**

##### **Clase GestorGasolineras**

* **Propósito de la Clase:** Gestiona una colección de objetos Gasolinera. Proporciona funcionalidades para obtener información agregada (como listas de provincias), filtrar gasolineras según criterios, calcular estadísticas de precios y encontrar la gasolinera más barata para un tipo de combustible y provincia específicos.
* **Atributos:**
  + - gasolineras: List<Gasolinera>: La lista de objetos Gasolinera que gestiona.
  + - df: static final DecimalFormat: Formateador para mostrar precios con tres decimales.
* **Constructores:**
  + + GestorGasolineras(gasolineras: List<Gasolinera>)
    - **Propósito:** Crea una instancia del gestor con una lista inicial de gasolineras.
    - **Parámetros:**
      1. gasolineras: La lista de Gasolinera a gestionar.
    - **Post-condiciones:** El gestor se inicializa con la lista proporcionada. Si la lista es null, se inicializa con una lista vacía y se emite una advertencia a System.err.
* **Métodos Públicos:**
  + + obtenerProvincias(): List<String>  
    - **Propósito:** Obtiene una lista única, ordenada alfabéticamente, de todas las provincias presentes en la colección de gasolineras.
    - **Retorno:** List<String> de provincias. Devuelve una lista vacía si no hay gasolineras o ninguna tiene información de provincia válida.
    - **Filtra:** Provincias nulas o vacías (isBlank()).
  + + obtenerTiposCombustibleDisponibles(): List<String>  
    - **Propósito:** Devuelve una lista predefinida de los tipos de combustible que la aplicación considera disponibles para filtrado y estadísticas.
    - **Retorno:** Una List<String> con los nombres de los tipos de combustible (e.g., "Gasolina95", "Gasolina98", "Diesel", "DieselPlus").
    - **Nota:** Esta lista es actualmente fija y no se deriva dinámicamente de los datos.
  + + filtrarGasolineras(tipo: String, provincia: String): List<Gasolinera>  
    - **Propósito:** Filtra la lista de gasolineras basándose en un tipo de combustible y una provincia. Para que una gasolinera sea incluida por tipo de combustible, debe tener un precio válido (mayor que 0 y no NaN) para ese tipo.
    - **Parámetros:**
      1. tipo: El tipo de combustible a filtrar (e.g., "gasolina95"). Si es nulo, vacío o "todos", no se filtra por tipo.
      2. provincia: La provincia a filtrar. Si es nula, vacía o "todas", no se filtra por provincia.
    - **Retorno:** Una nueva List<Gasolinera> con las gasolineras que cumplen los criterios.
    - **Algoritmo Overview:**
      1. Crea un stream a partir de la lista de gasolineras.
      2. Si provincia es válida y no es "todas", filtra por gasolineras cuya provincia coincida (ignorando mayúsculas/minúsculas y espacios).
      3. Si tipo es válido y no es "todos", normaliza tipo a minúsculas y filtra por gasolineras que tengan un precio positivo y no NaN para ese tipo de combustible.
      4. Recolecta los resultados en una nueva lista.
  + + obtenerEstadisticas(gasolinerasFiltradas: List<Gasolinera>, tipoCombustible: String): String  
    - **Propósito:** Calcula y devuelve estadísticas (cantidad, media, mínimo, máximo) de los precios para un tipoCombustible específico, basándose en una lista de gasolinerasFiltradas previamente.
    - **Parámetros:**
      1. gasolinerasFiltradas: La lista de gasolineras sobre la cual calcular las estadísticas.
      2. tipoCombustible: El tipo de combustible para el cual se calculan las estadísticas (no nulo, no vacío).
    - **Retorno:** Una String formateada con las estadísticas. Muestra "N/A" para valores no calculables (e.g., media si no hay datos, o min/max si son infinitos).
    - **Pre-condiciones:** tipoCombustible no debe ser nulo ni vacío. gasolinerasFiltradas no debe ser nula.
    - **Excepciones Thrown:** IllegalArgumentException si tipoCombustible es nulo/vacío o gasolinerasFiltradas es nula.
    - **Algoritmo Overview:**
      1. Valida los parámetros.
      2. Normaliza tipoCombustible a minúsculas.
      3. Usa DoubleSummaryStatistics sobre los precios del tipoCombustible de la lista gasolinerasFiltradas, excluyendo precios no válidos (NaN o &lt;= 0).
      4. Formatea los resultados (contador, media, mínimo, máximo) en una cadena.
  + + obtenerGasolineraMasBarata(tipo: String, provincia: String): Gasolinera  
    - **Propósito:** Encuentra y devuelve la gasolinera con el precio más bajo para un tipo de combustible y provincia específicos.
    - **Parámetros:**
      1. tipo: El tipo de combustible (no nulo, no vacío).
      2. provincia: La provincia (no nula, no vacía).
    - **Retorno:** El objeto Gasolinera más barato que cumple los criterios. Devuelve null si no se encuentra ninguna, la lista de gasolineras está vacía, o no hay gasolineras que cumplan el filtro con precio válido.
    - **Pre-condiciones:** tipo y provincia no deben ser nulos ni vacíos.
    - **Excepciones Thrown:** IllegalArgumentException si tipo o provincia son nulos o vacíos.
    - **Algoritmo Overview:**
      1. Valida los parámetros.
      2. Si la lista interna de gasolineras está vacía, retorna null.
      3. Normaliza tipo y provincia.
      4. Reutiliza filtrarGasolineras() con los parámetros normalizados para obtener una lista de candidatas.
      5. De la lista filtrada, encuentra la gasolinera con el precio mínimo para el tipo de combustible especificado (asegurándose de que el precio sea válido y > 0).

#### **Paquete presentacion**

##### **Clase VentanaPrincipal**

* **Propósito de la Clase:** Representa la interfaz gráfica de usuario (GUI) principal de la aplicación. Permite al usuario seleccionar una provincia y un tipo de combustible para filtrar gasolineras, ver estadísticas y encontrar la más barata. Extiende JFrame.
* **Atributos Principales (Swing y Lógica):**
  + - comboProvincias: JComboBox<String>: desplegable para seleccionar la provincia.
  + - comboTipos: JComboBox<String>: desplegable para seleccionar el tipo de combustible.
  + - btnFiltrar: JButton: botón para ejecutar la acción de filtrado.
  + - btnEstadisticas: JButton: botón para mostrar estadísticas.
  + - btnMasBarata: JButton: botón para encontrar la gasolinera más barata.
  + - txtEstadisticas: JTextArea: área de texto para mostrar estadísticas o información de la más barata.
  + - tablaResultados: JTable: tabla para mostrar los resultados del filtrado o la gasolinera más barata.
  + - modeloTabla: DefaultTableModel: modelo de datos para tablaResultados.
  + - gestor: GestorGasolineras
* **Constructores:**
  + + VentanaPrincipal(gestor: GestorGasolineras)
    - **Propósito:** Inicializa la ventana principal, configura sus componentes y establece los manejadores de eventos.
    - **Parámetros:**
      * gestor: La instancia de GestorGasolineras que proporcionará los datos y la lógica de negocio.
    - **Post-condiciones:** La ventana está configurada y lista para ser mostrada. Los desplegables de provincias y tipos de combustible se cargan con datos del gestor. Si el gestor es nulo, se muestra un mensaje de error y se usa un gestor "dummy" para evitar NullPointerExceptions, aunque la funcionalidad será limitada.
* **Métodos Principales (Privados, mayormente manejadores de eventos o helpers de UI):**
  + - inicializarComponentes(): Configura y distribuye los componentes Swing en la ventana (paneles, botones, tabla, etc.).
  + - cargarProvincias(): Llena comboProvincias con datos de gestor.obtenerProvincias(). Añade una opción "Todas".
  + - cargarTiposCombustible(): Llena comboTipos con datos de gestor.obtenerTiposCombustibleDisponibles(). Añade una opción "Todos".
  + - accionFiltrar(e: ActionEvent): Manejador del evento del botón "Filtrar". Obtiene la provincia y tipo de combustible seleccionados, valida las selecciones, llama a gestor.filtrarGasolineras() y actualiza la tabla con mostrarResultados(). Limpia el área de estadísticas.
  + - accionEstadisticas(e: ActionEvent): Manejador del evento del botón "Estadísticas". Obtiene la provincia y tipo de combustible seleccionados, valida las selecciones y llama a mostrarEstadisticas(). Limpia la tabla de resultados.
  + - accionMasBarata(e: ActionEvent): Manejador del evento del botón "Más Barata". Obtiene la provincia y tipo de combustible seleccionados, valida las selecciones y llama a mostrarGasolineraMasBarata().
  + - validarSeleccionProvincia(): String: Obtiene y valida la selección del comboProvincias. Devuelve null y muestra error si no es válida.
  + - validarSeleccionTipoCombustible(): String: Obtiene y valida la selección del comboTipos. Devuelve null y muestra error si no es válida.
  + - mostrarResultados(resultados: List<Gasolinera>, tipo: String): Actualiza modeloTabla con la lista de resultados. Muestra provincia, municipio, localidad, dirección y el precio del tipo de combustible seleccionado.
  + - mostrarEstadisticas(provincia: String, tipo: String): Obtiene las estadísticas del gestor para la provincia y tipo dados y las muestra en txtEstadisticas.
  + - mostrarGasolineraMasBarata(provincia: String, tipo: String): Obtiene la gasolinera más barata del gestor para la provincia y tipo dados. La muestra en la tabla (limpiándose previamente) y en txtEstadisticas.

##### **Clase MainApp**

* **Propósito de la Clase:** Contiene el método main que sirve como punto de entrada para la aplicación. Se encarga de configurar el look and feel, permitir al usuario seleccionar el archivo CSV de datos, inicializar las capas de lógica y presentación, y mostrar la ventana principal. También maneja errores globales durante el inicio.
* **Métodos Públicos Estáticos:**
  + + main(args: String[])
    - **Propósito:** Inicia la aplicación.
    - **Parámetros:**
      1. args: Argumentos de línea de comandos (no utilizados actualmente).
    - **Pre-condiciones:** Ninguna específica, aparte de un entorno Java funcional.
    - **Post-condiciones:** La aplicación se inicia, o se muestra un mensaje de error y la aplicación se cierra si ocurren problemas críticos durante la inicialización.
    - **Algoritmo Overview:**
      1. Intenta establecer el "Look and Feel" del sistema para la GUI.
      2. Muestra un JFileChooser para que el usuario seleccione el archivo CSV de gasolineras.
      3. Si el usuario no selecciona un archivo, muestra un mensaje y termina.
      4. Si se selecciona un archivo: a. Obtiene la ruta absoluta del archivo. b. Llama a LectorCSV.leerGasolineras() para cargar los datos. c. Crea una instancia de GestorGasolineras con los datos leídos. d. Crea y muestra una instancia de VentanaPrincipal, pasándole el gestor.
      5. Captura y maneja excepciones (CSVLecturaException, IllegalArgumentException, y Exception genérica) mostrando diálogos de error al usuario y terminando la aplicación o registrando el error.

### **2. Modelo de Dominio UML**

El modelo de dominio se centra en los conceptos clave del problema desde la perspectiva del negocio o del usuario, antes de considerar detalles de implementación de software.

* **Entidades Conceptuales Principales:**
  + **Gasolinera:**
    - Representa una estación de servicio.
    - *Atributos Conceptuales:* Ubicación (Provincia, Municipio, Localidad, Dirección, Código Postal), Precios de Combustibles (Precio Gasolina 95, Precio Gasolina 98, Precio Diésel, Precio Diésel Plus).
  + **Combustible:** (Implícito en los precios de Gasolinera)
    - Representa un tipo de combustible.
    - *Atributos Conceptuales:* Tipo (e.g., Gasolina 95, Diésel).
* **Relaciones Conceptuales:**
  + Un **Sistema de Consulta de Gasolineras** (representado por la aplicación en su conjunto) *gestiona* múltiples **Gasolineras**.
  + Una **Gasolinera** *ofrece* precios para diferentes tipos de **Combustible**.

**Diagrama de Dominio (Descripción Textual):**

[Gasolinera]

- Provincia

- Municipio

- Localidad

- Dirección

- Código Postal

- PrecioGasolina95

- PrecioGasolina98

- PrecioDiésel

- PrecioDiéselPlus

(Nota: 'Combustible' como entidad separada con atributos 'Tipo' y 'Precio' podría ser una alternativa si los tipos de combustible fueran más dinámicos o tuvieran más propiedades, pero dado el diseño actual donde los precios son atributos directos de Gasolinera, se simplifica así. El sistema en general maneja una colección de estas 'Gasolineras'.)

El foco está en la Gasolinera como el concepto central del dominio.

### **3. Diseño UML de Clases y Métodos (Diagrama de Clases)**

Este diagrama representa la estructura estática de las clases de software y sus interrelaciones.

**Clases y sus Miembros Principales (Visibilidad: + público, - privado, # protegido, ~ paquete):**

1. **modelo.CSVLecturaException**
   * *Extiende:* java.lang.RuntimeException
   * *Constructores:*
     + + CSVLecturaException(message: String)
     + + CSVLecturaException(message: String, cause: Throwable)
2. **modelo.Gasolinera**
   * *Atributos:*
     + - provincia: String
     + - municipio: String
     + - localidad: String
     + - cp: String
     + - direccion: String
     + - gasolina95: double
     + - gasolina98: double
     + - diesel: double
     + - dieselPlus: double
   * *Constructores:*
     + + Gasolinera(provincia: String, municipio: String, localidad: String, cp: String, direccion: String, gasolina95: double, gasolina98: double, diesel: double, dieselPlus: double)
   * *Métodos:*
     + + getProvincia(): String
     + + getMunicipio(): String
     + + getLocalidad(): String
     + + getCp(): String
     + + getDireccion(): String
     + + getGasolina95(): double
     + + getGasolina98(): double
     + + getDiesel(): double
     + + getDieselPlus(): double
     + + getPrecioCombustible(tipo: String): double
     + + toString(): String
3. **modelo.LectorCSV**
   * *Métodos Estáticos:*
     + + leerGasolineras(rutaArchivo: String): List<Gasolinera>
     + - parsePrecio(campo: String, indice: int): double
4. **logica.GestorGasolineras**
   * *Atributos:*
     + - gasolineras: List<Gasolinera>
     + - df: static final DecimalFormat
   * *Constructores:*
     + + GestorGasolineras(gasolineras: List<Gasolinera>)
   * *Métodos:*
     + + obtenerProvincias(): List<String>
     + + obtenerTiposCombustibleDisponibles(): List<String>
     + + filtrarGasolineras(tipo: String, provincia: String): List<Gasolinera>
     + + obtenerEstadisticas(gasolinerasFiltradas: List<Gasolinera>, tipoCombustible: String): String
     + + obtenerGasolineraMasBarata(tipo: String, provincia: String): Gasolinera
5. **presentacion.VentanaPrincipal**
   * *Extiende:* javax.swing.JFrame
   * *Atributos (seleccionados):*
     + - comboProvincias: JComboBox<String>
     + - comboTipos: JComboBox<String>
     + - btnFiltrar: JButton
     + - btnEstadisticas: JButton
     + - btnMasBarata: JButton
     + - txtEstadisticas: JTextArea
     + - tablaResultados: JTable
     + - modeloTabla: DefaultTableModel
     + - gestor: GestorGasolineras
   * *Constructores:*
     + + VentanaPrincipal(gestor: GestorGasolineras)
   * *Métodos Principales (simplificado, algunos son privados):*
     + - inicializarComponentes()
     + - cargarProvincias()
     + - cargarTiposCombustible()
     + - accionFiltrar(e: ActionEvent)
     + - accionEstadisticas(e: ActionEvent)
     + - accionMasBarata(e: ActionEvent)
     + - mostrarResultados(resultados: List<Gasolinera>, tipo: String)
     + - mostrarEstadisticas(provincia: String, tipo: String)
     + - mostrarGasolineraMasBarata(provincia: String, tipo: String)
6. **presentacion.MainApp**
   * *Métodos Estáticos:*
     + + main(args: String[])

**Interrelaciones Principales (Descripción Textual del Diagrama de Clases):**

* **Herencia:**
  + CSVLecturaException --|> RuntimeException (Generalización: CSVLecturaException es un tipo de RuntimeException).
  + VentanaPrincipal --|> JFrame (Generalización: VentanaPrincipal es un tipo de JFrame).
* **Asociación:**
  + GestorGasolineras "1" o "0..1" --- manages ---> "\*" Gasolinera (Composición: GestorGasolineras tiene una lista de Gasolinera. La vida de la lista de gasolineras que maneja el gestor depende del gestor).
    - Usaremos una línea sólida con un rombo relleno en el lado de GestorGasolineras para denotar composición: GestorGasolineras o&lt;>- List<Gasolinera>
  + VentanaPrincipal "1" --- uses ---> "1" GestorGasolineras (Asociación: VentanaPrincipal tiene una referencia a GestorGasolineras para interactuar con la lógica de negocio).
    - VentanaPrincipal --> GestorGasolineras
* **Dependencia (línea discontinua con flecha abierta):**
  + LectorCSV ..produces..> Gasolinera (El método leerGasolineras crea instancias de Gasolinera).
  + LectorCSV ..throws..> CSVLecturaException
  + Gasolinera ..throws..> IllegalArgumentException (Constructor)
  + GestorGasolineras ..uses..> Gasolinera (En sus métodos, opera sobre objetos Gasolinera).
  + GestorGasolineras ..throws..> IllegalArgumentException
  + VentanaPrincipal ..uses..> Gasolinera (Para mostrar datos en la tabla).
  + MainApp ..uses..> LectorCSV (Llama a leerGasolineras).
  + MainApp ..creates/uses..> GestorGasolineras.
  + MainApp ..creates/uses..> VentanaPrincipal.
  + MainApp ..handles..> CSVLecturaException, IllegalArgumentException.

**Representación Textual Simplificada de Relaciones Clave (formato Mermaid para fácil visualización si se pega en un editor compatible):**

Fragmento de código

classDiagram

class CSVLecturaException {

+CSVLecturaException(String)

+CSVLecturaException(String, Throwable)

}

RuntimeException <|-- CSVLecturaException

class Gasolinera {

-String provincia

-String municipio

-String localidad

-String cp

-String direccion

-double gasolina95

-double gasolina98

-double diesel

-double dieselPlus

+Gasolinera(...)

+getProvincia(): String

+getMunicipio(): String

+getLocalidad(): String

+getCp(): String

+getDireccion(): String

+getGasolina95(): double

+getGasolina98(): double

+getDiesel(): double

+getDieselPlus(): double

+getPrecioCombustible(String): double

+toString(): String

}

class LectorCSV {

<<Utility>>

+leerGasolineras(String): List~Gasolinera~

-parsePrecio(String, int): double

}

LectorCSV ..> Gasolinera : creates

LectorCSV ..> CSVLecturaException : throws

class GestorGasolineras {

-List~Gasolinera~ gasolineras

+GestorGasolineras(List~Gasolinera~)

+obtenerProvincias(): List~String~

+obtenerTiposCombustibleDisponibles(): List~String~

+filtrarGasolineras(String, String): List~Gasolinera~

+obtenerEstadisticas(List~Gasolinera~, String): String

+obtenerGasolineraMasBarata(String, String): Gasolinera

}

GestorGasolineras "1" \*-- "\*" Gasolinera : manages

class VentanaPrincipal {

-GestorGasolineras gestor

+VentanaPrincipal(GestorGasolineras)

-accionFiltrar()

-accionEstadisticas()

-accionMasBarata()

-mostrarResultados(List~Gasolinera~, String)

}

JFrame <|-- VentanaPrincipal

VentanaPrincipal --> GestorGasolineras : uses

VentanaPrincipal ..> Gasolinera : displays

class MainApp {

<<Application>>

+main(String[])

}

MainApp ..> LectorCSV : uses

MainApp ..> GestorGasolineras : creates/uses

MainApp ..> VentanaPrincipal : creates/shows

MainApp ..> CSVLecturaException : handles

MainApp ..> IllegalArgumentException : handles

**Notas sobre el diagrama Mermaid anterior:**

* <<Utility>> y <<Application>> son estereotipos para indicar el rol de la clase.
* List~Gasolinera~ indica una lista de objetos Gasolinera.
* \*-- indica composición.
* --> indica asociación dirigida.
* ..> indica dependencia.
* <|-- indica generalización (herencia).
* Se han simplificado los métodos en el diagrama Mermaid por brevedad, pero la documentación completa de los métodos está arriba.
* La relación entre MainApp y las otras clases es principalmente de creación y orquestación dentro del método main.

**Documentación del Sistema de Gestión de Estaciones de Servicio1. Descripción Detallada de la Funcionalidad de Clases y Métodos**

A continuación, se presenta una descripción pormenorizada de cada clase proporcionada:**Paquete** **modeloClase** **Gasolinera**

* **Propósito de la Clase:** Representa una entidad de estación de servicio individual, almacenando su información de ubicación y los precios de diversos tipos de combustible. Esta clase se considera inmutable una vez creada (los precios y la ubicación no se modifican después de la instanciación).
* **Atributos:**
  + provincia: String: Provincia donde se encuentra ubicada la estación de servicio.
  + municipio: String: Municipio donde se encuentra ubicada la estación de servicio.
  + localidad: String: Localidad donde se encuentra ubicada la estación de servicio.
  + cp: String: Código postal de la estación de servicio.
  + direccion: String: Dirección completa de la estación de servicio.
  + gasolina95: double: Precio de la Gasolina 95.
  + gasolina98: double: Precio de la Gasolina 98.
  + diesel: double: Precio del Diésel.
  + dieselPlus: double: Precio del Diésel Plus.
* **Constructores:**
  + - Gasolinera(provincia: String, municipio: String, localidad: String, cp: String, direccion: String, gasolina95: double, gasolina98: double, diesel: double, dieselPlus: double)<!-- end list -->
    - **Propósito:** Construye una nueva instancia de Gasolinera.
    - **Parámetros:**
      * provincia: Nombre de la provincia (no nulo, no vacío).
      * municipio: Nombre del municipio (no nulo, no vacío).
      * localidad: Nombre de la localidad (no nulo, no vacío).
      * cp: Código postal (puede ser nulo o vacío si no es esencial).
      * direccion: Dirección de la estación de servicio (no nula, no vacía).
      * gasolina95: Precio para Gasolina 95 (se espera double, Double.NaN si no disponible).
      * gasolina98: Precio para Gasolina 98 (se espera double, Double.NaN si no disponible).
      * diesel: Precio para Diésel (se espera double, Double.NaN si no disponible).
      * dieselPlus: Precio para Diésel Plus (se espera double, Double.NaN si no disponible).
    - **Pre-condiciones:** Los parámetros provincia, municipio, localidad y direccion no deben ser nulos ni cadenas vacías (después de trim()). Los precios deben ser valores numéricos válidos o Double.NaN.
    - **Post-condiciones:** Se crea un nuevo objeto Gasolinera con los datos proporcionados.
    - **Excepciones Lanzadas:** IllegalArgumentException si alguno de los parámetros de cadena obligatorios es nulo o vacío.
* **Métodos Públicos:**
  + - getProvincia(): String
    - getMunicipio(): String
    - getLocalidad(): String
    - getCp(): String
    - getDireccion(): String
    - getGasolina95(): double
    - getGasolina98(): double
    - getDiesel(): double
    - getDieselPlus(): double<!-- end list -->
    - **Propósito:** Métodos de acceso para obtener los valores de los atributos correspondientes.
    - **Retorno:** El valor del atributo respectivo.
    - getPrecioCombustible(tipo: String): double<!-- end list -->
    - **Propósito:** Obtiene el precio de un tipo de combustible específico.
    - **Parámetros:**
      * tipo: El tipo de combustible (e.g., "gasolina95", "diesel"). Se normaliza a minúsculas internamente.
    - **Retorno:** El precio (double) del combustible especificado. Devuelve Double.NaN si el tipo es nulo, vacío, o no corresponde a ninguno de los tipos conocidos ("gasolina95", "gasolina98", "diesel", "dieselplus").
    - **Pre-condiciones:** Ninguna explícita más allá de la validez del objeto.
    - **Post-condiciones:** El precio del combustible es devuelto sin modificar el estado del objeto.
    - toString(): String<!-- end list -->
    - **Propósito:** Devuelve una representación en cadena del objeto Gasolinera, útil para depuración o visualización simple.
    - **Retorno:** Una cadena que incluye la dirección, localidad, provincia y los precios de todos los combustibles. Los precios no disponibles (Double.NaN) se muestran como "N/A".

**Clase** **LectorCSV**

* **Propósito de la Clase:** Proporciona funcionalidad para leer datos de estaciones de servicio desde un archivo CSV y convertirlos en una lista de objetos Gasolinera. Es una clase de utilidad con métodos estáticos.
* **Métodos Públicos Estáticos:**
  + - leerGasolineras(rutaArchivo: String): List<Gasolinera><!-- end list -->
    - **Propósito:** Lee un archivo CSV que contiene datos de estaciones de servicio y los transforma en una lista de objetos Gasolinera.
    - **Parámetros:**
      1. rutaArchivo: La ruta completa al archivo CSV.
    - **Retorno:** Una List<Gasolinera> poblada con los datos del archivo. Devuelve una lista vacía si el archivo no contiene datos válidos o está vacío (después de la cabecera).
    - **Pre-condiciones:**
      1. rutaArchivo no debe ser nulo ni vacío.
      2. El archivo especificado por rutaArchivo debe existir y ser legible.
      3. El archivo CSV debe tener una línea de cabecera que se omite.
      4. Las líneas de datos deben seguir un formato esperado con un número mínimo de campos (al menos 17 para acceder a los índices de precios esperados).
      5. Los campos de precios deben ser parseables a double según la lógica de parsePrecio.
    - **Post-condiciones:** Se devuelve una lista de objetos Gasolinera. Las líneas con datos incorrectos o insuficientes se omiten y se registra una advertencia en System.err.
    - **Excepciones Lanzadas:**
      1. CSVLecturaException:
         * Si rutaArchivo es nulo o vacío.
         * Si ocurre FileNotFoundException al intentar abrir el archivo.
         * Si ocurre IOException durante la lectura del archivo.
         * Si una línea no tiene el número mínimo de campos esperados.
         * Si faltan datos esenciales para construir un objeto Gasolinera (e.g., provincia vacía).
    - **Algoritmo Overview:**
      1. Valida la ruta del archivo.
      2. Abre el archivo usando BufferedReader y FileReader.
      3. Lee y descarta la primera línea (cabecera).
      4. Itera sobre las líneas restantes: a. Divide la línea en campos usando una expresión regular que maneja comas dentro de campos entrecomillados. b. Verifica que haya suficientes campos. Si no, lanza CSVLecturaException. c. Extrae los campos correspondientes a provincia, municipio, localidad, CP, dirección y los diferentes precios de combustible. d. Convierte los campos de precio usando el método parsePrecio. e. Intenta crear un nuevo objeto Gasolinera. f. Si la creación es exitosa, lo añade a la lista. g. Si ocurre NumberFormatException o IllegalArgumentException al procesar una línea, imprime un mensaje de error y continúa con la siguiente línea (la línea problemática se omite).
      5. Devuelve la lista de estaciones de servicio.
* **Métodos Privados Estáticos:**
  + - parsePrecio(campo: String, indice: int): double
    - **Propósito:** Convierte una representación de precio en formato String (proveniente del CSV) a un valor double. Maneja distintos formatos (con coma o punto decimal, o enteros representando milésimas).
    - **Parámetros:**
      1. campo: El valor del campo de precio como String.
      2. indice: El índice de la columna (usado para mensajes de error, aunque no en la implementación actual del println).
    - **Retorno:** El precio como double. Devuelve Double.NaN si el campo es nulo, vacío, "null", o si no se puede convertir a número.
    - **Algoritmo Overview:**
      1. Si campo es nulo, devuelve Double.NaN.
      2. Limpia el campo: quita espacios, comillas.
      3. Si el campo procesado está vacío o es "null", devuelve Double.NaN.
      4. Reemplaza comas (,) por puntos (.) para normalizar el separador decimal.
      5. Intenta convertir a double: a. Si contiene un punto