Programación Orientada a Objetos. 2023-2024-Q2.

Proyecto: Gestor de base de datos.

Objetivos

Los objetivos de este proyecto son:

- Realización de una aplicación orientada a objetos partiendo desde prácticamente cero.
- Gestionar los datos de la aplicación mediante estructuras de objetos de una cierta complejidad.
- Procesar entradas de datos mediante el teclado.
- Utilizar herencia y polimorfismo en un programa orientado a objetos.
- Gestionar excepciones.
- Utilizar las clases de Java para operaciones de Entrada/Salida.

Introducción. Conceptos básicos de bases de datos relacionales

El objetivo del ejercicio evaluable de este cuatrimestre es construir las piezas básicas de un gestor de bases de datos relacional rudimentario.

Una base de datos relacional está formada por una serie de tablas.

Cada fila de una de las tablas representa información de una entidad (podéis asimilarla a un objeto).

Cada columna de una de las tablas contiene el valor de un atributo de esa entidad.

Toda tabla debe contener lo que se conoce como **clave primaria** (clave única en el javadoc que os pasamos). La clave primaria es el atributo (o conjunto de atributos) que constituye(n) un identificador único para cada entidad (objeto). Si, por ejemplo, cada fila contiene la información de un coche, y las columnas de esa tabla incluyen los valores de los atributos marca, matrícula y modelo, la clave primaria será la columna matricula. Si cada fila incluyese información de un alumno matriculado en la ETSETB, la clave primaria podría ser el DNI.

El programa a desarrollar presentará al usuario un interfaz de usuario que le permitirá interaccionar con dicho programa para:

1. **Crear una nueva tabla no existente (comando "crea")**. Ejemplo:

   ```
   coches crea marca modelo *matricula
   ```

   Crea una tabla llamada "`coches`" con 3 columnas de nombres "`marca`", "`modelo`" y "`matricula`" respectivamente. La clave `matricula` es la clave primaria (única) de la tabla (el carácter '*' antes de matricula así lo indica).

   El número de columnas queda determinado por el número de claves que siguen a `crea`.

2. **Añadir una nueva fila a una tabla ya existente (comando "añade")**. Ejemplo:

   ```
   coches añade marca=Seat modelo=Ibiza matricula=B1234XD
   ```

   Este comando añade a la tabla de nombre "`coches` una nueva fila con los valores "`Seat`", "`Ibiza`" y "`B1234XD`" en las columnas "`marca`", "`modelo`" y "`matricula`" respectivamente.

3. **Comando Busca**. El comando "busca" se utiliza para presentar buscar determinadas filas (o todas) y presentarlas por pantalla. A continuación, se muestran los diferentes usos:

a. **Presentar todas las filas de una tabla**. Ejemplo:
`coches busca`

Busca en la tabla "coches" todas las filas y las presenta una debajo de otra.

b. **Buscar la(s) fila(s) que cumpla(n) un cierto criterio de búsqueda (comando "busca")**. A continuación se detallan los cuatro criterios de búsqueda:

i. **Búsqueda con "="**. Ejemplo:

```
coches busca marca=Seat
```

Busca en la tabla llamada `"coches"` todas las filas que contengan en la columna `"marca"` **el String `"Seat"`** y las presenta.

ii. **Búsqueda con "#"**. Ejemplo:

```
coches busca marca#eat
```

Busca en la tabla "coches" todas las filas que contengan en la columna `"marca"` **un String que contenga el String `"eat"`** y las presenta.

iii. **Búsqueda con ">"**. Ejemplo: si se generase una tabla para gestionar multas impuestas a vehículos utilizando los comandos siguientes:

```
multas crea matricula cuantía
multas añade matricula=B1234XD cuantía=100
multas añade matricula=2345FXZ cuantía=101
multas añade matricula=B1234XD cuantía=60
```

El comando con el nuevo criterio ">"

```
multas busca cuantía>80
```

Presentaría las filas 1 y 2 correspondientes a las matrículas `B1234XD` y `2345FXZ` (cuantías de 100 y101 respectivamente).

iv. **Búsqueda con "<"**. Ejemplo: en la tabla anterior, el comando

```
multas busca cuantía<80
```

Presentaría la fila 3 correspondiente a la matrícula `B1234XD` (cuantía de 60).

4. **Eliminar aquella(s) fila(s) que cumpla(n) con un cierto criterio (comando "elimina")**. Ejemplos:

```
coches elimina marca=Seat
```

Elimina de la tabla llamada `"coches"` todas las filas que contengan en la columna `"marca"` **el String `"Seat"`**.

5. **Abandonar el programa (comando "salir")**.

6. **Ordenar una tabla en función de los valores de una determinada clave**, en orden descendente o en orden ascendente. Así por ejemplo en la tabla de multas antes mencionadas, el comando:

```
multas ordena cuantía
```

haría que el sistema nos mostrase la tabla ordenada por valores de la cuantía de las multas en orden ascendente. El comando .

```
multas ordena cuantía desc
```

haría que el sistema nos mostrase la tabla ordenada por valores de la cuantía de las multas en orden descendiente.

7. **Exportar una base de datos completa (esto es, exportar todas sus tablas) a un archivo de texto**. El comando:

   ```
   exporta exportacion.txt
   ```

   Generará un archivo de texto que contendrá todos los comandos necesarios para crear y llenar todas las tablas que la base de datos contiene en el instante en que se ejecuta dicho comando. Es decir, por cada tabla debe insertarse un comando `crea` y tantos comandos `añade` como filas tenga la tabla. Encontraréis un ejemplo detallado de lo que debe hacer este comando en el javadoc del método

   ```
   public void exporta(String archivo) ;
   ```

Detalles de los contenidos del programa.

El diagrama de clases se da en la última hoja del enunciado.

Debéis tener en cuenta que el diagrama no muestra todos los métodos de las clases, solo algunos de los más relevantes. **PARA UNA COMPLETA DESCRIPCIÓN DE LO QUE DEBÉIS IMPLEMENTAR EN CADA CLASE, DEBÉIS LEER CUIDADOSAMENTE EL JAVADOC QUE SE DISTRIBUYE CON ESTE DOCUMENTO Y AQUELLA PARTE DEL CÓDIGO QUE NO TENÉIS QUE IMPLEMENTAR**.

A continuación, se dan algunos detalles de las clases que aparecen en el diagrama:

Package upc.etsetb.poo.basededatos.iu:

- **InterfazUsuario**: Es la clase encargada de gestionar el interfaz de usuario. Como su nombre indica contiene el código que permite al usuario interaccionar con el programa.
- **Teclado**: Clase que en la que se definen dos métodos estáticos auxiliares para gestionar la entrada de líneas de texto por el teclado. El contenido de esta clase se os da hecho.
- **`Main`**: Esta clase contiene el método main(). Se os entrega completa. Se sugiere comentar la línea de invocación al método BateriaDePruebas.ejecutaTodas() hasta que no hayáis completado el código. El método main() crea un nuevo objeto InterfazUsuario, un nuevo objeto Controlador y ejecuta un bucle que procesa los comandos entrados por teclado.

Package upc.etsetb.poo.basededatos.dominio:

- **BaseDatosException**: Superclase de las clases excepción que gestionará el programa.

Package upc.etsetb.poo.basededatos.dominio.tabla:

- **FilaDatos**: Esta clase representa una fila de datos dentro de una tabla. Su atributo es un mapa en el que las claves son los nombres de las diferentes columnas y los valores, aquellos valores que hay para esa fila en las columnas correspondientes.
- **Tabla**: Esta clase representa una tabla de la base de datos. La tabla la formará una lista de objetos FilaDatos. Cada uno de esos objetos representa una fila de la tabla. Cada objeto Tabla tendrá además un nombre único. Finalmente, todo objeto Tabla tendrá asociado un objeto Esquema con la información correspondiente a las columnas que contiene esa tabla y a cual de ellas es la clave primaria.

- **ClaveInexistenteException**: subclase de BaseDatosException. Se lanza cuando se intenta gestionar una clave que no existe.
- **ValorClaveUnicaException**: subclase de BaseDatosException. Se lanza cuando se intenta añadir a la tabla una fila cuya clave única tiene un valor que ya había sido dada a la clave única de otra fila de esa tabla.

Package upc.etsetb.poo.basededatos.dominio.esquema:

En este package se gestiona el esquema de los objetos Tabla. Como se ha comentado antes un objeto Esquema contiene los detalles de la definición de una tabla (qué columnas hay en esa tabla y cual de ellas es la clave única).

- **Clave**: Esta clase representa una columna dentro del objeto Esquema correspondiente a un objeto Tabla. Los objetos de esta clase tienen como atributos el nombre de la clave y un booleano que indica si dicha clave es una clave única o no.
- **Esquema**: Los objetos de esta clase contienen información de la estructura interna de una tabla. En concreto contienen un mapa de parejas en las que la clave es el nombre de una clave y el valor es un objeto Clave (que como se ha comentado antes contiene ese nombre y un booleano indicativo de si es clave primaria o no).
- **ClaveYaexisteException**: subclase de BaseDatosException. Se lanza cuando se intenta añadir a una tabla una clave que ya existe.

Package upc.etsetb.poo.basededatos.casosdeuso.busqueda:

- **`Criterio`**: Esta clase representa un criterio de búsqueda o eliminación en las filas (objetos FilaDatos) de una Tabla. De entre los atributos de Criterio cabe destacar:
    - `nombreClave`: un objeto String contiene el nombre de una columna de la tabla (una clave de la tabla).
    - `valorAComprobar`: un objeto String.
    - `tipo`: un atributo de tipo int que indica el tipo de criterio de búsqueda: IGUAL o CONTIENE.

  Esta clase es abstracta e incluye el método abstracto

  ```
  public abstract boolean esCumplido(FilaDatos f)
  ```

  que es implementado por todas sus subclases.
  En cada una de esas subclases, el método buscará en el argumento `f` la clave cuyo nombre coíncida con el valor del atributo `nombreClave`, tomará el valor de dicha clave y la comparará con el valor del atributo `valorAComprobar`. En función de la subclase en la que esté implementado este método, dicha comparación permitirá concluir si el criterio se cumple o no.
- **`CriterioIgual`**: En esta subclase, el criterio se cumple si los dos valores son iguales.
- **`CriterioContiene`**: En esta subclase, el criterio se cumple si el valor de una cierta celda de la tabla contiene un cierto string.
- **`CriterioMayorQue`**: En esta subclase, el criterio se cumple si el valor de una cierta celda de la tabla contiene la representación textual de un cierto número y ese número representado es mayor que otro dado.
- **`CriterioMenorQue`**: En esta subclase, el criterio se cumple si el valor de una cierta celda de la tabla contiene la representación textual de un cierto número y ese número representado es menor que otro dado.

Package upc.etsetb.poo.basededatos.casosdeuso:

- **`Controlador`**: Este objeto gestionará las tablas que forman la base de datos. Este objeto interaccionará por un lado con el interfaz de usuario y por otro con las tablas. Del interfaz de usuario recogerá las peticiones del usuario y se encargará de invocar a los métodos pertinentes de los objetos Tabla. Al interfaz de usuario retornará los resultados de dichas operaciones.
- **`ComparadorFilas`**: Implementa la interface `Comparator<FilaDatos>` y compara dos objetos de clase `FilaDatos` en función de una clave. Se usará para ordenar las filas de una tabla según los valores de una cierta clave.
- **`TablaInexistenteException`**: Excepción creada y lanzada cuando se intenta acceder a una tabla inexistente.

El programa incluye además un corrector que puntuará vuestro código de forma automática conforme vayáis desarrollándolo.

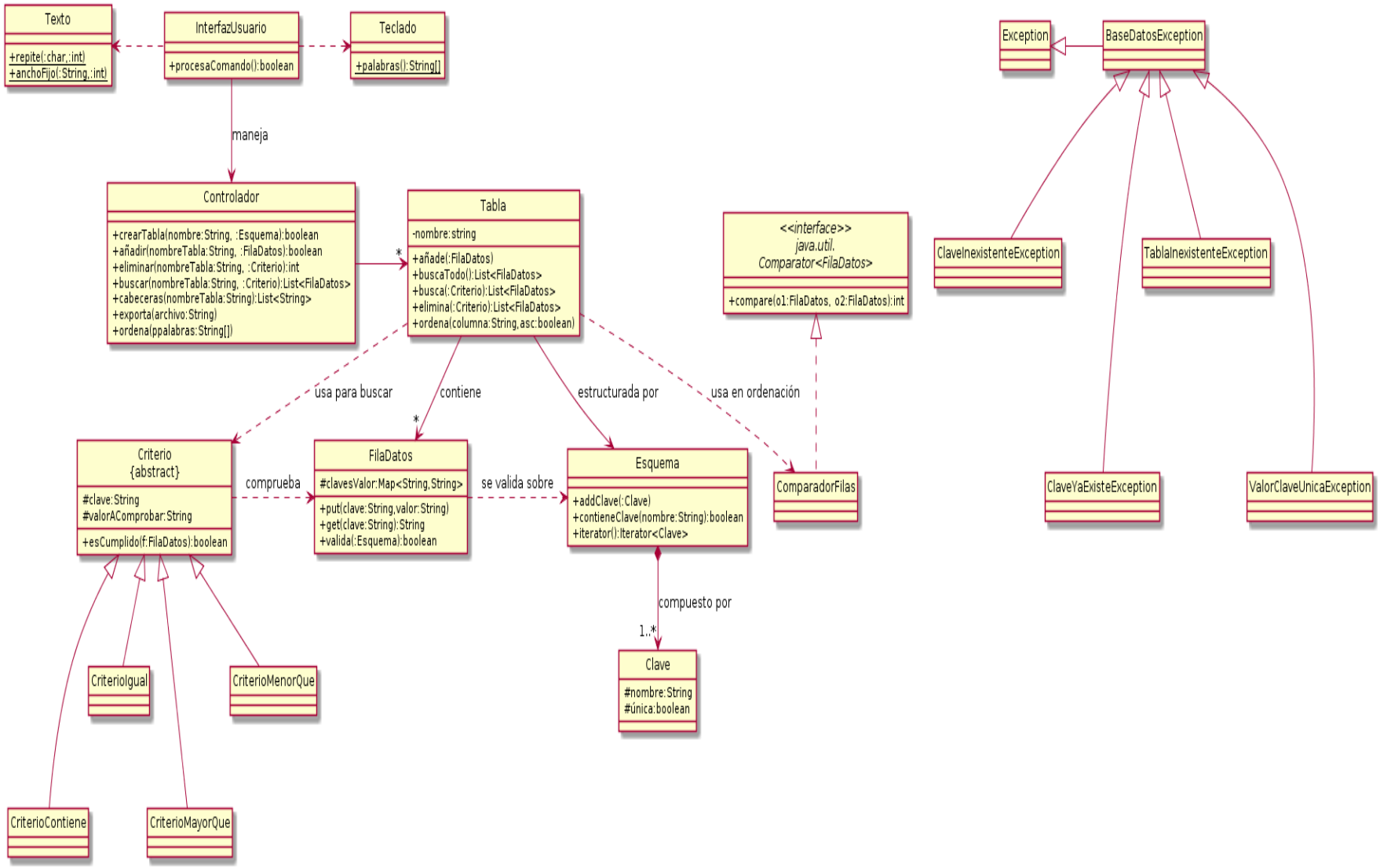
El paquete que contiene este documento contiene además:

1. El javadoc del proyecto, con todos los detalles de lo que tenéis que implementar**. ES DE OBLIGADA LECTURA PARA LA REALIZACIÓN DE LA PRÁCTICA**.
2. El código de la clase `Main`.
3. El código completo de la clase `InterfazUsuario`, incluyendo el correspondiente al tratamiento de los nuevos comandos.

Orden de implementación

Para que el corrector funcione adecuadamente, debéis llevar a cabo la implementación de las clases en el orden siguiente:

1. `Clave`
2. `BaseDatosException`
3. `ClaveYaExisteException`
4. `Esquema`
5. `ClaveInexistenteException`
6. `FilaDatos`
7. `ComparadorFilas`
8. `ValorClaveUnicaException`
9. `Tabla menos métodos:`
   a. `public List<FilaDatos> busca(Criterio criterio)`
   b. `public List<FilaDatos> elimina(Criterio criterio)`
10. `Criterio`
11. `CriterioIgual`
12. `CriterioContiene`
13. `CriterioMayorQue`
14. `CriterioMenorQue`
15. `Métodos de Tabla:`
    a. `public List<FilaDatos> busca(Criterio criterio)`
    b. `public List<FilaDatos> elimina(Criterio criterio)`
16. `TablaInexistenteException`
17. `Controlador`



En el diagrama debéis suponer que hay las siguientes **relaciones de dependencia**, no mostradas porque de hacerlo se cruzarían líneas en el mismo: entre `FilaDatos` y `ClaveInexistenteException`; entre `Esquema` y `ClaveYaExisteException`; entre `Controlador` y `TablaInexistenteException`; entre `Tabla` y `ValorClaveUnicaException`.