



Acceso a Datos – Tarea 04

DESARROLLO DE APLICACIONES MULTIPLATAFORMA

FELIPE RODRIGUEZ

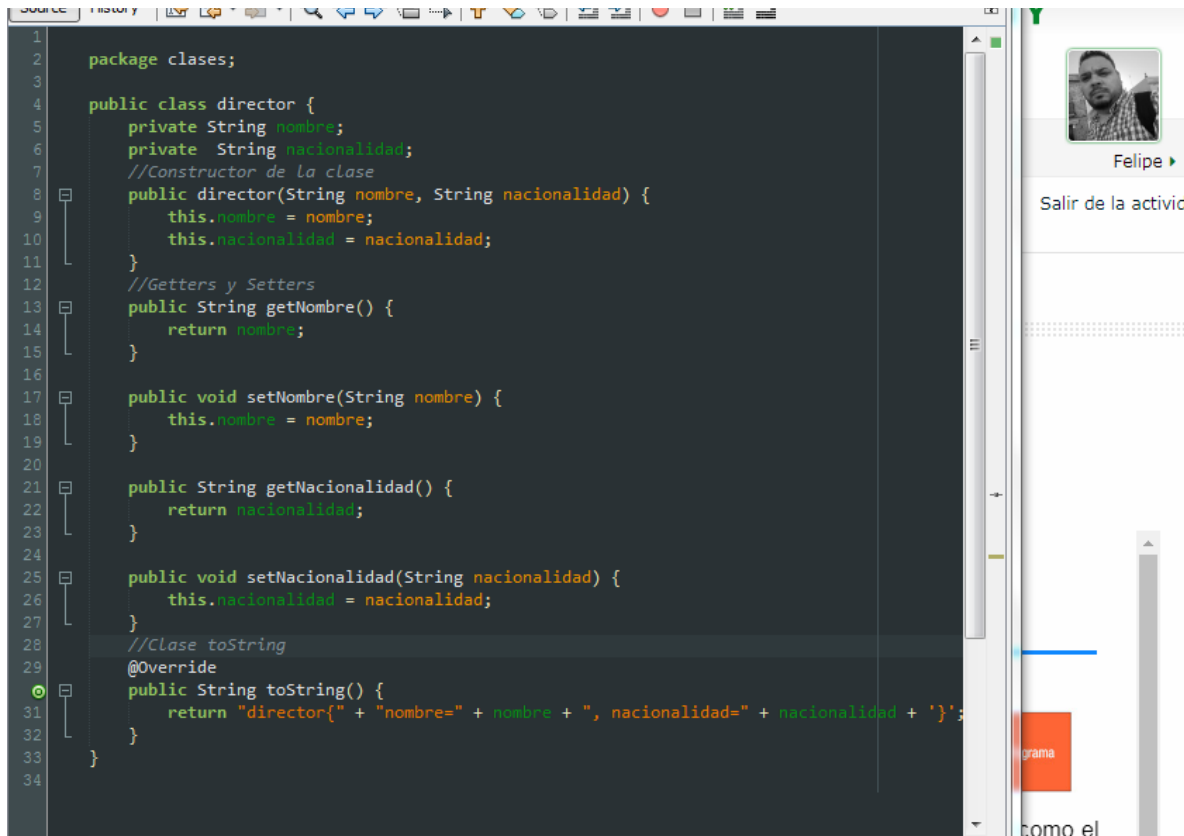
Contenido

Ejercicio 1	2
Creando la clase director:.....	2
Creando la clase película	2
Creando los objetos directores y objetos.....	4
Almacenando las películas	4
Método Main inicial de pruebas	4
Comprobando que funciona (1).....	5
Comprobando que funciona (2).....	6
Ejercicio 2	7
Introduciendo la contraseña.....	7
Borrando todos los datos.....	7
Creando el tipo vivienda, dirección y la función publicidad.....	8
Creando los tipos vivienda y dirección	8
Creando la función publicidad.....	8
Creando la tabla viviendas_alquiler y sus consultas	8
Creando la tabla viviendas_alquiler	8
Insertando registros de ejemplo.....	9
Consulta de todas las ciudades diferentes existentes en la BD	9
Consulta de todas las viviendas dadas por una ciudad	9
Devolviendo el valor publicidad de una vivienda dada.....	9
Probando el funcionamiento de todo.....	10
Consultando la primera vivienda inicial.....	10
Probando otra ciudad y una vivienda aleatoria.....	10
Ejercicio 3	11

Ejercicio 1

NOTA: Obviaremos la parte de la descarga e instalación de la librería Db4o en nuestro proyecto al considerar que ya se encuentra documentado en el temario. Asimismo, no se han realizado tareas de validación de los datos de entrada (se considera que los datos que se van a introducir son válidos), dado que se considera que no son la finalidad de esta tarea y sopesaría más carga de trabajo que la tarea en si misma.

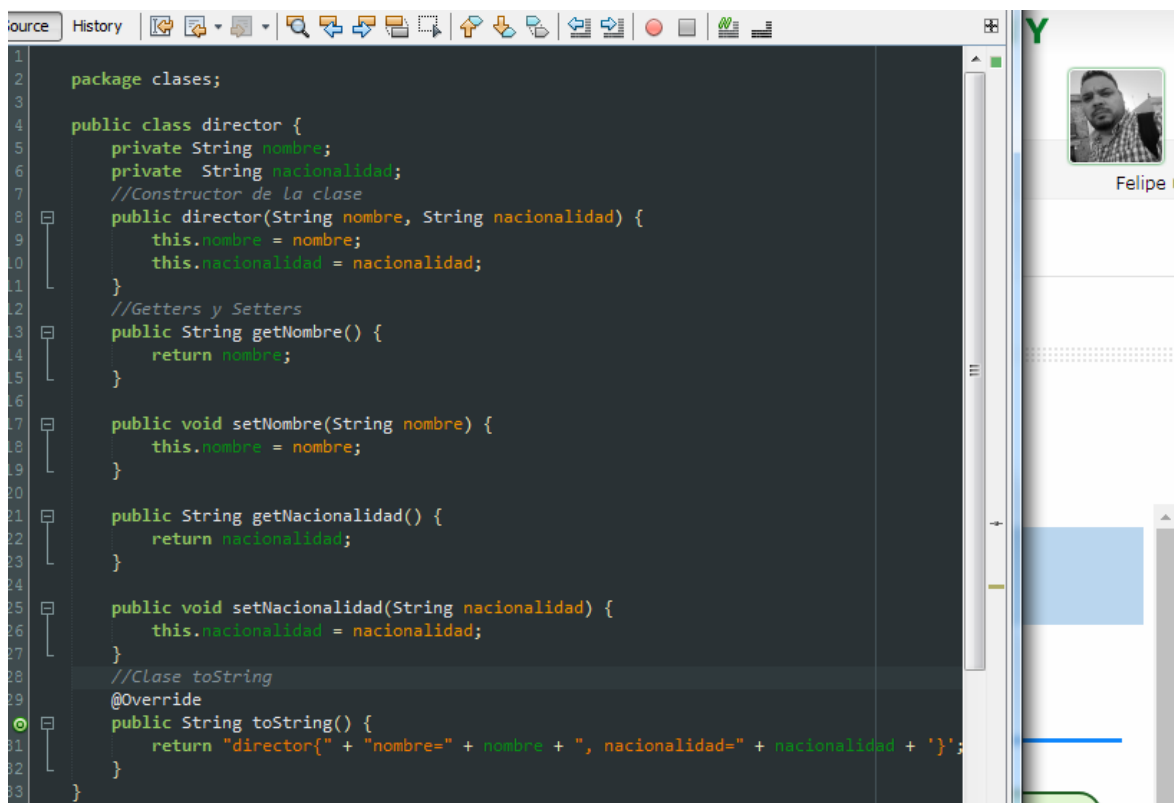
Creando la clase director:



```
1 package clases;
2
3
4 public class director {
5     private String nombre;
6     private String nacionalidad;
7     //Constructor de la clase
8     public director(String nombre, String nacionalidad) {
9         this.nombre = nombre;
10        this.nacionalidad = nacionalidad;
11    }
12    //Getters y Setters
13    public String getNombre() {
14        return nombre;
15    }
16
17    public void setNombre(String nombre) {
18        this.nombre = nombre;
19    }
20
21    public String getNacionalidad() {
22        return nacionalidad;
23    }
24
25    public void setNacionalidad(String nacionalidad) {
26        this.nacionalidad = nacionalidad;
27    }
28    //Clase toString
29    @Override
30    public String toString() {
31        return "director{" + "nombre=" + nombre + ", nacionalidad=" + nacionalidad + '}';
32    }
33 }
34
```

On the right side of the IDE, there is a user profile for 'Felipe' with a 'Salir de la actividad' button and a 'Programa' button at the bottom.

Creando la clase película



The image shows a screenshot of an IDE window with a dark theme. The main editor displays Java code for a class named 'director'. The code includes package declarations, private attributes for 'nombre' and 'nacionalidad', a constructor, getters, setters, and a 'toString' method. A sidebar on the right shows a user profile for 'Felipe' with a small circular avatar. The IDE interface includes a top toolbar with various icons for file operations, editing, and running, and a left margin with line numbers and a class explorer tree.

```
1 package clases;
2
3
4 public class director {
5     private String nombre;
6     private String nacionalidad;
7     //Constructor de la clase
8     public director(String nombre, String nacionalidad) {
9         this.nombre = nombre;
10        this.nacionalidad = nacionalidad;
11    }
12    //Getters y Setters
13    public String getNombre() {
14        return nombre;
15    }
16
17    public void setNombre(String nombre) {
18        this.nombre = nombre;
19    }
20
21    public String getNacionalidad() {
22        return nacionalidad;
23    }
24
25    public void setNacionalidad(String nacionalidad) {
26        this.nacionalidad = nacionalidad;
27    }
28    //Clase toString
29    @Override
30    public String toString() {
31        return "director{" + "nombre=" + nombre + ", nacionalidad=" + nacionalidad + '}';
32    }
33 }
```


Creando los objetos directores y objetos

```
db.close();
}
}

public static void almacenarPelículas(ObjectContainer db){
    //Creamos tres directores
    director d1 = new director("Steven Spielberg", "USA");
    director d2 = new director("Paul Verhoeven", "Holanda");
    director d3 = new director("Duncan Jones", "Reino Unido");

    //Creamos seis películas
    pelicula p1 = new pelicula("Ready Player One", d1, 140);
    pelicula p2 = new pelicula("Robocop", d2, 103);
    pelicula p3 = new pelicula("Warcraft: El Origen", d3, 123);
    pelicula p4 = new pelicula("Encuentros en la tercera fase", d1, 135);
    pelicula p5 = new pelicula("Desafío Total", d2, 109);
    pelicula p6 = new pelicula("Moon", d3, 93);

    //Almacenamos las películas
}
```




Felipe ▶

Salir de la actividad

Almacenando las películas

```
pelicula p4 = new pelicula("Encuentros en la tercera fase", d1, 135);
pelicula p5 = new pelicula("Desafío Total", d2, 109);
pelicula p6 = new pelicula("Moon", d3, 93);

//Almacenamos las películas
db.store(p1);
db.store(p2);
db.store(p3);
db.store(p4);
db.store(p5);
db.store(p6);
```




Felipe ▶

Salir de la actividad

Método Main inicial de pruebas

```
/**
 * Método main, que iniciará el programa e introducirá valores en la BD
 * @param args
 */
public static void main(String[] args) {
    //Conectamos con la base de datos
    ObjectContainer db;
    db = Db4oEmbedded.openFile(Db4oEmbedded.newConfiguration(), "películas.db4o");

    try {
        //Llamamos al método que introducirá valores en la BD
        almacenarPelículas(db);
    } catch (Exception e) {
        //En caso de errores que muestre mensaje por consola
        System.err.print(e);
    } finally {
        //Cerramos las operaciones con la BD
        db.close();
    }
}
```

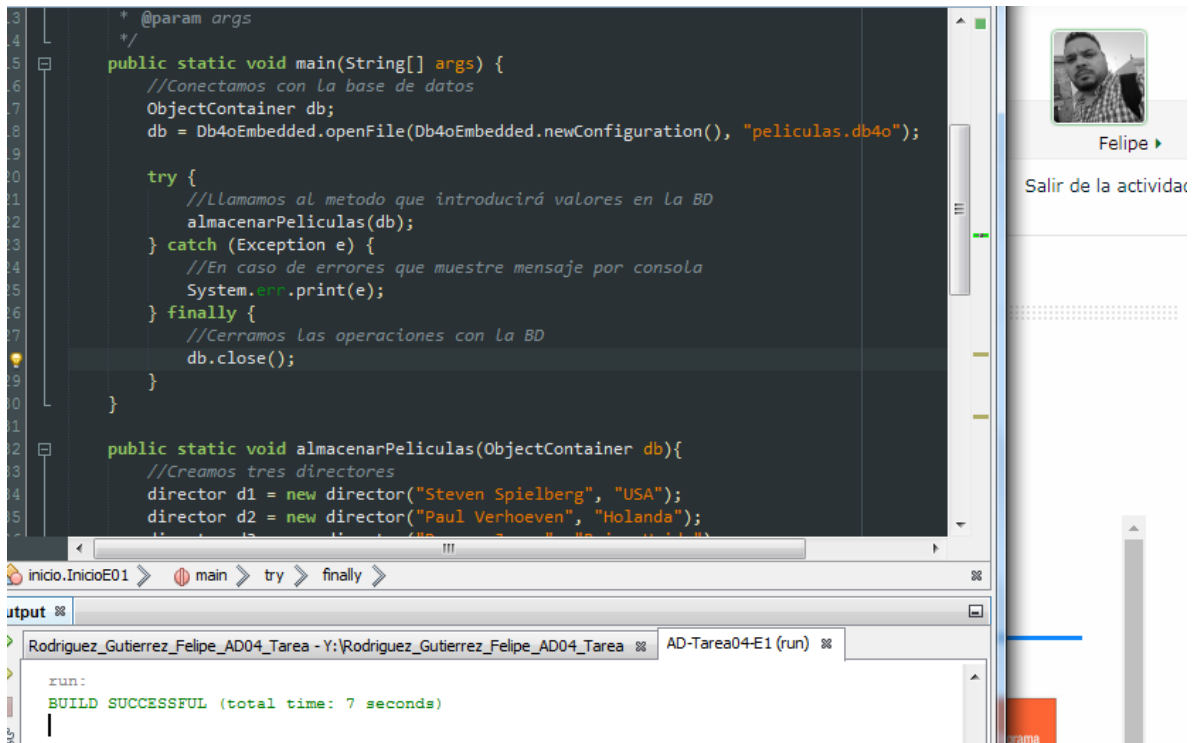


Felipe ▶

Salir de la actividad

Comprobando que funciona (1)

Con este método inicial, si ejecutamos el mismo, podremos observar por consola que crea satisfactoriamente la base de datos.



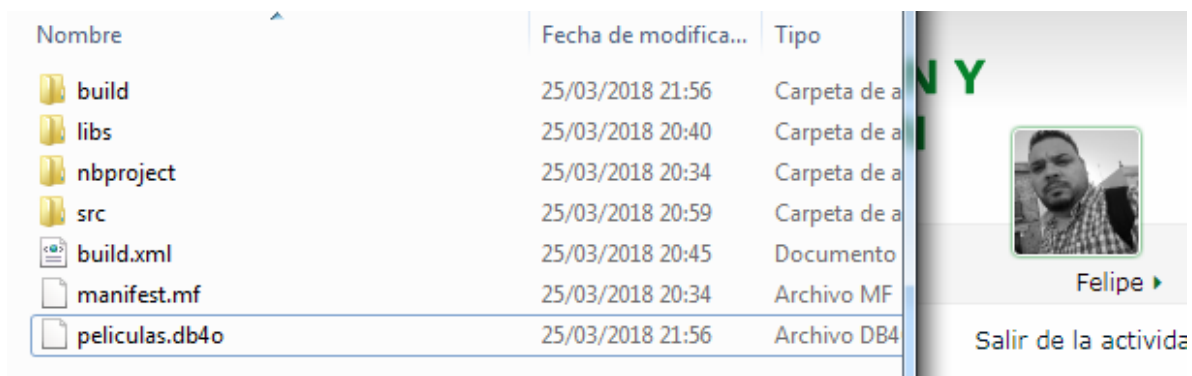
The screenshot shows an IDE with a Java file. The code defines a `main` method that connects to a database, calls `almacenarPelículas`, and prints any exceptions. The `almacenarPelículas` method creates two `director` objects. The output window shows a successful build.

```
3  * @param args
4  */
5  public static void main(String[] args) {
6      //Conectamos con la base de datos
7      ObjectContainer db;
8      db = Db4oEmbedded.openFile(Db4oEmbedded.newConfiguration(), "películas.db4o");
9
10     try {
11         //Llamamos al metodo que introducirá valores en la BD
12         almacenarPelículas(db);
13     } catch (Exception e) {
14         //En caso de errores que muestre mensaje por consola
15         System.err.print(e);
16     } finally {
17         //Cerramos las operaciones con la BD
18         db.close();
19     }
20 }
21
22 public static void almacenarPelículas(ObjectContainer db){
23     //Creamos tres directores
24     director d1 = new director("Steven Spielberg", "USA");
25     director d2 = new director("Paul Verhoeven", "Holanda");
26     //Insertamos los directores en la BD
27     db.store(d1);
28     db.store(d2);
29 }
```

Output:

```
run:
BUILD SUCCESSFUL (total time: 7 seconds)
```

Si miramos en el directorio de nuestro proyecto, podremos observar que se ha creado la misma en su interior como se muestra en la imagen siguiente.

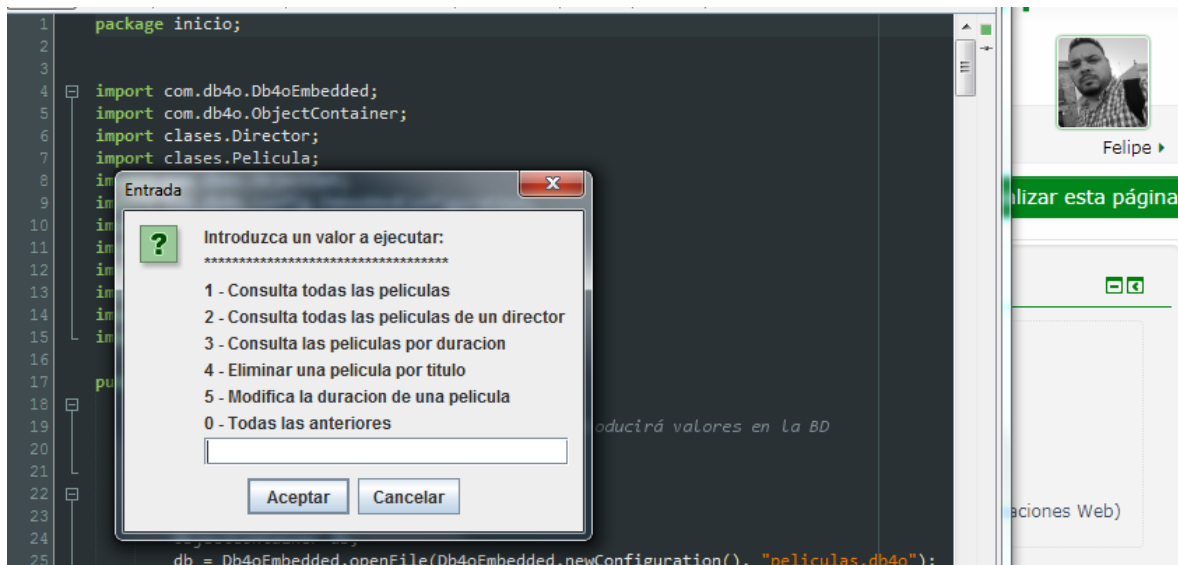


The screenshot shows a file explorer with the following files and folders:

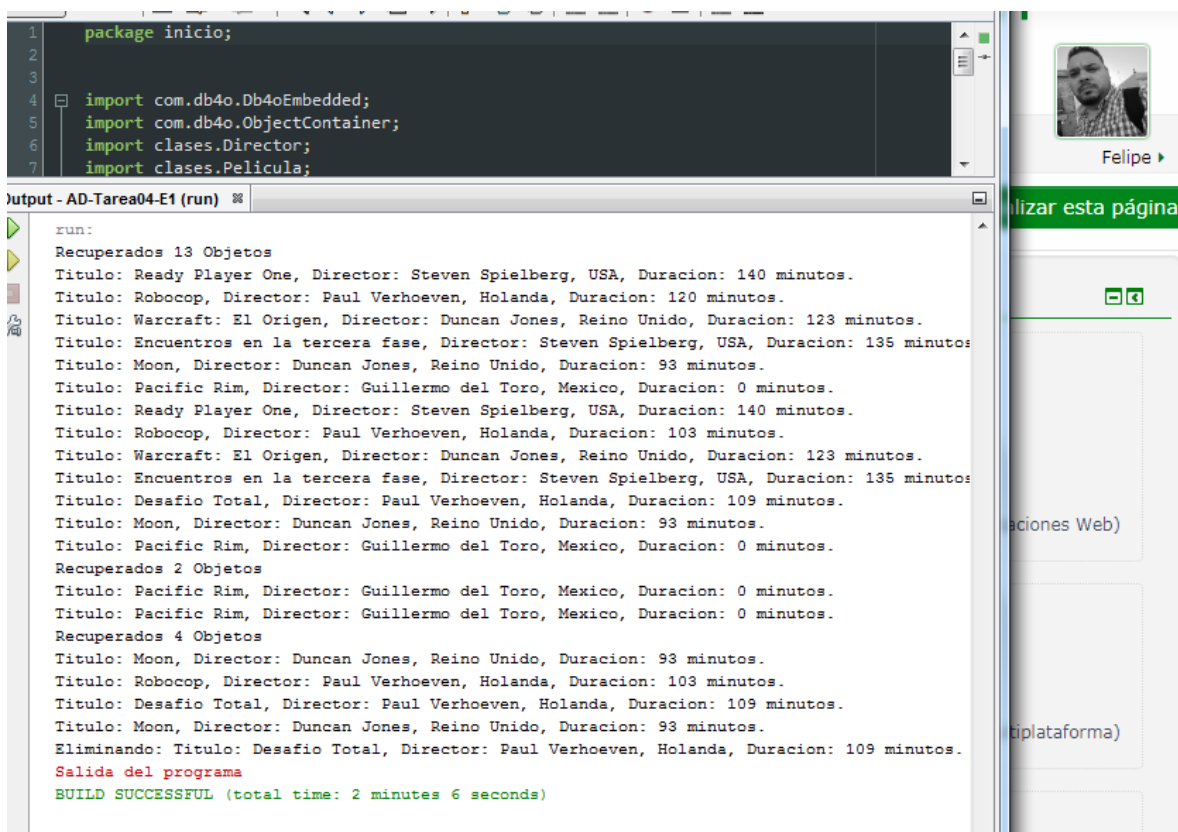
Nombre	Fecha de modifica...	Tipo
build	25/03/2018 21:56	Carpeta de a
libs	25/03/2018 20:40	Carpeta de a
nbproject	25/03/2018 20:34	Carpeta de a
src	25/03/2018 20:59	Carpeta de a
build.xml	25/03/2018 20:45	Documento
manifest.mf	25/03/2018 20:34	Archivo MF
películas.db4o	25/03/2018 21:56	Archivo DB4

Ahora, añadiremos el resto de funcionalidades hasta estar completa, las cuales al ejecutarse las mismas, mostrarán los resultados requeridos en pantalla.

Comprobando que funciona (2)



Ejecutando el punto “Todas las anteriores”, mostrará por consola todas las acciones de golpe. Los resultados mostrados pueden ser diferentes a los que se puedan ver en otras ejecuciones (el ejercicio se distribuirá sin el archivo películas.db4o eliminado para que se inicie desde cero).

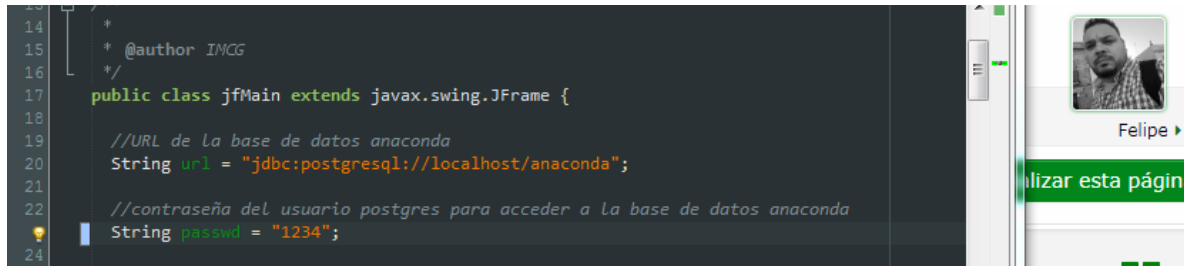


Ejercicio 2

NOTA: Obviaremos la parte de la descarga e instalación de PostgreSQL tanto en el sistema como en nuestro proyecto al considerar que ya se encuentra documentado en el temario. Asimismo, no se han realizado tareas de validación de los datos de entrada (se considera que los datos que se van a introducir son válidos), dado que se considera que no son la finalidad de esta tarea y sopesaría más carga de trabajo que la tarea en si misma.

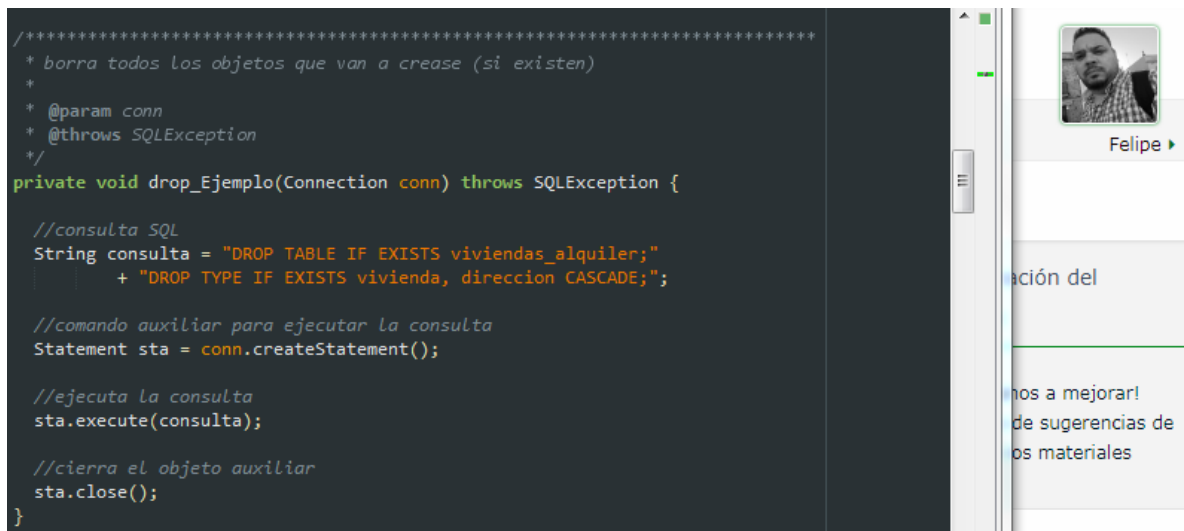
Introduciendo la contraseña

Introduciendo la contraseña de PostgreSQL, que se ha definido en durante la instalación. En nuestro caso se ha utilizado la última versión disponible del gestor, 10.3. En nuestro caso, la contraseña que vamos a usar ya estaba definida en el proyecto.



Borrando todos los datos

Este método viene predefinido con el proyecto de serie. No se le realizan cambios.



Creando el tipo vivienda, dirección y la función publicidad

NOTA: Se han modificado algunas líneas de orden y se han suprimido otras con motivo de aligerar líneas de código que se consideran innecesarias.

Creando los tipos vivienda y dirección

```
//+++++
//escribe la consulta SQL para construir el tipo 'vivienda', el tipo
//'direccion', y la función 'publicidad', de acuerdo con las
//indicaciones dadas
sta.execute("CREATE TYPE vivienda AS("
+ "planta INTEGER,"
+ "metros_2 INTEGER,"
+ "num_habitaciones INTEGER,"
+ "num_banios INTEGER,"
+ "arrendador VARCHAR(25));"

sta.execute("CREATE TYPE direccion AS("
+ "calle VARCHAR(40),"
+ "ciudad VARCHAR(25),"
+ "provincia VARCHAR(25),"
+ "codigo_postal VARCHAR(5));"
```

Creando la función publicidad

```
//+++++
//crea la función que transforma el tipo estructurado en una cadena
sta.execute("CREATE FUNCTION publicidad(vivienda) RETURNS varchar AS "
+ "$SELECT 'Se alquila piso de '||$1.metros_2|| ' metros cuadrados"
+ " en planta '||$1.planta||', con '||$1.num_habitaciones||' habitaciones y '"
+ "|| $1.num_banios||' baños';$$"
+ "LANGUAGE SQL");"
```

Creando la tabla viviendas_alquiler y sus consultas

NOTA: Se han modificado algunas líneas de orden y se han suprimido otras con motivo de aligerar líneas de código que se consideran innecesarias.

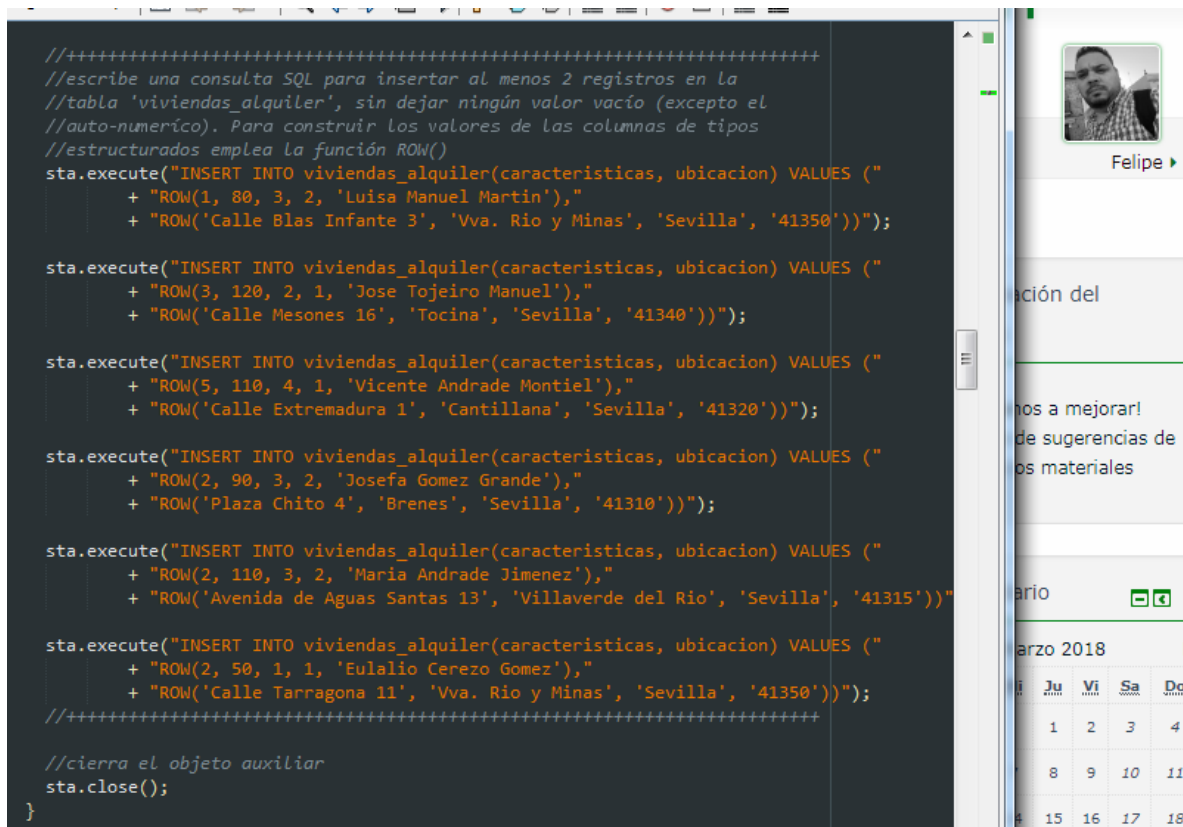
Creando la tabla viviendas_alquiler

```
/* crea la tabla 'viviendas_alquiler'
*
* @param conn
* @throws SQLException
*/
private void crear_Tabla(Connection conn) throws SQLException {
//comando auxiliar para ejecutar la consulta
Statement sta = conn.createStatement();

//+++++
//escribe la consulta SQL para crear la tabla 'viviendas_alquiler' de acuerdo
//con las indicaciones dadas
//ejecuta la consulta
sta.execute("CREATE TABLE viviendas_alquiler("
+ "vivienda_id serial, "
+ "caracteristicas vivienda, "
+ "ubicacion direccion);"
//+++++

//cierra el objeto auxiliar
sta.close();
}
```

Insertando registros de ejemplo



```
//+++++
//escribe una consulta SQL para insertar al menos 2 registros en la
//tabla 'viviendas_alquiler', sin dejar ningún valor vacío (excepto el
//auto-numérico). Para construir los valores de las columnas de tipos
//estructurados emplea la función ROW()
sta.execute("INSERT INTO viviendas_alquiler(caracteristicas, ubicacion) VALUES ("
+ "ROW(1, 80, 3, 2, 'Luisa Manuel Martin'),"
+ "ROW('Calle Blas Infante 3', 'Vva. Río y Minas', 'Sevilla', '41350'))");

sta.execute("INSERT INTO viviendas_alquiler(caracteristicas, ubicacion) VALUES ("
+ "ROW(3, 120, 2, 1, 'Jose Tojeiro Manuel'),"
+ "ROW('Calle Mesones 16', 'Tocina', 'Sevilla', '41340'))");

sta.execute("INSERT INTO viviendas_alquiler(caracteristicas, ubicacion) VALUES ("
+ "ROW(5, 110, 4, 1, 'Vicente Andrade Montiel'),"
+ "ROW('Calle Extremadura 1', 'Cantillana', 'Sevilla', '41320'))");

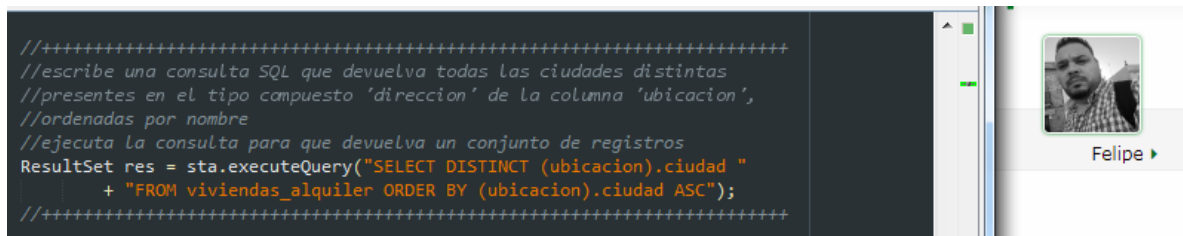
sta.execute("INSERT INTO viviendas_alquiler(caracteristicas, ubicacion) VALUES ("
+ "ROW(2, 90, 3, 2, 'Josefa Gomez Grande'),"
+ "ROW('Plaza Chito 4', 'Brenes', 'Sevilla', '41310'))");

sta.execute("INSERT INTO viviendas_alquiler(caracteristicas, ubicacion) VALUES ("
+ "ROW(2, 110, 3, 2, 'Maria Andrade Jimenez'),"
+ "ROW('Avenida de Aguas Santas 13', 'Villaverde del Río', 'Sevilla', '41315'))");

sta.execute("INSERT INTO viviendas_alquiler(caracteristicas, ubicacion) VALUES ("
+ "ROW(2, 50, 1, 1, 'Eulalio Cerezo Gomez'),"
+ "ROW('Calle Tarragona 11', 'Vva. Río y Minas', 'Sevilla', '41350'))");
//+++++

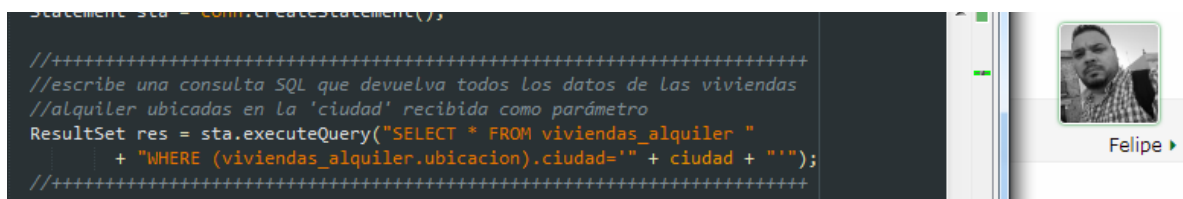
//cierra el objeto auxiliar
sta.close();
}
```

Consulta de todas las ciudades diferentes existentes en la BD



```
//+++++
//escribe una consulta SQL que devuelva todas las ciudades distintas
//presentes en el tipo compuesto 'direccion' de la columna 'ubicacion',
//ordenadas por nombre
//ejecuta la consulta para que devuelva un conjunto de registros
ResultSet res = sta.executeQuery("SELECT DISTINCT (ubicacion).ciudad "
+ "FROM viviendas_alquiler ORDER BY (ubicacion).ciudad ASC");
//+++++
```

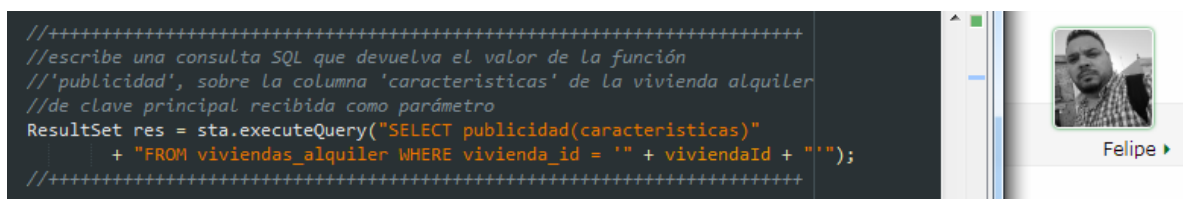
Consulta de todas las viviendas dadas por una ciudad



```
Statement sta = con.createStatement();

//+++++
//escribe una consulta SQL que devuelva todos los datos de las viviendas
//alquiler ubicadas en la 'ciudad' recibida como parámetro
ResultSet res = sta.executeQuery("SELECT * FROM viviendas_alquiler "
+ "WHERE (viviendas_alquiler.ubicacion).ciudad='" + ciudad + "'");
//+++++
```

Devolviendo el valor publicidad de una vivienda dada



```
//+++++
//escribe una consulta SQL que devuelva el valor de la función
//'publicidad', sobre la columna 'caracteristicas' de la vivienda alquiler
//de clave principal recibida como parámetro
ResultSet res = sta.executeQuery("SELECT publicidad(caracteristicas)"
+ "FROM viviendas_alquiler WHERE vivienda_id = '" + viviendaId + "'");
//+++++
```

Probando el funcionamiento de todo

NOTA: Se le ha añadido una línea al código para centrar el cuadro en la ventana del sistema.

Consultando la primera vivienda inicial

Oferta de vivienda alquiler en la ciudad de: Brenes

vivienda_id	caracteristicas	ubicacion
4	(2,90,3,2,"Josefa Gomez Grande")	("Plaza Chito 4",Brenes,Sevilla,41...

Para la vivienda seleccionada...

Consultar el valor devuelto por la función 'publicidad' sobre el tipo 'vivienda' de la columna 'caracteristicas'

Se alquila piso de 90 metros cuadrados en planta 2, con 3 habitaciones y 2 baños

Probando otra ciudad y una vivienda aleatoria

Oferta de vivienda alquiler en la ciudad de: Vva. Ri...

vivienda_id	caracteristicas	ubicacion
1	(1,80,3,2,"Luisa Manuel Martin")	("Calle Blas Infante 3", "Vva. Rio y ...
6	(2,50,1,1,"Eulalio Cerezo Gomez")	("Calle Tarragona 11", "Vva. Rio y ...

Para la vivienda seleccionada...

Consultar el valor devuelto por la función 'publicidad' sobre el tipo 'vivienda' de la columna 'caracteristicas'

Se alquila piso de 50 metros cuadrados en planta 2, con 1 habitaciones y 1 baños

Ejercicio 3

Conclusiones personales

Como conclusiones personales la verdad es que me ha gustado esta tarea, me ha resultado muy accesible con los ejemplos a descargar. En esta ocasión ha sido más intuitiva, considero que es muy muy similar a SQL, cosa que facilita mucho la tarea. Trabajar con objetos en esta ocasión me ha parecido mucho más intuitivo y sencillo.

Mejoras en mi tarea:

Como mejoras en mi programa, principalmente haría en la primera tarea una verificación de los datos de entrada, pero nos desviaríamos de la línea de esta tarea y de este módulo. Tal vez aplicar una interfaz gráfica en el primer ejercicio.

Dificultades que te has encontrado:

Sin querer parecer presuntuoso, no me ha supuesto gran dificultad siguiendo los ejemplos y el temario (consideré Hibernate más enrevesado). Tal vez la consulta de las ciudades se atragantó el tiempo de irme a hacer una café.

Opinión personal del trabajo realizado:

Añadir a mi opinión personal, pues me ha gustado trabajar con db4o por poder incorporar una base de datos funcional en su propia aplicación con solo adjuntar la librería, aunque su "deprecación" por no existir una línea continuista en esta aplicación, hace que alguien que quiera desarrollar una aplicación quiera usar estas bases de datos que, pese a que funcionan, no tienen evolución. Con PostgreSQL se conoce que hay mercado laboral y es muy solicitado en ofertas de empleo. La única "pega" que le encuentro es que nuestra aplicación depende de un servidor de base de datos externo, o al menos es lo que llevo visto con el curso como en esta tarea.



Re: Conclusiones personales
de Rodríguez Gutiérrez, Felipe - miércoles, 28 de marzo de 2018, 18:20

Como conclusiones personales la verdad es que me ha gustado esta tarea, me ha resultado muy accesible con los ejemplos a descargar. En esta ocasión ha sido más intuitiva, considero que es muy muy similar a SQL, cosa que facilita mucho la tarea. Trabajar con objetos en esta ocasión me ha parecido mucho más intuitivo y sencillo.

Mejoras en mi tarea:

Como mejoras en mi programa, principalmente haría en la primera tarea una verificación de los datos de entrada, pero nos desviaríamos de la línea de esta tarea y de este módulo. Tal vez aplicar una interfaz gráfica en el primer ejercicio.

Dificultades que te has encontrado:

Sin querer parecer presuntuoso, no me ha supuesto gran dificultad siguiendo los ejemplos y el temario (consideré Hibernate más enrevesado). Tal vez la consulta de las ciudades se atragantó el tiempo de irme a hacer una café.

Opinión personal del trabajo realizado:

Añadir a mi opinión personal, pues me ha gustado trabajar con db4o por poder incorporar una base de datos funcional en su propia aplicación con solo adjuntar la librería, aunque su "deprecación" por no existir una línea continuista en esta aplicación, hace que alguien que quiera desarrollar una aplicación quiera usar estas bases de datos que, pese a que funcionan, no tienen evolución. Con PostgreSQL se conoce que hay mercado laboral y es muy solicitado en ofertas de empleo. La única "pega" que le encuentro es que nuestra aplicación depende de un servidor de base de datos externo, o al menos es lo que llevo visto con el curso como en esta tarea.

[Enlace permanente](#) | [Marcar como no leído](#) | [Mostrar mensaje anterior](#) | [Editar](#) | [Borrar](#) | [Responder](#)