FARMEASY

Revisión acoplamiento, cohesión y generación de documentación

Integrantes

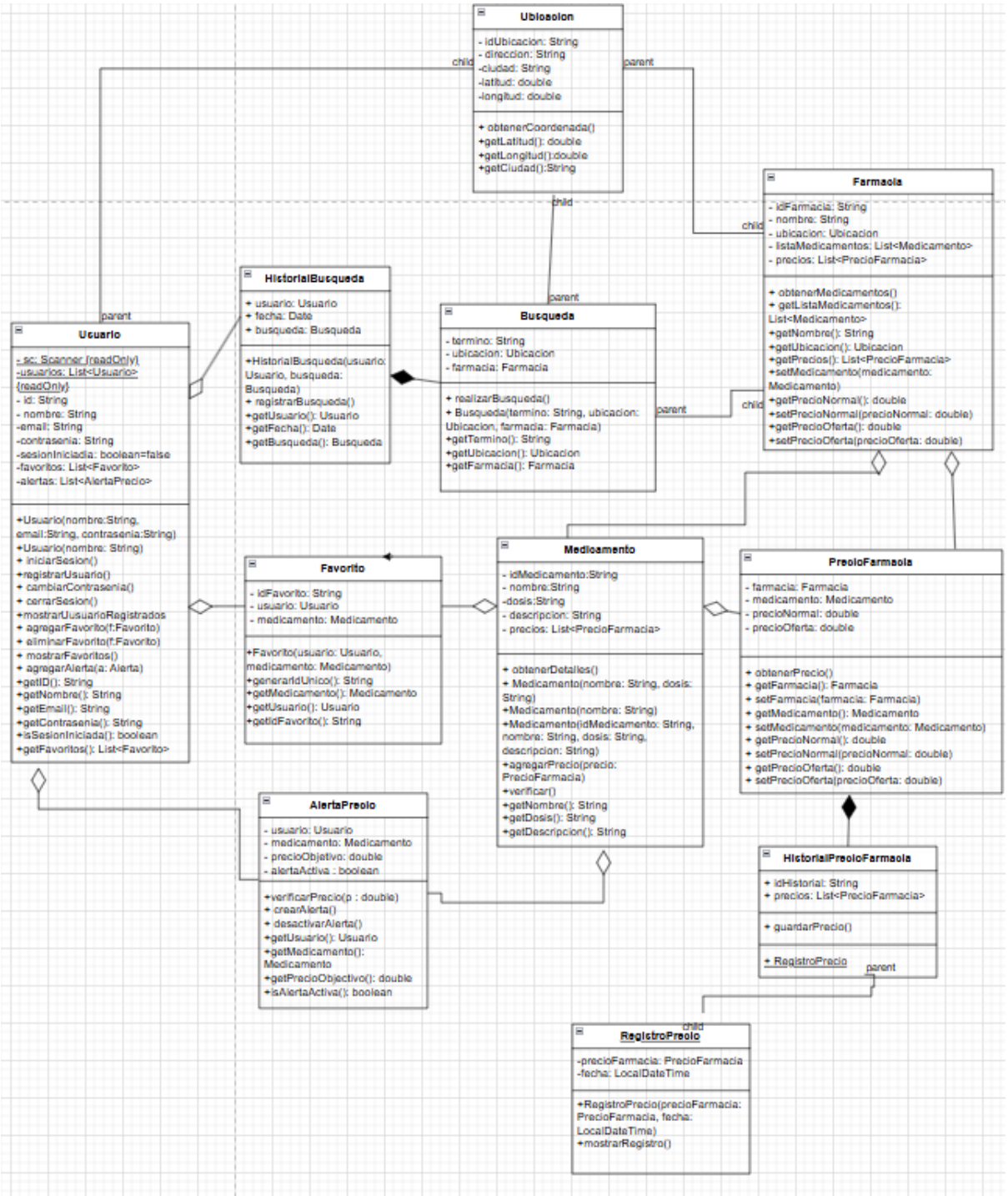
José Burgos
Cristopher Gallegos
Enrique Pincheira

Universidad de La frontera
05-05-2025

Introducción

En este documento se describe el programa desarrollado llamado “FarmEasy”, una plataforma dedicada a facilitar la búsqueda de medicamentos y su comparación de precios.
A continuación, se presenta la documentación generada con Javadoc, el diagrama de clases que representa el funcionamiento del sistema y un análisis sobre el grado de acoplamiento y cohesión del mismo, junto con propuestas de mejora.

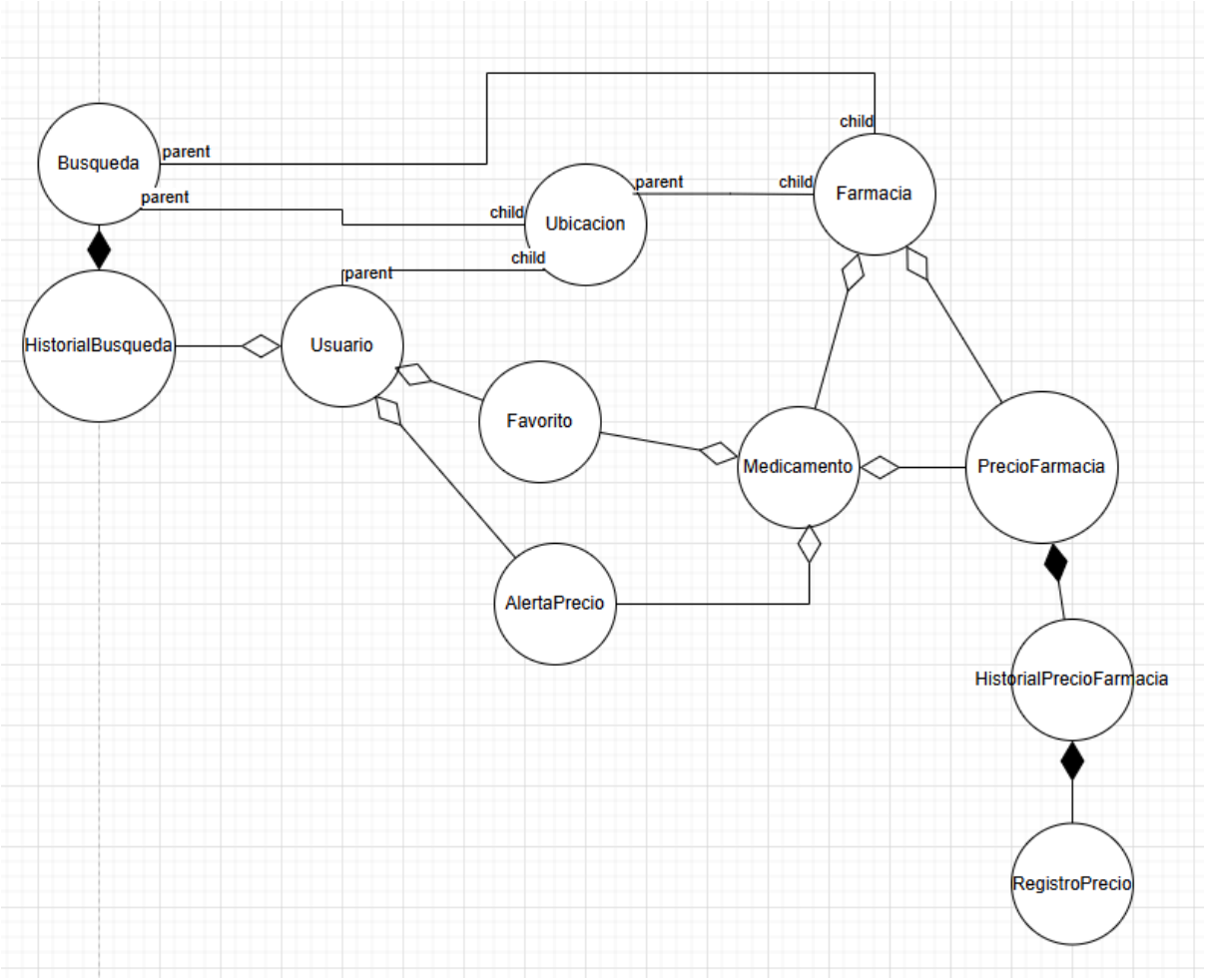
Diagrama de clases



En este diagrama se representa la estructura de clases del sistema FarmEasy. Se incluyen las relaciones entre clases como Usuario, Medicamento, Favorito, AlertaPrecio, HistorialBusqueda, Farmacia, Búsqueda, entre otros.

Análisis de acoplamiento y cohesión

Diseño simplificado



El sistema está compuesto por clases como Usuario, Medicamento, Favorito, AlertaPrecio, HistorialBusqueda y Búsqueda, las cuales modelan las funcionalidades clave del sistema.

Análisis del acoplamiento:

- El sistema tiene un acoplamiento bajo en general, ya que las clases están relacionadas de forma coherente. La clase Usuario tiene un acoplamiento moderado, ya que se conecta con varias otras clases. Esto es comprensible por su rol central, pero podría mejorarse.

Análisis de cohesión:

- La mayoría de las clases tienen una cohesión alta, ya que cada una se enfoca en una tarea específica. Sin embargo, “Usuario” agrupa muchas responsabilidades (gestionar favoritos, alertas, búsquedas), lo que disminuye su cohesión.

¿Se cumple con bajo acoplamiento y alta cohesión?

En general, sí se cumple. Las clases del sistema están bien separadas y cada una tiene una función clara. Por ejemplo, hay clases específicas para los medicamentos, para las búsquedas, para los favoritos, etc.

Sin embargo, hay una pequeña excepción en la clase “Usuario”, ya que está encargada de hacer muchas cosas al mismo tiempo: manejar los favoritos, el historial de búsquedas, las alertas, etc. Esto puede hacer que el código sea más difícil de mantener a futuro.

Propuesta de mejora

Para que el sistema sea más fácil de mantener y de ampliar en el futuro, una buena idea sería separar las responsabilidades de la clase “Usuario” en varias clases más pequeñas. Por ejemplo:

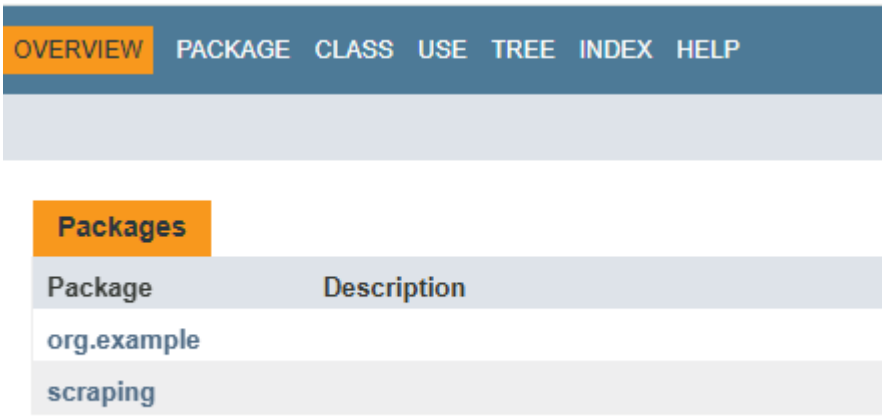
- Mover métodos de favorito a la clase Favoritos
- Mover métodos de alerta a la clase AlertaPrecio
- Mover métodos del historial de búsqueda.

De esta forma, cada clase tendría una única tarea específica, lo que ayuda a que el código sea más ordenado y fácil de entender.

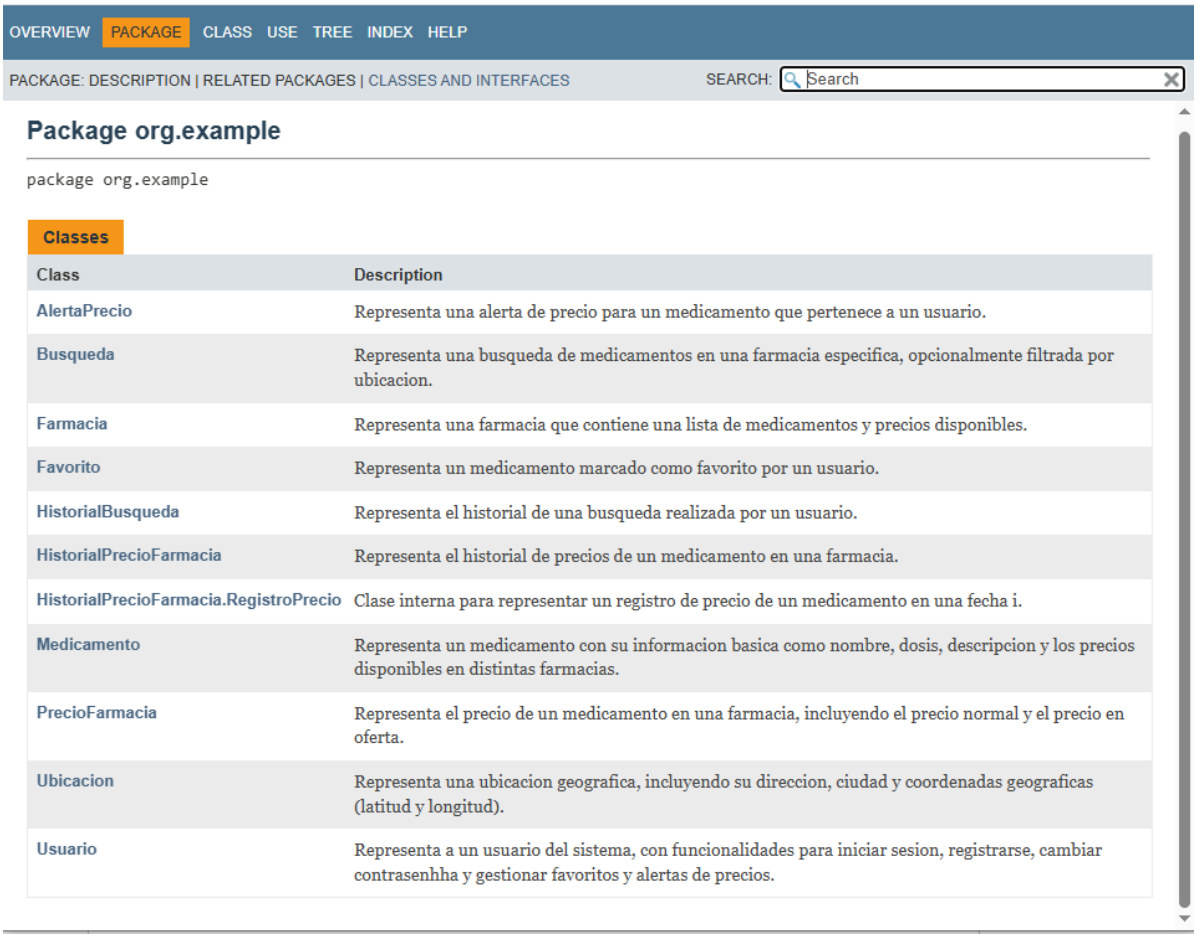
Javadoc

A continuación, se presenta un resumen con algunas clases de la documentación del código fuente generado mediante Javadoc, el cual describe las clases, atributos y métodos del sistema. (el resto de clases está en el index.html subido al repositorio)

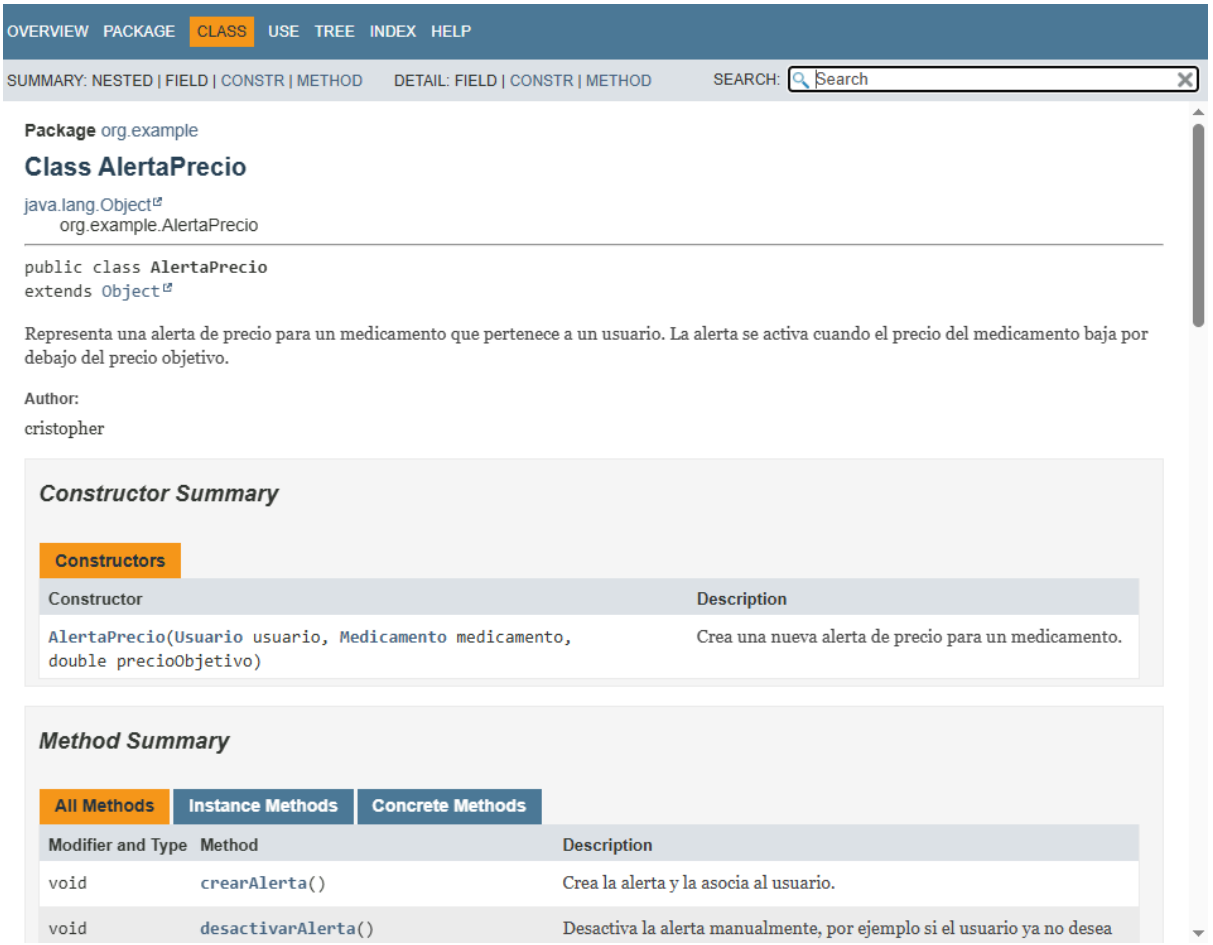
Packages del proyecto



Clases del proyecto dentro del package



Clase AlertaPrecio con constructor y algunos métodos



OVERVIEW PACKAGE CLASS USE TREE INDEX HELP

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD SEARCH: Search

Package org.example

Class AlertaPrecio

java.lang.Object
org.example.AlertaPrecio

```
public class AlertaPrecio
extends Object
```

Representa una alerta de precio para un medicamento que pertenece a un usuario. La alerta se activa cuando el precio del medicamento baja por debajo del precio objetivo.

Author:
cristopher

Constructor Summary

Constructors

Constructor	Description
`AlertaPrecio(Usuario usuario, Medicamento medicamento, double precioObjetivo)`	Crea una nueva alerta de precio para un medicamento.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
void	`crearAlerta()`	Crea la alerta y la asocia al usuario.
void	`desactivarAlerta()`	Desactiva la alerta manualmente, por ejemplo si el usuario ya no desea

Clase Búsqueda con métodos

Package org.example

Class Busqueda

java.lang.Object
org.example.Busqueda

```
public class Busqueda
extends Object
```

Representa una busqueda de medicamentos en una farmacia especifica, opcionalmente filtrada por ubicacion.

Author:
Jose

Constructor Summary

Constructors

Constructor	Description
`Busqueda(String termino, Ubicacion ubicacion, Farmacia farmacia)`	Crea una nueva busqueda con los parametros dados.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
Farmacia	`getFarmacia()`	Obtiene la farmacia en la que se realiza la busqueda.
String	`getTermino()`	Obtiene el termino de busqueda.
Ubicacion	`getUbicacion()`	Obtiene la ubicacion de la bÃºsqueda.
void	`realizarBusqueda()`	Realiza una busqueda del termino especificado dentro de la farmacia y muestra resultados.

Methods inherited from class java.lang.Object

Clase Medicamento con constructores

Package org.example

Class Medicamento

java.lang.Object
    org.example.Medicamento

```
public class Medicamento
extends Object
```

Representa un medicamento con su informacion basica como nombre, dosis, descripcion y los precios disponibles en distintas farmacias.

Author:

cristopher

Constructor Summary

Constructors

Constructor	Description
`Medicamento(String nombre)`	Constructor que inicializa el medicamento solo con el nombre.
`Medicamento(String nombre, String dosis)`	Constructor simple que inicializa el medicamento con nombre y dosis.
`Medicamento(String idMedicamento, String nombre, String dosis, String descripcion)`	Constructor completo que inicializa todos los campos del medicamento.

Métodos clase medicamento

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
void	`agregarPrecio(PrecioFarmacia precio)`	Agrega un precio a la lista de precios del medicamento.
String	`getDescripcion()`	Devuelve la descripcion del medicamento.
String	`getDosis()`	Devuelve la dosis del medicamento.
String	`getNombre()`	Devuelve el nombre del medicamento.
void	`obtenerDetalles()`	Muestra los detalles del medicamento, incluyendo nombre, dosis, descripcion y precios en farmacias si estan disponibles.
void	`verificar()`	MÃ©todo para realizar verificaciones relacionadas con el medicamento, como disponibilidad o cambios en los precios.

Methods inherited from class java.lang.Object

`equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

Clase Ubicación

Package org.example

Class Ubicacion

java.lang.Object
    org.example.Ubicacion

```
public class Ubicacion
extends Object
```

Representa una ubicacion geografica, incluyendo su direccion, ciudad y coordenadas geograficas (latitud y longitud).

Author:

Enrique

Constructor Summary

Constructors

Constructor	Description
`Ubicacion()`	

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
`String`	`getCiudad()`	Devuelve el nombre de la ciudad asociada a la ubicaciÃ³n.
`double`	`getLatitud()`	Devuelve la latitud de la ubicaciÃ³n.
`double`	`getLongitud()`	Devuelve la longitud de la ubicaciÃ³n.
`void`	`obtenerCordenada()`	Obtiene las coordenadas geogrÃificas (latitud y longitud) de la ciudad especificada.

Methods inherited from class java.lang.Object

Clase Usuario y constructores

Package org.example

Class Usuario

java.lang.Object
    org.example.Usuario

```
public class Usuario
extends Object
```

Representa a un usuario del sistema, con funcionalidades para iniciar sesion, registrarse, cambiar contrasenhha y gestionar favoritos y alertas de precios.

Author:

Jose

Constructor Summary

Constructors

Constructor	Description
`Usuario(String nombre)`	Constructor para crear un nuevo usuario solo con nombre.
`Usuario(String nombre, String email, String contrasenia)`	Constructor para crear un nuevo usuario con nombre, correo electronico y contrasenia.

Metodos clase usuario

Method Summary

All Methods | **Static Methods** | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
void	`agregarAFavoritos(Favorito favorito)`	Agrega un medicamento a los favoritos del usuario.
void	`agregarAlerta(AlertaPrecio alerta)`	Agrega una alerta de precio a la lista del usuario.
void	`cambiarContrasenia()`	Permite al usuario cambiar su contrasenia.
void	`cerrarSesion()`	Cierra la sesion del usuario actual.
void	`eliminarDeFavoritos(Favorito favorito)`	Elimina un medicamento de los favoritos del usuario.
String	`getContrasenia()`	Devuelve la contrasenia del usuario.
String	`getEmail()`	Devuelve el correo electronico del usuario.
List<Favorito>	`getFavoritos()`	Devuelve la lista de medicamentos favoritos del usuario.
String	`getId()`	Devuelve el ID del usuario.
String	`getNombre()`	Devuelve el nombre del usuario.
void	`iniciarSesion()`	Inicia sesion en el sistema validando el correo electronico y la contrasenia.
boolean	`isSesionIniciada()`	Indica si el usuario ha iniciado sesion.
void	`mostrarFavoritos()`	Muestra todos los medicamentos favoritos del usuario.
static void	`mostrarUsuariosRegistrados()`	Muestra los usuarios registrados en el sistema.
void	`registrarUsuario()`	Registra un nuevo usuario en el sistema.

Conclusión

El sistema FarmEasy fue diseñado con una estructura organizada. La mayoría de sus clases cumplen con los principios de bajo acoplamiento y alta cohesión, lo que favorece la mantenibilidad y escalabilidad del sistema. Se identificó una mejora en la clase Usuario, lo que permitirá una mejor modularización del sistema.

Repositorio

https://github.com/j1gnacio/farmeasy.git