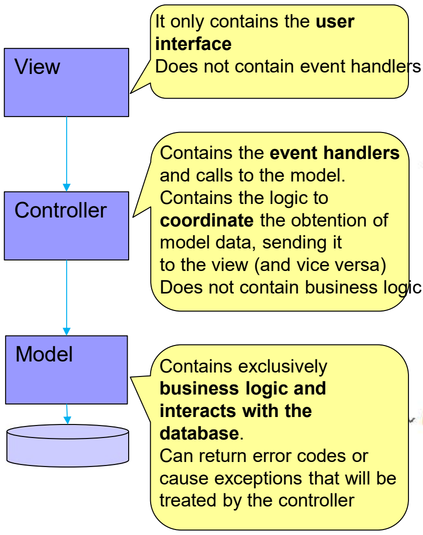
Part 2: Application architecture

- Objective:
  - □ Facilitate the creation of the initial project structure
  - □ Reduce the amount of code written to implement each functionality
  - □ Clearly separate the interaction with the user from the business processes
- Common frameworks (Spring Boot, ASP.Net) provide:
  - □ An architecture pattern based on MVC
  - □ A data persistence system (jpa, hibernate, EF)
  - □ Note: in these frameworks the model we are talking about is a series of levels, e.g. in Spring Boot: Service, Repository and Entity.
- With the "known" technologies so far (swing, databases, sql)
  - □ We will have something “similar" (saving distances...)

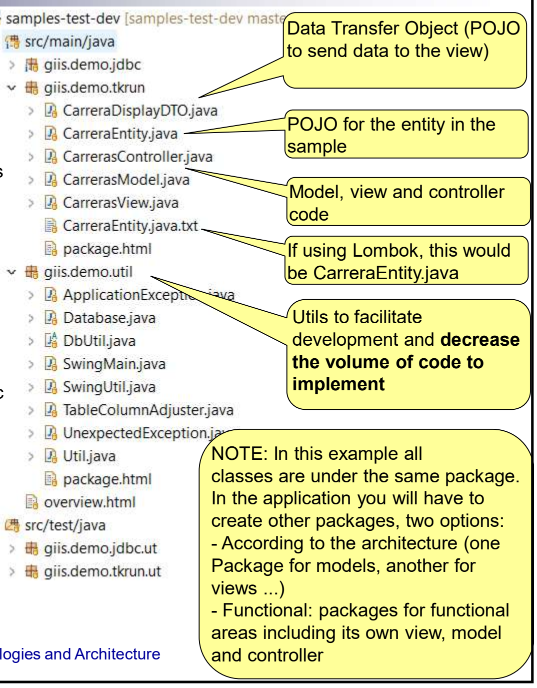
MVC pattern



It only contains the **user interface**
Does not contain event handlers

Contains the **event handlers** and calls to the model.
Contains the logic to **coordinate** the obtention of model data, sending it to the view (and vice versa)
Does not contain business logic

Contains exclusively **business logic and interacts with the database**.
Can return error codes or cause exceptions that will be treated by the controller



```
samples-test-dev [samples-test-dev maste
  src/main/java
    giis.demo.jdbc
    giis.demo.tkrun
      CarreraDisplayDTO.java
      CarreraEntity.java
      CarrerasController.java
      CarrerasModel.java
      CarrerasView.java
      CarreraEntity.java.txt
      package.html
    giis.demo.util
      ApplicationExcepti   java
      Database.java
      DbUtil.java
      SwingMain.java
      SwingUtil.java
      TableColumnAdjuster.java
      UnexpectedException.ja
      Util.java
      package.html
    overview.html
  src/test/java
    giis.demo.jdbc.ut
    giis.demo.tkrun.ut
```

Data Transfer Object (POJO to send data to the view)

POJO for the entity in the sample

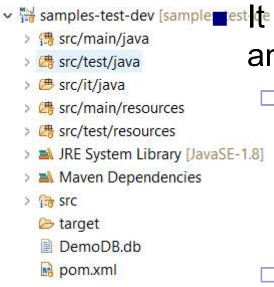
Model, view and controller code

If using Lombok, this would be CarreraEntity.java

Utils to facilitate development and **decrease the volume of code to implement**

NOTE: In this example all classes are under the same package. In the application you will have to create other packages, two options:
- According to the architecture (one Package for models, another for views ...)
- Functional: packages for functional areas including its own view, model and controller

Maven project structure



```
samples-test-dev [sample
  src/main/java
  src/test/java
  src/it/java
  src/main/resources
  src/test/resources
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  DemoDB.db
  pom.xml
```

- It allows to get all the dependencies, compile and run tests ...
  - □ Structure
    - pom.xml describes dependencies
    - src / main / java application code
    - src / test / java testing code
    - src / main / resources: other files (e.g. configuration)
  - □ From Eclipse:
    - Have the M2Eclipse plugin installed
    - Make sure that JDK is configured: From build path, edit JRE System Library and in Environment check that JavaSE-1.8 points to a JDK instead of a JRE
    - Maven-> Update Project
    - Run As-> Maven install (runs everything, including tests)

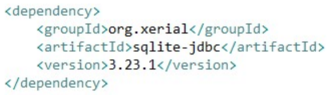
Main program
(swing applications):
giis.demo.util.SwingMain

See javadoc for more details

Unit testing will be treated later

Notes on the application and maven

- In other frameworks, e.g. Spring Boot:
  - □ The previous model is a set of levels: service, repository, entity
  - □ They reserve "model" to refer to a model of the view.
  - □ But they always differentiate view, controller and the rest
- Maven is the "standard" to build and test a project, although there are others widely used as gradle:
  - □ Never add binaries (neither the database nor libraries) to the application's Git repository.
  - □ To add a library or driver, edit the pom.xml and include the dependency (then update the project in eclipse):



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

Functionality of the tkrun sample User Stories

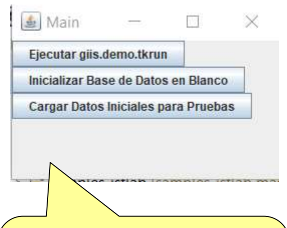
- **As a user I want to see the competitions in which registration is open.**

  The user can register for a competition between the established start and end dates and also later until the day of the competition (inclusive). It will display the id and description of all competitions excluding the past ones, with the indication of the “Open” status in which registration can be made. To illustrate the handling of lists, in addition to table format, the list of competitions will be displayed in a combo box. The reference date is the current date, but this date will be simulated using a text field and a button to update the list.

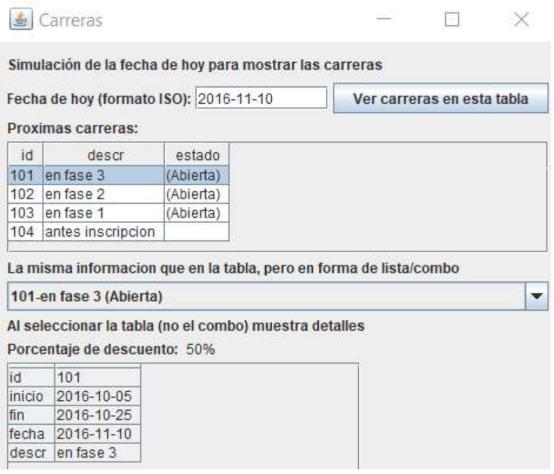
- **As a user I want to view the detailed data of the selected competition.**

  If the registration is done on the established registration dates, a 30% discount is applied, if it is afterwards, 0% is applied and on the day of the race a 50% surcharge applies. When you select a competition, you must see all the details of it and the percentage of discount or surcharge applicable based on today's date. When today's date changes, the table will be updated with the active competitions, keeping the previous selection with the corresponding details.

Functionality of tkrun User interface


Main
Ejecutar giis.demo.tkrun
Inicializar Base de Datos en Blanco
Cargar Datos Iniciales para Pruebas

For the developed application it is advisable to have a launch screen similar to this one to facilitate the loading of data or tests while developing user stories



Simulación de la fecha de hoy para mostrar las carreras

Fecha de hoy (formato ISO): 2016-11-10

Proximas carreras:

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

La misma informacion que en la tabla, pero en forma de lista/combo

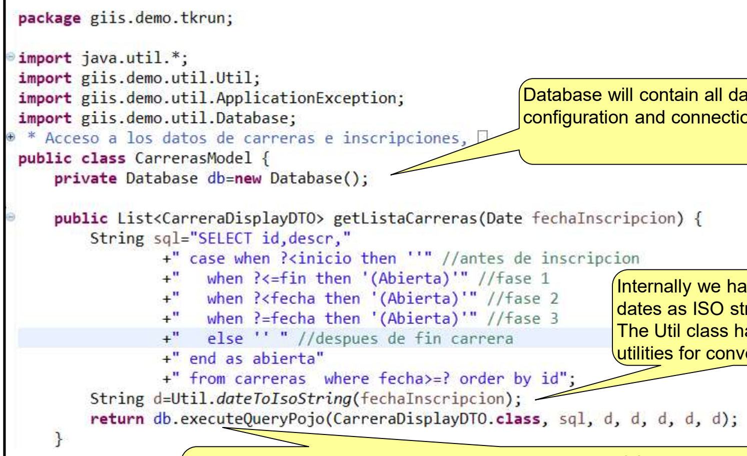
101-en fase 3 (Abierta)

Al seleccionar la tabla (no el combo) muestra detalles

Porcentaje de descuento: 50%

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

Model



```
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);
    }
```

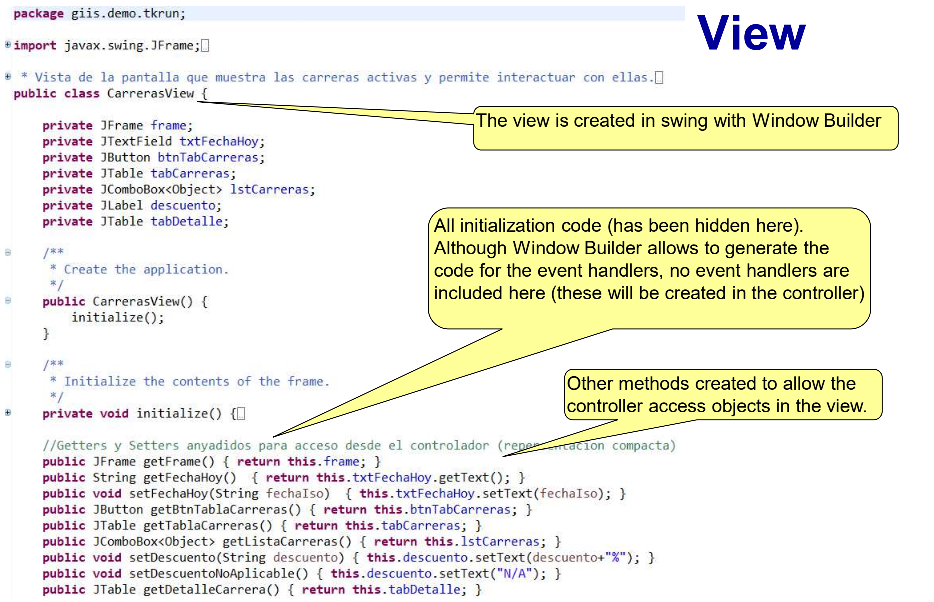
Database will contain all database configuration and connection management

Internally we handle all dates as ISO string.
The Util class has some utilities for conversions

In a single line it executes the query obtaining a list of CarreraDisplayDTO objects.
This DTO will be the one that the controller will receive to send to the view.
The Database class, in addition to the connection data, contains several utilities that use Apache Commons DbUtils to facilitate these tasks

Model - Notes

- Each function must execute a complete action that can be developed and tested separately.
- It must include the necessary "semantic" validations (see below)
- Maximize the use of SQL (use join, group by ...), otherwise performance is severely penalized. Example:
  - What not to do: To display data from two tables: one SQL to get all the items from the first table, and for each one, another SQL to find more data about each item
  - What to do: a single SQL with joins to get all the items together with their data.
  - Difference: For n items:
    - In the first one we execute n + 1 queries to the database.
    - In the second we execute 1 query to the database
  - **The factor that penalizes the most** is the number of times we access the database, much more than the process of the database server itself to obtain the data



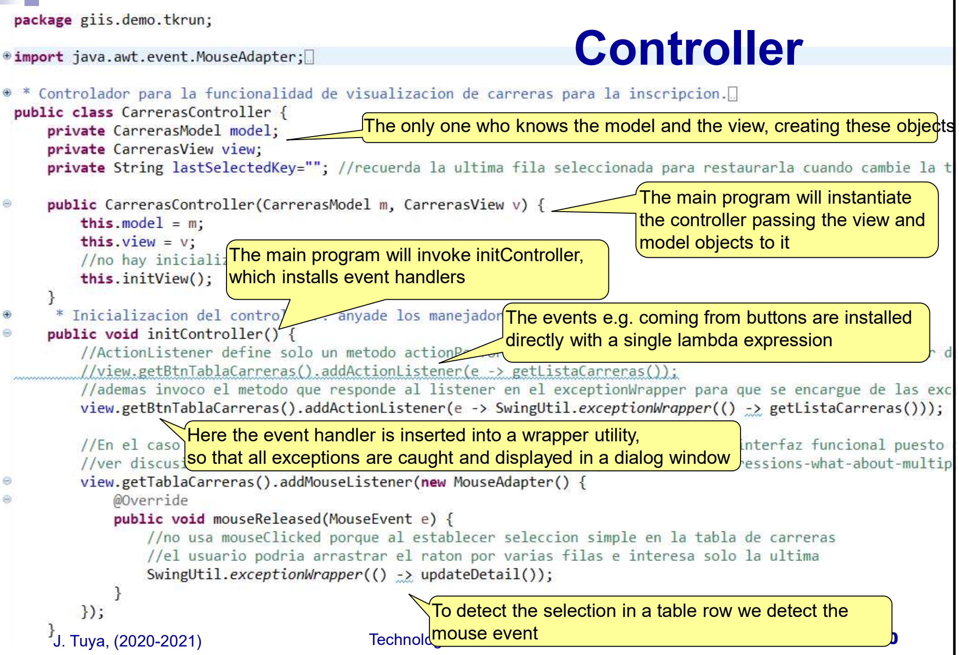
View

```
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 (repre      tacion 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 getBtnTablaCarreras() { 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; }
```

The view is created in swing with Window Builder

All initialization code (has been hidden here). Although Window Builder allows to generate the code for the event handlers, no event handlers are included here (these will be created in the controller)

Other methods created to allow the controller access objects in the view.



Controller

```
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 iniciali
        this.initView();
    }
     * Inicializacion del contro      anyade los manejador
    public void initController() {
        //ActionListener define solo un metodo actionP
        //view.getBtnTablaCarreras().addActionListener(e -> getListaCarreras());
        //ademas invoco el metodo que responde al listener en el exceptionWrapper para que se encargue de las exc
        view.getBtnTablaCarreras().addActionListener(e -> SwingUtil.exceptionWrapper(() -> getListaCarreras()));

        //En el caso                                                 interfaz funcional puesto
        //ver discus                                                 essions-what-about-multip
        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());
            }
        });
    }
```

The only one who knows the model and the view, creating these objects

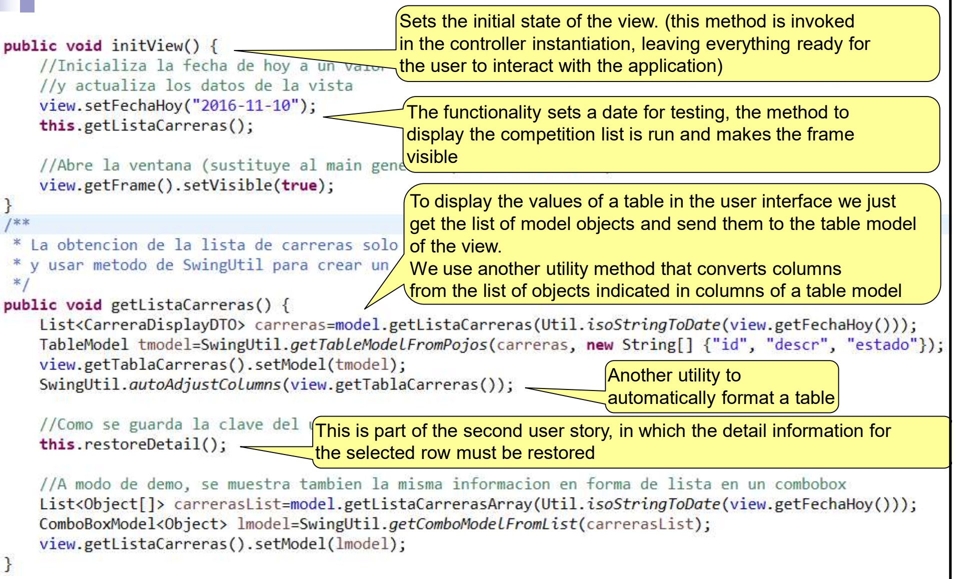
The main program will instantiate the controller passing the view and model objects to it

The main program will invoke initController, which installs event handlers

The events e.g. coming from buttons are installed directly with a single lambda expression

Here the event handler is inserted into a wrapper utility, so that all exceptions are caught and displayed in a dialog window

To detect the selection in a table row we detect the mouse event



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

    //Abre la ventana (sustituye al main gene
    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);
}
```

Sets the initial state of the view. (this method is invoked in the controller instantiation, leaving everything ready for the user to interact with the application)

The functionality sets a date for testing, the method to display the competition list is run and makes the frame visible

To display the values of a table in the user interface we just get the list of model objects and send them to the table model of the view.
We use another utility method that converts columns from the list of objects indicated in columns of a table model

Another utility to automatically format a table

This is part of the second user story, in which the detail information for the selected row must be restored

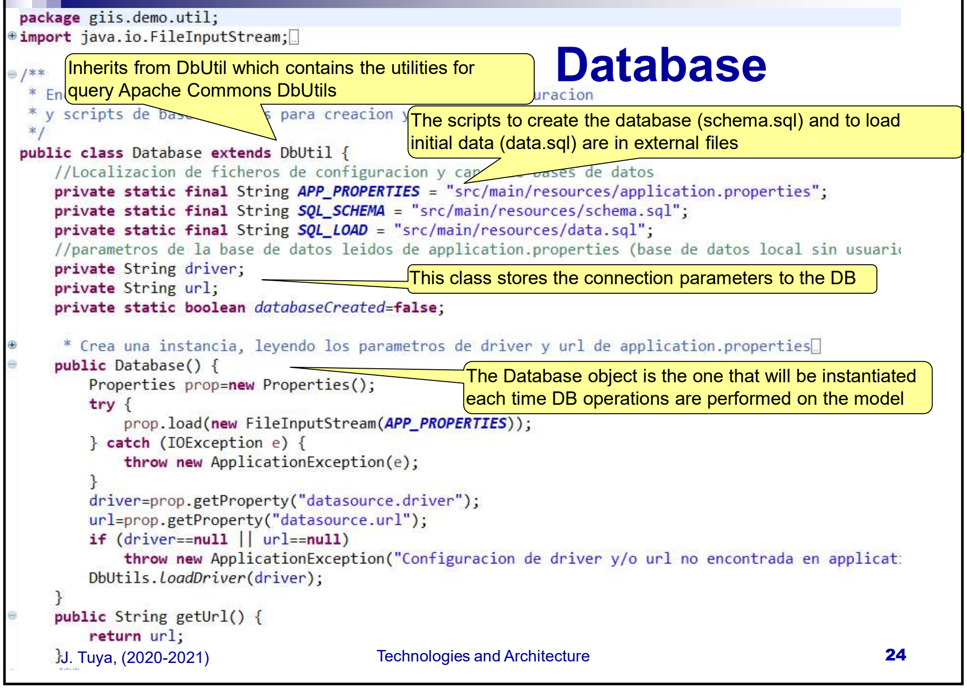
Controller - Notes

- The controller is the "glue" between the view and the model.
- It is the only one who knows view and model, the model does not know the view, or vice versa.
- Simple interaction example:
  - □ User interacts with the view, changing data and pressing a button
  - □ Controller recognizes this action in the handler event and executes its own method that will coordinate the rest of the actions.
    - Collect the data from the view, convert it to the parameters required by the model, and invoke the corresponding function of the model
    - Collect the result of the model function, convert to the format required by the view (e.g. a table model) and update this
- Learn how to use the methods provided in SwingUtil:
  - □ It simplifies the code a lot
  - □ New functions can be adapted or added if needed (coordinate within the team)
- Where do we put the validations?

Validation - Notes

- Typical frameworks (e.g. Spring Boot or ASP.NET) provide their own mechanisms for validation, sometimes including simple validations tagging entities.
- In our case we will follow this criterion:
  - □ Syntactic validations: Pertaining to the nature of a data. Validate in controller
  - □ Semantic validations: Pertaining to the use that a business process makes for a data. Validate in the model
- Example: a model function receives two dates and a numeric id from an object in the database:
  - □ In the controller we validate that the dates are syntactically correct, the id a number ...
  - □ In the model, we validate that the dates form a valid interval, that the id is the one corresponding to an existing object in the database...
- For validations in the model it is enough to throw an exception. If we have installed the event handler with the wrapper, it will be in charge of displaying the message to the user.

Database



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

/**
 * En                                                    uracion
 * y scripts de bas          s para creacion y
 */
public class Database extends DbUtil {
    //Localizacion de ficheros de configuracion y car        bases 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 leidos de application.properties (base de datos local sin usuari
    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;
    }
```

Inherits from DbUtil which contains the utilities for query Apache Commons DbUtils

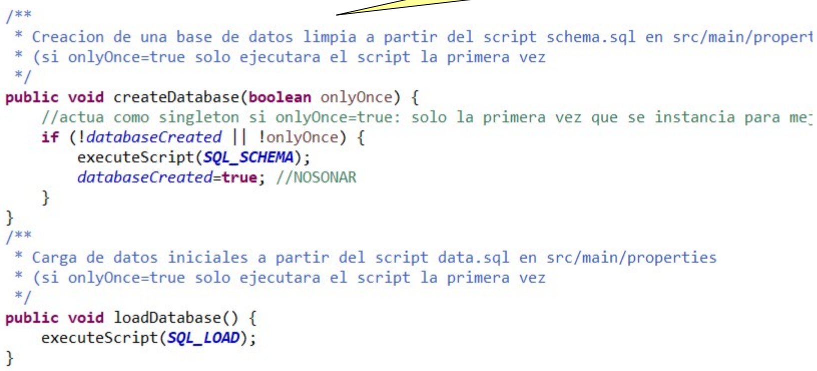
The scripts to create the database (schema.sql) and to load initial data (data.sql) are in external files

This class stores the connection parameters to the DB

The Database object is the one that will be instantiated each time DB operations are performed on the model

Database

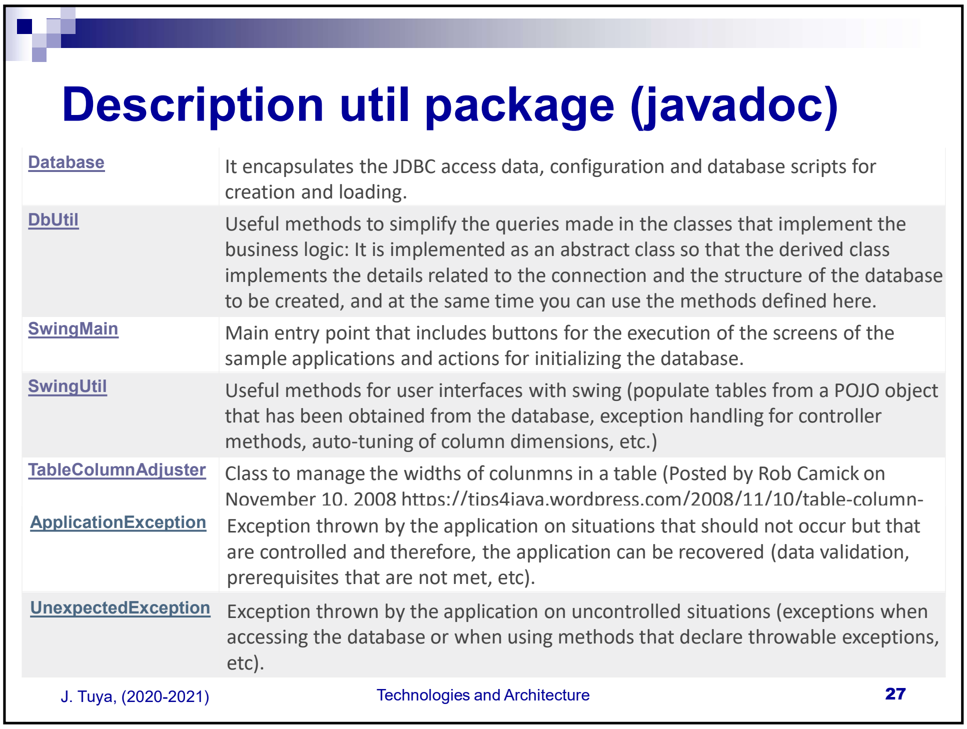
Utilities to create and populate the DB, used by external files whose name is defined in this class



```
/**
 * Creacion de una base de datos limpia a partir del script schema.sql en src/main/propert
 * (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);
}
```

Database - Notes

- It is important to use an instance of the Database object in all classes in the model. In this way we disengage from the control of the connections.
- Define schema.sql and data.sql, which allow you to easily create and populate the database (and do this interactively as seen above)
- If sqlite is not used, configure the driver in application.properties and add the dependency in pom.xml
- Remember to strictly follow the Java conventions for capitalization in the classes that represent the DB tables



Description util package (javadoc)

Database	It encapsulates the JDBC access data, configuration and database scripts for creation and loading.
DbUtil	Useful methods to simplify the queries made in the classes that implement the business logic: It is implemented as an abstract class so that the derived class implements the details related to the connection and the structure of the database to be created, and at the same time you can use the methods defined here.
SwingMain	Main entry point that includes buttons for the execution of the screens of the sample applications and actions for initializing the database.
SwingUtil	Useful methods for user interfaces with swing (populate tables from a POJO object that has been obtained from the database, exception handling for controller methods, auto-tuning of column dimensions, etc.)
TableColumnAdjuster	Class to manage the widths of colunmns in a table (Posted by Rob Camick on November 10, 2008 https://tips4java.wordpress.com/2008/11/10/table-column-
ApplicationException	Exception thrown by the application on situations that should not occur but that are controlled and therefore, the application can be recovered (data validation, prerequisites that are not met, etc).
UnexpectedException	Exception thrown by the application on uncontrolled situations (exceptions when accessing the database or when using methods that declare throwable exceptions, etc).