

Sistemas de Información

Tecnologías y Arquitectura

Javier Tuya
Grupo de Investigación en Ingeniería del Software
<http://giis.uniovi.es>

Curso 2022-2023



1

Contenidos

■ Proyecto de ejemplo

- ☐ <https://github.com/javiertuya/samples-test-dev>
- ☐ Javadoc: <https://javiertuya.github.io/samples-test-dev>

1. Conexión JDBC a base de datos (giis.demo.jdbc)

- ☐ Conexiones y consultas
- ☐ Manejo de excepciones y parámetros
- ☐ Apache Commons DbUtils y Lombok

2. Arquitectura de aplicación (giis.demo.tkrun)

- ☐ Patrón MVC y estructura proyecto Maven
- ☐ Descripción funcionalidad mostrada como ejemplo
- ☐ Descripción código del Modelo, Vista y Controlador
- ☐ Configuración de la base de datos

J. Tuya, (2022-2023)

Tecnologías y Arquitectura

2

2

Parte 1:

Conexión JDBC a base de datos

■ Objetivo:

- Repaso de los mecanismos para acceder a la base de datos utilizando jdbc
- Componentes adicionales de Apache Commons que simplifican el acceso

■ Código fuente en paquete giis.demo.jdbc

J. Tuya, (2021-2022)

Tecnologías y Arquitectura

3

3

```
package giis.demo.jdbc;
import java.sql.*;

/* Ejemplos de acceso a una base de datos con conexion JDBC y bases de datos */
public class DemoJdbc {
    //informacion de conexion a la base de datos utilizada
    public static final String DRIVER="org.sqlite.JDBC";
    public static final String URL="jdbc:sqlite:DemoDB.db";
    //datos para SQLServer:
    //com.microsoft.sqlserver.jdbc.SQLServerDriver
    //jdbc:sqlserver://localhost:1433;DatabaseName=*****;user=*****;password=*****
    private Logger log=Logger.getLogger(DemoJdbc.class);

    /* Demo basico de acceso a bases de datos, parte 1: */
    public void demo1Basic() {
        try {
            //instalacion de la clase que contiene al driver (basta con hacerlo una vez)
            //el driver (en este caso sqljdbc.jar) ha sido descargado y puesto en el classpath
            //Esto normalmente no es necesario, desde a partir de JDBC 4.0 los drivers se autoregistran
            Class.forName(DRIVER);
            //Definicion de la cadena de conexion, especifica para cada gestor de bases de datos
            String connString=URL;

            //Ejecuta acciones de actualizacion: insertar datos en una tabla
            Connection cn=DriverManager.getConnection(connString);
            Statement stmt = cn.createStatement();

            try {
                stmt.executeUpdate("drop table if exists test");
            } catch (SQLException e) {
                //ignora excepcion, que se causara si la tabla no existe en la bd (p.e. al ejecutar la primera vez)
            }

            stmt.executeUpdate("create table test(id int not null, id2 int, text varchar(32))");
            stmt.executeUpdate("insert into test(id,id2,text) values(1,null,'abc')");
            stmt.executeUpdate("insert into test(id,id2,text) values(2,9999,'xyz')");
            stmt.close(); //no olvidar cerrar estos objetos
            cn.close();
        }
    }
}
```

Parámetros de configuración:
-driver: particular del SGBD usado
-url: especifica cuál es la BD (sintaxis dependiente del SGBD)

Instanciación de la clase y creación de conexión

Statement es el objeto básico que debe existir para manipular los datos

Ejecución de SQL
(if exists en el drop es dependiente del SGBD, puede no existir en otros)

No olvidar cerrar los objetos de conexión y statement

J. Tuya, (2021-2022)

Tecnologías y Arquitectura

4

4

Conexiones y consultas

Como siempre, crear conexión y statement

executeQuery para ejecutar SQL que devuelve datos en un ResultSet

Next es el iterador sobre un ResultSet. Cada vez que se ejecuta, avanza la posición de lectura. Cuando es falso indica que no hay más datos

Cada campo se obtiene con un método get* getString siempre devuelve el valor como string

Precaución con lectura de valores null de la BD.

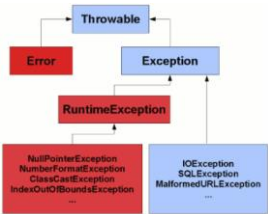
```
//Consulta todas las filas a partir del resultado de una query SQL
cn=DriverManager.getConnection(connString);
stmt=cn.createStatement();
ResultSet rs=stmt.executeQuery("select id,id2,text from test order by id desc");
while (rs.next()) { //cada vez que se llama
    int id=rs.getInt(1); //obtencción de un valor indicando el nombre de la columna
    String id2=rs.getString(2); //obtencción de un valor indicando el nombre de la columna
    if (rs.wasNull()) //comprobación de valores nulos (respecto del último get realizado)
        id2="NULO";
    String text=rs.getString("text"); //obtencción de un valor indicando el nombre de la columna
    log.info("demo1B "+id+" "+id2+" "+text);
}
rs.close();
stmt.close();
cn.close();
```

Manejo de excepciones

Siempre hay que cerrar todos los objetos que se abre, Que a veces es difícil (tenemos Conection, Statement, ResultSet) El "try with resources" realiza todo esto en un solo try Asegurando que pase lo que pase se cierra todo (sin necesidad de usar finally)

```
public void demo2TryWithResources() {
    createTable();
    //En un mismo try se pueden poner diferentes sentencias que se gestionan recursos que hay que cerrar
    try (Connection cn=DriverManager.getConnection(URL);
        Statement stmt=cn.createStatement();
        ResultSet rs=stmt.executeQuery("select id,id2,text from test order by id desc")) {
        while (rs.next()) { //cada vez que se llama rs.next() avanza el cursor a una fila
            int id=rs.getInt(1); //obtencción de un valor con un tipo de dato especificado, indicando el numero de columna
            String id2=rs.getString(2); //obtencción de un valor como string, aunque sea entero
            if (rs.wasNull()) //comprobación de valores nulos (respecto del último get realizado)
                id2="NULO";
            String text=rs.getString("text"); //obtencción de un valor indicando el nombre de la columna
            log.info("demo2TryWithResources: "+id+" "+id2+" "+text);
        }
    } catch (SQLException e) {
        throw new UnexpectedException(e);
    }
}
```

Así capturamos solamente una vez la excepción. Diferenciar tipos de excepciones: https://www.javamex.com/tutorials/exceptions/exceptions_hierarchy.shtml



Parámetros

Para pasar parámetros usar PreparedStatement (evita problemas de seguridad por inyección de código)

```
public void demo3Parameters() {
    createTable();
    //En vez de crear un Statement y pasar el sql en executeQuery,
    //se crea un PreparedStatement con el sql, luego se le ponen los parámetros y finalmente se ejecuta
    try (Connection cn=DriverManager.getConnection(URL);
        PreparedStatement pstmt=cn.prepareStatement("select id,id2,text from test where id=?")) {
        pstmt.setInt(1, 2); // pone valor 2 en el primer (y unico) parametro
        try (ResultSet rs=pstmt.executeQuery()) {
            while (rs.next()) {
                log.info("demo3Parameters: "+rs.getInt(1)+" "+rs.getInt(2)+" "+rs.getString(3));
            }
        }
    } catch (SQLException e) {
        throw new UnexpectedException(e);
    }
    //de forma similar se pueden ejecutar acciones de actualizacion sobre el PreparedStatement
}
```

Se instancia la PreparedStatement usando ? como placeholder para los valores

Luego se asignan los valores, una sentencia set* por cada parámetro
La primera columna en SQL siempre empieza por 1

J. Tuya, (2021-2022)

Tecnologías y Arquitectura

7

7

Apache Commons DbUtils

```
package g11s.demo.jdbc;
public class Entity {
    private Integer id;
    private Integer id2;
    private String text;

    public Integer getId() { return this.id; }
    public Integer getId2() { return this.id2; }
    public String getText() { return this.text; }
    public void setId(Integer value) { this.id = value; }
    public void setId2(Integer value) { this.id2 = value; }
    public void setText(String value) { this.text = value; }
}

public void demo4DbUtils() {
    createTable();
    //El siguiente ejemplo define un handler para los resultsets y cada una de las columnas se mapea a un atributo de una consulta
    //como una lista de objetos (POJO) que se puede usar Apache Commons DbUtils para procesar los resultados de la consulta que seria necesar
    //para leer un resultSet y poner los resultados en los objetos.
    Connection conn=null;
    List<Entity> pojoList; //lista de objetos
    try {
        conn=DriverManager.getConnection(URL);
        //declara el handler que permitira obtener una lista de objetos de la clase indicada
        BeanListHandler<Entity> beanListHandler=new BeanListHandler<>(Entity.class);
        //Declara el runner que ejecutara la consulta
        QueryRunner runner=new QueryRunner();
        //ejecuta la consulta, el ultimo argumento es el parametro (lista variable si hay mas)
        String sql="select id,id2,text from test where id=?";
        pojoList=runner.query(conn, sql, beanListHandler, 2);
    } catch (SQLException e) {
        throw new UnexpectedException(e);
    } finally {
        DbUtils.closeQuietly(conn); //usar este metodo para cerrar bien todos los objetos creados
    }
}
```

Para evitar recorrer manualmente los resultsets y cada una de las columnas se puede usar Apache Commons DbUtils

Basta crear un handler en este caso para objetos POJO (beans). En este caso, un objeto de la clase Entity

y un QueryRunner que será el que ejecute la query

Al ejecutar el runner con la consulta indicada, se tendrá una lista de objetos de la clase utilizada con los valores obtenidos de la BD

Método propio para cerrar bien todos los objetos creados

NOTA: Requiere definir clases para las entidades representadas en cada tabla con sus correspondientes getters y setters

J. Tuya, (2021-2022)

Tecnologías y Arquitectura

8

8



Apache Con

IMPORTANTE: Cuando se usan estos componentes debe mantenerse de forma estricta el convenio de capitalización de Java:

- Capitalizar todas las palabras que forman un identificador excepto la primera letra de nombres de métodos y variables.
 - No utilizar subrayados
- Seguir también estos criterios en nombres de tablas y campos de la BD

Si no definimos un objeto concreto para los valores a obtener de la BD
Podemos realizarlo de forma genérica para obtener todos los valores
En un map

```
List<Map<String,Object>> mapList; //lista de maps que seran devueltos por la query
```

```
try {
    conn=DriverManager.getConnection(URL);
    String sql="select id,id2,text from test where id?";
    mapList=new QueryRunner().query(conn, sql, new MapListHandler(),1);
} catch (SQLException e) {
    throw new UnexpectedException(e);
} finally {
    DbUtils.closeQuietly(conn);
}
for (Map<String,Object> item : mapList)
    log.info("demo4DbUtils (Map): "+item.get("id")+" "+item.get("id2")+" "+item.get("text"));
```

En este caso usamos MapListHandler en vez de BeanListHandler.
Notar la forma compacta de instanciar los objetos y ejecutar todo
En una sola línea.
Comparar con el código necesario si no se usa DbUtils

```
//Existen otros tipos de handlers para manejar otros tipos de valores como escalares, arrays o listas,
//y metodos de QueryRunner para sentencias sql de actualizacion
//Ver mas documentacion:
//http://commons.apache.org/proper/commons-dbutils/apidocs/index.html
//https://commons.apache.org/proper/commons-dbutils/examples.html
```

Al estar los datos leídos en un map,
esta sería la forma de manipularlos

J. Tuya, (2021-2022)

Tecnologías y Arquitectura

9

9



Lombok

Esta sería la forma de definir el objeto POJO.
Incluir los getters y setters para cada atributo.
Añadir constructor si se necesita instanciar
desde programa

Si se usa lombok se evita crear manualmente
los getters y setters
Basta con indicar una anotación a nivel de la clase
si se requiere constructor con todos los parametros,
incluir anotacion AllArgsConstructor

```
package gils.demo.jdbc;
public class Entity {
    private Integer id;
    private Integer id2;
    private String text;

    public Integer getId() { return this.id; }
    public Integer getId2() { return this.id2; }
    public String getText() { return this.text; }
    public void setId(Integer value) { this.id=value; }
    public void setId2(Integer value) { this.id2=value; }
    public void setText(String value) { this.text=value; }
}
```

```
package gils.demo.jdbc;
import lombok.*;
@Getter @Setter
public class Entity {
    private Integer id;
    private Integer id2;
}
```

IMPORTANTE:
Para ejecutar desde eclipse hay que
Poder instalarlo en el entorno.
<https://projectlombok.org/setup/eclipse>

Más información:
<https://www.sitepoint.com/declutter-pojos-with-lombok-tutorial/>
<http://www.baeldung.com/intro-to-project-lombok>

J. Tuya, (2021-2022)

Tecnologías y Arquitectura

10

10

Parte 2: Arquitectura de aplicación

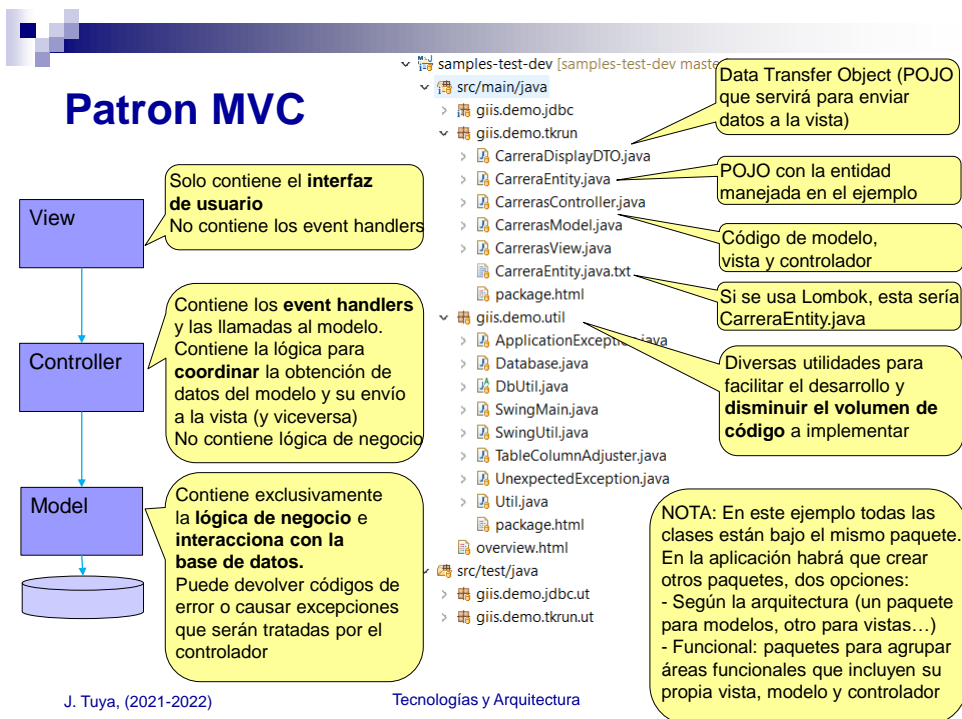
- **Objetivo:**
 - Facilitar la creación de la estructura inicial del proyecto
 - Reducir la cantidad del código escrito para implementar cada funcionalidad
 - Separar claramente el diálogo con el usuario de los procesos de negocio
- Los frameworks comunes (Spring Boot, ASP.Net) proporcionan:
 - Un patron de arquitectura basado en MVC
 - Un sistema de persistencia de datos (jpa, hibernate, EF)
 - Nota: en estos frameworks el modelo del que hablamos son una serie de niveles, p.e. en Spring Boot: Servicio, Repositorio y Entidad.
- Con las tecnologías “conocidas” hasta el momento (swing, bases de datos, sql)
 - Tendremos algo “análogo” (salvando las diferencias...)

J. Tuya, (2021-2022)

Tecnologías y Arquitectura

11

11

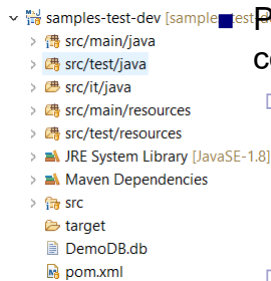


J. Tuya, (2021-2022)

Tecnologías y Arquitectura

12

Estructura proyecto Maven



Programa principal
(aplicaciones swing):
giis.demo.util.SwingMain

Ver javadoc del proyecto
para más detalles

La parte de pruebas
unitarias será tratada
más adelante

J. Tuya, (2021-2022)

Permite obtener todas las dependencias,
compilar y ejecutar pruebas...

□ Estructura

- pom.xml describe dependencias
- src/main/java código de aplicación
- src/test/java código de pruebas
- src/main/resources: otros archivos (p.e. configuración)

□ Desde Eclipse:

- Tener instalado el plugin M2Eclipse
- Asegur que esta configurado JDK: Desde build path, editar JRE System Library y en Environment comprobar que JavaSE-1.8 apunta a un JDK en vez de un JRE
- Maven->Update Project
- Run As->Maven install (ejecuta todo, incluyendo pruebas)

Tecnologías y Arquitectura

13

13

Notas sobre la aplicación y maven

- En otros frameworks, p.e. Spring Boot:
 - El modelo anterior es un conjunto de niveles servicio, repositorio, entidad
 - Reservan “modelo” para referirse a un modelo de la vista.
 - Pero **siempre se diferencia vista, controlador y el resto**
- Maven es el “estándar” para el build y test de un proyecto, aunque hay otros muy usados como gradle:
 - **Nunca incorporar binarios (ni la base de datos ni librerías)** al repositorio Git de la aplicación.
 - Para añadir una Liberia o driver, editar el pom.xml e incluir la dependencia (luego actualizar el proyecto en eclipse):

```
<dependency>
  <groupId>org.xerial</groupId>
  <artifactId>sqlite-jdbc</artifactId>
  <version>3.23.1</version>
</dependency>
```

J. Tuya, (2021-2022)

Tecnologías y Arquitectura

14

14

Funcionalidad del ejemplo tkrun Historias de usuario

- **Como usuario quiero visualizar las carreras en las que está abierta la inscripción.**

El usuario puede inscribirse a una carrera entre las fechas de inicio y fin establecidas y también posteriormente hasta el día de la carrera (inclusive). Visualizará el id y descripción de todas las carreras excluyendo las pasadas, con la indicación del estado Abierto en las que se puede realizar inscripción. Para ilustrar el manejo de listas, además de formato tabla, se mostrará la lista de carreras en un combo box. La fecha de referencia es la fecha actual, pero se simulará esta fecha mediante un campo de texto y un botón para actualizar la lista.

- **Como usuario quiero visualizar los datos detallados de la carrera seleccionada.**

Si la inscripción se realiza en las fechas de inscripción establecidas se aplica un descuento del 30%, si es después se aplica 0% y el día de la carrera un recargo del 50%. Cuando seleccione una carrera debe visualizar todos los detalles de ésta y el porcentaje de descuento o recargo aplicable en función de la fecha de hoy. Cuando cambie la fecha de hoy se actualizará la tabla con las carreras activas, manteniendo la selección previa los detalles correspondientes.

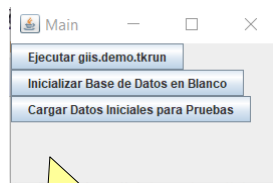
J. Tuya, (2021-2022)

Tecnologías y Arquitectura

15

15

Funcionalidad del ejemplo tkrun Interfaz de usuario



Para la aplicación desarrollada se aconseja tener una pantalla de lanzamiento similar a esta para facilitar la carga de datos o pruebas mientras se desarrollan historias de usuario

id	descr	estado
101	en fase 3	(Abierta)
102	en fase 2	(Abierta)
103	en fase 1	(Abierta)
104	antes inscripcion	

id	descr
101	en fase 3 (Abierta)

id	inicio	fin	fecha	descr
101	2016-10-05	2016-10-25	2016-11-10	en fase 3

J. Tuya, (2021-2022)

Tecnologías y Arquitectura

16

16

Modelo

```
package giis.demo.tkrun;

import java.util.*;
import giis.demo.util.Util;
import giis.demo.util.ApplicationException;
import giis.demo.util.Database;
* Acceso a los datos de carreras e inscripciones,
public class CarrerasModel {
    private Database db=new Database();

    public List<CarreraDisplayDTO> getListaCarreras(Date fechaInscripcion) {
        String sql="SELECT id,descr,"
            +" case when ?<inicio then "" //antes de inscripcion
            +" when ?<=fin then '(Abierta)'" //fase 1
            +" when ?<fecha then '(Abierta)'" //fase 2
            +" when ?=fecha then '(Abierta)'" //fase 3
            +" else "" //despues de fin carrera
            +" end as abierta"
            +" from carreras where fecha=? order by id";
        String d=Util.dateToIsoString(fechaInscripcion);
        return db.executeQueryPojo(CarreraDisplayDTO.class, sql, d, d, d, d, d);
    }
}
```

Utiliza esta clase de utilidad que contendrá toda la configuración de la base de datos y manejo de las conexiones

Internamente manejo todas las fechas como string ISO. La clase Util tiene algunas utilidades para conversiones

En una sola línea ejecuta la consulta obteniendo una lista de objetos CarreraDisplayDTO. Este DTO será el que recibirá el controlador para enviarlo a la vista. La clase Database, además de los datos de conexión contiene varias utilidades que utilizan Apache Commons DbUtils para facilitar estas tareas

J. Tuya, (2021-2022)

Tecnologías y Arquitectura

17

17

Modelo - Notas

- Cada función debe ejecutar una acción completa que pueda ser desarrollada y probada por separado.
- Debe incluir las validaciones “semánticas” necesarias (ver más adelante)
- Maximizar el uso de SQL (usar join, group by...), lo contrario penaliza severamente el rendimiento. Ejemplo:
 - Lo que **no se debe** hacer: Para mostrar datos de dos tablas: un SQL para obtener todos los ítems de la primera tabla, y para cada una, otro SQL para buscar más datos sobre cada ítem
 - Lo que **se debe** hacer: un único SQL con joins para obtener todos los ítems junto con sus datos.
 - Diferencia: Para n ítems:
 - En el primero ejecutamos n+1 consultas a la base de datos.
 - En el segundo ejecutamos 1 consulta a la base de datos
 - El factor que más penaliza es el número de veces que consultamos la base de datos, mucho más que el propio proceso del servidor de base de datos para obtener los datos

J. Tuya, (2021-2022)

Tecnologías y Arquitectura

18

18



```

package giis.demo.tkrun;

import javax.swing.JFrame;

/* Vista de la pantalla que muestra las carreras activas y permite interactuar con ellas. */
public class CarrerasView {

    private JFrame frame;
    private JTextField txtFechaHoy;
    private JButton btnTabCarreras;
    private JTable tabCarreras;
    private JComboBox<Object> lstCarreras;
    private JLabel descuento;
    private JTable tabDetalle;

    /**
     * Create the application.
     */
    public CarrerasView() {
        initialize();
    }

    /**
     * Initialize the contents of the frame.
     */
    private void initialize() {

        //Getters y Setters anyadidos para acceso desde el controlador (representacion compacta)
        public JFrame getFrame() { return this.frame; }
        public String getFechaHoy() { return this.txtFechaHoy.getText(); }
        public void setFechaHoy(String fechaIso) { this.txtFechaHoy.setText(fechaIso); }
        public JButton getBtnTabCarreras() { return this.btnTabCarreras; }
        public JTable getTablaCarreras() { return this.tabCarreras; }
        public JComboBox<Object> getListaCarreras() { return this.lstCarreras; }
        public void setDescuento(String descuento) { this.descuento.setText(descuento+"%"); }
        public void setDescuentoNoAplicable() { this.descuento.setText("N/A"); }
        public JTable getDetalleCarrera() { return this.tabDetalle; }
    }
}

```

Vista

La vista está creada en swing con Window Builder

Todo el código de inicialización (se ha ocultado aquí). Aunque Window Builder permite generar el código del controlador de eventos, No se incluye aquí ningún controlador de eventos (estos se crearán en la vista)

Otros métodos creados para permitir que el controlador acceda a objetos de la vista.

J. Tuya, (2021-2022)

Tecnologías y Arquitectura

19

19



```

package giis.demo.tkrun;

import java.awt.event.MouseAdapter;

/* Controlador para la funcionalidad de visualizacion de carreras para la inscripcion. */
public class CarrerasController {
    private CarrerasModel model;
    private CarrerasView view;
    private String lastSelectedKey=""; //recuerda la ultima fila seleccionada para restaurarla cuando cambie la t

    public CarrerasController(CarrerasModel m, CarrerasView v) {
        this.model = m;
        this.view = v;
        //no hay inicializ
        this.initView();
    }

    /* Inicializacion del controlador, añade los manejadores de eventos */
    public void initController() {
        //ActionListener define solo un metodo actionPerformed
        //view.getBtnTabCarreras().addActionListener(e -> getListaCarreras());
        //ademas invoco el metodo que responde al listener en el exceptionWrapper para que se encargue de las exce
        view.getBtnTabCarreras().addActionListener(e -> SwingUtil.exceptionWrapper(() -> getListaCarreras()));

        //En el caso
        //ver discus
        view.getTablaCarreras().addMouseListener(new MouseAdapter() {
            @Override
            public void mouseReleased(MouseEvent e) {
                //no usa mouseClicked porque al establecer seleccion simple en la tabla de carreras
                //el usuario podria arrastrar el raton por varias filas e interesa solo la ultima
                SwingUtil.exceptionWrapper(() -> updateDetail());
            }
        });
    }
}

```

Controlador

Es el único que conoce el modelo y la vista, creando estos objetos

El programa principal instanciará el controlador pasándoles los objetos de la vista y modelo

El programa principal invocará a initController, que instala los manejadores de eventos

Los eventos p.e. procedentes de botones se instalan directamente con una única expresión lambda

En este caso el controlador de evento se inserta en una utilidad wrapper, de forma que todas las excepciones son capturadas y mostradas en una ventana de diálogo

Para detectar la selección en una fila de tabla se realiza de esta forma, detectando el evento del ratón

J. Tuya, (2021-2022)

Tecnol

20

```

public void initView() {
    //Inicializa la fecha de hoy a un valor
    //y actualiza los datos de la vista
    view.setFechaHoy("2016-11-10");
    this.getListaCarreras();

    //Abre la ventana (sustituye al main generador)
    view.getFrame().setVisible(true);
}

/**
 * La obtencion de la lista de carreras solo
 * y usar metodo de SwingUtil para crear un
 */
public void getListaCarreras() {
    List<CarreraDisplayDTO> carreras=model.getListaCarreras(Util.isoStringToDate(view.getFechaHoy()));
    TableModel tmodel=SwingUtil.getTableModelFromPojos(carreras, new String[] {"id", "descr", "estado"});
    view.getTablaCarreras().setModel(tmodel);
    SwingUtil.autoAdjustColumns(view.getTablaCarreras());

    //Como se guarda la clave del
    this.restoreDetail();

    //A modo de demo, se muestra tambien la misma informacion en forma de lista en un combobox
    List<Object[]> carrerasList=model.getListaCarrerasArray(Util.isoStringToDate(view.getFechaHoy()));
    ComboBoxModel<Object> lmodel=SwingUtil.getComboModelFromList(carrerasList);
    view.getListaCarreras().setModel(lmodel);
}

```

Establece el estado inicial de la vista. (este método se invoca en la instanciación del controlador, dejando todo listo para que el usuario interactúe con la aplicación)

La funcionalidad pone una fecha para pruebas, ejecuta el método para mostrar la lista de carreras y hace visible el frame

Mostrar los valores en una tabla en el interfaz de usuario se reduce a obtener la lista de objetos del modelo y enviarlos al table model de la vista. Utiliza otro método de utilidad que convierte las columnas de la lista de objetos indicadas en columnas de un table model

Otra utilidad para formatear automáticamente la tabla

Esto forma parte de la segunda historia de usuario, en que se debe restaurar la información de detalle correspondiente a la fila seleccionada

J. Tuya, (2021-2022)

Tecnologías y Arquitectura

21

21

Controlador - Notas

- El controlador es el “pegamento” entre la vista y el modelo.
- Es el único que conoce vista y modelo, el modelo no conoce la vista, ni viceversa.
- Ejemplo interacción simple:
 - Usuario interactúa con la vista, cambiando datos y pulsando un botón
 - Controlador reconoce esta acción en el evento handler y ejecuta un método propio que coordinará el resto de acciones.
 - Recoje los datos de la vista, los convierte a los parámetros requeridos por el modelo e invoca a la función correspondiente del modelo
 - Recoje el resultado de la función del modelo, convierte al formato requerido por la vista (p.e. un table model) y actualiza esta
- Aprender a utilizar los métodos que se proporcionan en SwingUtil:
 - Simplifica mucho el código
 - Se pueden adaptar o añadir nuevas funciones si se necesita (coordinarse en el equipo)
- Donde se ponen las validaciones?

J. Tuya, (2021-2022)

Tecnologías y Arquitectura

22

22

Validaciones - Notas

- Los frameworks típicos (p.e. Spring Boot o ASP.NET) proporcionan mecanismos propios para la validación, a veces incluyendo validaciones simples etiquetando las entidades.
- En nuestro caso seguiremos este criterio:
 - Validaciones sintácticas: Propias de la naturaleza de un dato. Validar en el controlador
 - Validaciones semánticas: Propias del uso que un proceso de negocio realiza para un dato. Validar en el modelo
- Ejemplo: una función del modelo recibe dos fechas y un id numérico de un objeto de la base de datos:
 - En el controlador validar que las fechas sean sintácticamente correctas, el id un número...
 - En el modelo validar que las fechas formen un intervalo válido, que el id sea el correspondiente a un objeto existente en la base de datos...
- Para validaciones en el modelo basta con lanzar una excepción. Si hemos instalado el controlador de eventos con el wrapper, este se encargará de mostrar el mensaje al usuario.

J. Tuya, (2021-2022)

Tecnologías y Arquitectura

23

23

```
package giis.demo.util;
import java.io.FileInputStream;

/**
 * Hereda de DbUtil que contiene las utilidades para
 * realizar consultas con Apache Commons DbUtils
 * y scripts de base de datos para creación y
 * mantenimiento de la base de datos
 */
public class Database extends DbUtil {
    //Localizacion de ficheros de configuracion y carpetas de datos
    private static final String APP_PROPERTIES = "src/main/resources/application.properties";
    private static final String SQL_SCHEMA = "src/main/resources/schema.sql";
    private static final String SQL_LOAD = "src/main/resources/data.sql";
    //parametros de la base de datos leídos de application.properties (base de datos local sin usuario)
    private String driver;
    private String url;
    private static boolean databaseCreated=false;

    * Crea una instancia, leyendo los parametros de driver y url de application.properties
    public Database() {
        Properties prop=new Properties();
        try {
            prop.load(new FileInputStream(APP_PROPERTIES));
        } catch (IOException e) {
            throw new ApplicationException(e);
        }
        driver=prop.getProperty("datasource.driver");
        url=prop.getProperty("datasource.url");
        if (driver==null || url==null)
            throw new ApplicationException("Configuracion de driver y/o url no encontrada en applicat:");
        DbUtils.loadDriver(driver);
    }

    public String getUrl() {
        return url;
    }
}
J. Tuya, (2021-2022)
```

Base de Datos

Los scripts para crear la base de datos (schema.sql) y para cargar datos iniciales (data.sql) están en ficheros externos

Esta clase guarda los parámetros de conexión a la BD

El objeto Database es el que se instanciará cada vez que se realicen operaciones con BD en el modelo

Tecnologías y Arquitectura

24

24

Base de datos

Utilidades para crear y poblar la BD, que utilizan los archivos externos cuyo nombre se define en esta clase

```
/**
 * Creacion de una base de datos limpia a partir del script schema.sql en src/main/property
 * (si onlyOnce=true solo ejecutara el script la primera vez
 */
public void createDatabase(boolean onlyOnce) {
    //actua como singleton si onlyOnce=true: solo la primera vez que se instancia para me
    if (!databaseCreated || !onlyOnce) {
        executeScript(SQL_SCHEMA);
        databaseCreated=true; //NOSONAR
    }
}
/**
 * Carga de datos iniciales a partir del script data.sql en src/main/properties
 * (si onlyOnce=true solo ejecutara el script la primera vez
 */
public void loadDatabase() {
    executeScript(SQL_LOAD);
}
```

Base de datos - Notas

- Importante utilizar una instancia del objeto Database en todas las clases del modelo. De esta forma nos desentendemos del control de las conexiones.
- Definir schema.sql y data.sql, que permiten crear y poblar fácilmente la base de datos (y realizar esto interactivamente como se vio anteriormente)
- Si no se usa sqlite, configurar el driver en application.properties y añadir la dependencia en pom.xml
- Recordar seguir estrictamente los convenios de Java para la capitalización en las clases que representan las tablas de la BD



Descripción paquete util (javadoc)

<u>Database</u>	Encapsula los datos de acceso JDBC, lectura de la configuracion y scripts de base de datos para creacion y carga.
<u>DbUtil</u>	Metodos de utilidad para simplificar las queries realizadas en las clases que implementan la logica de negocio: Se implementa como una clase abstracta para que la clase derivada implemente los detalles relativos a la conexion y a la estructura de la base de datos a crear, y a la vez pueda usar los metodos que se definen aqui.
<u>SwingMain</u>	Punto de entrada principal que incluye botones para la ejecucion de las pantallas de las aplicaciones de ejemplo y acciones de inicializacion de la base de datos.
<u>SwingUtil</u>	Metodos de utilidad para interfaces de usuario con swing (poblar tablas a partir de un objeto POJO que ha sido obtenido desde la base de datos, manejo de excepciones para metodos del controlador, autoajuste de la dimension de columnas, etc)
<u>ApplicationException</u>	Excepcion producida por la aplicacion antes situaciones que no deberian ocurrir pero que son controladas y por tanto, la aplicacion se puede recuperar (validacion de datos, prerequisites que no se cumplen, etc)
<u>UnexpectedException</u>	Excepcion producida por la aplicacion antes situaciones incontroladas (excepciones al acceder a la base de datos o al utlizar metodos que declaran excepciones throwable, etc)