Course 2018-2019

Deliverable 1. IPC Laboratory

IPC

UPV - DSIC

Contents

1. Case Study

The Primary Care Clinic Advanced Medical Services wishes to obtain a new appointment management application that allows its reception staff to manage the appointments of the patients of the clinic. The new system will handle the data of the patients, of the doctors who attend the clinic, as well as the medical appointments that the patients arrange.

After carrying out an analysis of the requirements of the new system, these have been represented in a series of use scenarios that are described below (only the requirements that must be taken into account are described).

On the other hand, due to the limitation of the information management system to be used, operations to modify the information are not contemplated, so that it will only be possible to create or eliminate the records of people, doctors and medical appointments.

1.1. To add a patient

María, reception staff, attends to a patient at the clinic who is requesting an appointment. After checking that it is not present in the current list of patients, access the functionality that allows you to register it with all its data:

- Personal identity number (dni/nif/nie/passport)
- Name
- Surname
- Telephone
- Photography

María ask the patient if he wants to add his photo to its card. The patient aggress, thus María takes a picture of the patient with a small webcam and stores it on the hard disk of the computer where the dating system resides.

The system checks the information, adding the patient to the list of patients in the clinic, and then informing Maria that the process has been successfully completed. María checks that the entered data are correct, choosing the new patient from the list of clients, to obtain all their data and thus verify them with the patient.

1.2. To delete a patient

After receiving the call from a new client of the clinic, Sara, reception staff, had added the new patient José Carlos García to the list of patients at the clinic. However, after verifying his data, she realizes that she has made a mistake in his phone number, so she must remove it from the system and then add it again. Sara, selects the option to delete after selecting the patient from the patient list. The system **checks that the patient has no appointment assigned**. Unfortunately, Sara had already added the appointment that the client had requested, so the system informs Sara that **she cannot eliminate the patient** García.

Sara removes the appointment created and re-selects the client from the patient list, and re-accesses the option to delete patient. The system checks that **there is no recorded appointment** for the patient and removes him, informing Sara that **he has been removed** from the system. Sara can now finish solving her mistake by adding the patient again.

1.3. To show patient's data details

Dr. Ramirez needs to contact one of his patients in order to modify the medication guidelines that he had prescribed, since he had been wrong. For this he goes to reception where he finds Maria. The doctor gives Maria the name and surname of the patient. María, look for the patient in the patient list and after selecting it, clicks on

the corresponding option that allows you to know all the patient's personal data. María notes the phone on a poster and gives it to the doctor.

1.4. To add a doctor

The clinic has closed a new collaboration agreement with a new primary care doctor. The head of the clinic tells the new doctor to do the favour of going through reception to be added to the medical table. Pedro, receives it and takes her a picture with the computer's webcam, storing it on the hard drive. Subsequently, access to the functionality that allows you to add it to the medical table, asking the doctor the data needed by the system:

- Personal identity number (DNI/NIF/NIE/passport)
- Name
- Surname
- Telephone
- Photography (optional)
- Days in which the doctor visits
- Starting time of the medical visit(the start time is common to all the days)
- Ending time of the medical visit(the start time is common to all the days
- Examination room in which the doctor visits, selected among the possible ones in the clinic.

The system checks the information and adds to the new patient, informed Pedro that the new doctor has been added to the clinic's list of doctors. Pedro checks that he has no introduced any wrong field, selecting the doctor from the list of doctors and selecting the feature to show their detailed information.

1.5. To delete a doctor

After entering the data of a new doctor, Pedro realizes that he has made a mistake with his telephone number, so he has to delete his record. First, he selects the doctor from the list, and after that, he accesses to the option to eliminate the doctor. The system **verifies that the doctor does not have any concerted appointment**, and since he does not have any appointment, it deletes the record and informs Pedro that the doctor has been eliminated.

1.6. To show doctor's data details

María needs to know the email of the doctor Pedro Martinez to send him the results of a patient, so she accesses the option to see the list of doctors. After selecting the doctor from the list, she accesses to the option to show their details. After that, she can access all the doctor's information (identification number, name, surname, telephone number, photograph, days of the week in which the visits will take place, start and end times of the visit and the room where he will visit).

1.7. To add a medical appointment

Laura Soriano, patient of the clinic, calls by telephone to make an appointment. Maria receives the call, and asks Laura with which doctor she wants the appointment. Laura tells him that she wants to arrange an appointment with her doctor, Pedro Martínez. Maria has access to the option to create a medical appointment. After that, select Laura Soriano from the list of Customers, then Pedro Martínez from the list of Doctors and inform Laura of possible appointments, since she can see which days of the week she has appointments, from what time and until what hour. It is possible to make an appointment at fifteen minute intervals. Laura tells him what date and time is right for her, and María selects it and stores the appointment

1.8. To delete a medical appointment

Laura Soriano has come up with an unforeseen event that prevents her from attending her next appointment, so she decides to take it. Call the clinic, where Maria takes care of you. María accesses the list of appointments, and selects the appointment to be deleted, by pressing the corresponding option. The system checks that the **appointment has not yet happened**, so it deletes it and informs the user that it has been deleted.

1.9.To show a medical appointment

Juan Hernandez had confirmed a medical appointment in the clinic, but he does not remember the day or the doctor that has to attend him, so he decides to go to the clinic to have the paper appointment again. Go to reception where Pedro attends him. Pedro accesses the list of Juan's appointments, and selects the next scheduled one. Take out the information of the appointment, indicating to Juan on what date the appointment is, the time, the doctor and room where he should go.

2. Data persistence

The persistence of the information will be done through an XML file, using the java JAXB library. Specifically, you have been provided with an access library (clinicDBAccess.jar) capable of storing and retrieving all the information of the clinic: patients, doctors, visiting rooms and appointments. For the library to work properly, the XML file clinicDB.xml must be located on the user's home, which on Windows systems is located at:

```
C:\users\userName
```

Where `userName` is the user's nickname used to initiate the session in Windows.

If you work on the computers in the laboratory or access from the virtual desktop, the user's home is located directly at:

```
W:\
```

The file *clinicDB.xml* should not be manipulate by manually, but through the API that the *clinicDBAccess.jar* library provides. As a starting point, you are provided with a *clinicDB.xml* file that contains the names of several doctors, patients, appointments and all the visiting rooms of the clinic.

2.1.Inclusion of the library *clinicDBAccess.jar* in the project

First of all, download and save the file clinicDBAccess.jar, which contains the access library, to the folder of your project.

The projects created by Netbeans for FXML applications have a `Libraries` folder where you can add the external libraries that you want. To do this, just locate on that folder, and from the context menu (right mouse button), select the option `Add JAR / Folder`, as shown in Figure 1 .

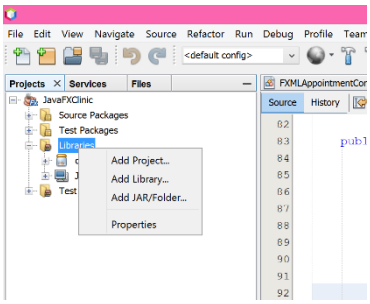

Figure 1. To select the adding Library option

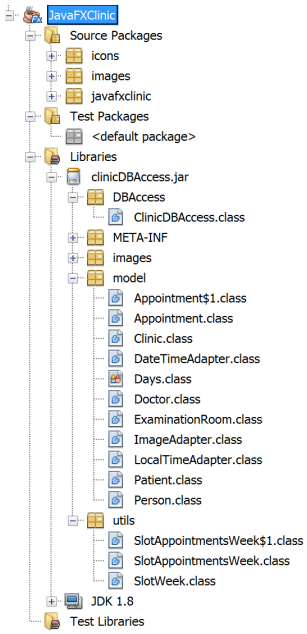

Figure 2. ClinicDBAccess.jar Library added to the project

After selecting the option, a dialog will appear where the jar file containing the library should be selected, which in this case should be stored in the project folder (where it had been downloaded). After accepting, the library is loaded, showing all available classes (Figure 2).

2.2. *clinicDBAccess.jar* API

As mentioned above, you should not directly manipulate the XML file that contains the information of the clinic, but access it through the API provided. Thus, the library provides an access class called ClinicDBAccess, which allows to load the information of the clinic from the file, as well as to store it in it. This class implements the Singleton pattern, thus only one object can be instantiated in the application.

Before getting the ClinicDBAccess, it is possible to access to all the data of the clinic through diverse public methods, which can be consulted in Table 1.

public ClinicDBAccess()	Reutrns a new ClinicDBAccess object if it has not been reviously created. If it was created, it returns the same object. When it creates the object, it checks if the XML file exists in the user's home and uploads the information of the clinic. If the file does not exist, it creates an empty one, so creating a clinic without information
public String getClinicName()	Returns the name of the Clinic
public void setClinicName(String newName)	It updates the name of the clinic by the given string
public ArrayList<Patient> getPatients()	Returns an ArrayList with all the patients of the clinic
public ArrayList<Patient> getPatients()	Returns an ArrayList with all the doctors of the clinic
public ArrayList<Appointment> getAppointments()	Returns an ArrayList with all appointments registered in the clinic
public ArrayList<Appointment> getPatientAppointments(String patientId)	Returns an ArrayList with all appointments registered in the clinic for the patient which identifier is equal to *patientId*
public ArrayList<Appointment> getDoctorAppointments(String doctorId)	Returns an ArrayList with all appointments registered in the clinic for the doctor which identifier is equal to *doctorId*
public boolean hasAppointments(Patient patient)	Returns TRUE if the patient has some appointment stored in the system.
public boolean hasAppointments(Doctor doctor)	Returns TRUE if the doctor has some appointment stored in the system.
public ArrayList<ExaminationRoom> getExaminationRooms()	Returns an ArrayList with all the examination rooms available at the clinic
public boolean saveDB()	It stores the current state of the data of the clinic in the XML file. Thus, it stores the doctors patients, appointments and examination rooms. It return TRUE if the data are stored, and FALSE on the contrary.

Table 1. API of the ClinicDBAcess class

2.3. Model classes

Clinic

The ClinicDBAccess class has as an attribute one object of the Clinic class that corresponds with the root node of the data persistence in the XML file. This API has not provided a direct way to get this object, but it offers some public methods (Table 1) to get needed data about the lists which are handled by the clinic object. Concretely, Cinic manage the following attributes:

- **doctors**: ArrayList with the doctors of the clinic.
- **patientes**: ArrayList with the patients of the clinic.
- **appointments**: ArrayList with the appointments of the clinic.
- **examinationRooms**: ArrayList with the examination rooms of the clinic.

```
@XmlRootElement(name = "clinic")
public class Clinic {
    private ArrayList<Doctor> doctors = new ArrayList<>();
    private ArrayList<Patient> patients = new ArrayList<>();
    private ArrayList<ExaminationRoom> examinationRooms = new ArrayList<>();
    private ArrayList<Appointment> appointments = new ArrayList<>();
```

Person

This class contains the attributes shared between doctors and patients., in order to reuse source code. Thus, doctors and patiens share the following attributes, which are accessible by means of getter and setters methods:

- String **identifier**: person's identity number, as the DNI, NIE or Passport number.
- String **name**: person's name.
- String **surname:** person's surname
- String **telephon**: contact telephone number.
- Image **photo**: person's photograph

Doctor

It stores all the information about a clinic doctor, offering setter and getter methods to access the data that is managed. In this case, we have all the attributes of the Person class plus:

- ArrayList<Days> **visitDays**: list with the days of the week in which this doctor visits. Days is an enumerated that can have the following values:{Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday}
- ExaminationRoom **examinationRoom**: examination room in which the doctor visits.
- LocalTime **visitStartTime**: starting hour and minutes of the visit (common for every day in which the doctor visits), for example 9:15. The valid minutes are: 00, 15, 30, and 45.
- LocalTime **visitEndTime**: ending hour and minutes of the visit (common for every day in which the doctor visits), for example 10:30. The valid minutes are: 00, 15, 30, and 45.

Patient

It stores all the information about a patient in the clinic, offering setter and getter methods to access the data that is managed. The data available are the same as those explained for the Person class.

Appointment

Corresponds to the data of a medical appointment, being possible to access its information from the corresponding setter and getter methods. Specifically, they are available:

- LocalDateTime **appointmentDateTime**: date and time of the medical appointment.
- Doctor **doctor**: doctor assigned to the medical appointment.
- Patient **patient**: patient attending the medical appointment.

The visiting room where the appointment is made can be known from the doctor of the appointment.

In addition, the class has two public methods that allow knowing which day of the week and which week of the year corresponds to the appointment (Table 2).

Table 2. Public methods of Appointment to manage the time

public Days getAppointmentDay()	Returns the day of the week to which the appointment corresponds, that is, one of the following values: Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday
public int getAppointmentWeek()	Returns the week number to which the appointment corresponds.

ExaminationRoom

Medical visit room. It is possible to know and establish the following attributes through the getter and setter methods:

- int **identNumber**: number of the examination room.
- String **equipmentDescription**: description of the equipment of the room.

The classes DateTimeAdapter, ImageAdapter and LocalTimeAdapter are used internally by the library to store and retrieve the attributes of the DateTime, Image and LocalTime types, respectively, from the XML file, so it is not necessary to use them directly.

2.4. Classes of Utils

SlotWeek

For a specific time, it indicates the availability of assigning a new appointment to a doctor throughout the week. It is possible to access the following attributes through its corresponding getters and setters:

- LocalTime **slot**: time which corresponds to the slot(for example 10:15)
- String **mondayAvailability**: availability on Monday.
- String **tuesdayAvailability**: availability on Tuesday.
- String **wednesdayAvailability**: availability on Wednesday.
- String **thursdayAvailability**: availability on Thursday
- String **fridayAvailability**: availability on Friday
- String **saturdayAvailability**: availability on Saturday.
- String **sundayAvailability**: availability on Sunday.

SlotAppointmentsWeek

This class offers a static method that returns an ArrayList with a doctor's availability slots for a specific week of the year, based on the appointment information of that method. Specifically, the header of the method is:

```
public static ArrayList<SlotWeek> getAppointmentsWeek(int week, ArrayList<Days> visitDays, LocalTime visitStartTime, LocalTime visitEndTime, ArrayList<Appointment> appointments)
```

where **week** it is the week of the year from which the availability will be obtained; **visitDays**, the days of the week in which the doctor attends; **visitStartTime**, the time when the visit starts; **visitEndTime**, time the visit ends; **appointments**, list all appointments recorded for the doctor in the system.

The method checks the list of appointments, if any appointment corresponds to the requested week, the patient's identifier is added in the slot corresponding to the indicated time and day. If there is a combination of time and day available to add a new appointment, it write in the corresponding field "Free". However, if the doctor does not visit on that day and time, it returns the "Not Available" value. In the following example, it is shown how this method would fill an element of the corresponding ArrayList at 10:30 in the morning, for a doctor who visits on Tuesday and Thursday, from 10:30 to 12:30, and who has a visit already registered in the week on Thursday at 10:30:

- Slot: 10:30

- mondayAvailability: "Not Available"
- tuesdayAvailability: “Free”
- wednesdayAvailability: "Not Available"
- thursdayAvailability: “4567833D”
- fridayAvailability: "Not Available"
- saturdayAvailability: "Not Available"
- sundayAvailability: "Not Available"

This class can be used, if desired, to view the weekly availability of the doctors.

2.5. Using the ClinicDBAccess library from the project

Access to the library

To access the methods of the library, it is necessary to first instantiate an object of the ClinicDBAccess class, using the static method *getSingletonClinicDBAccess*(). Afterwards, the registered information can be obtained or modified through the provided API. For example, in the following code a new patient is added to the list of patients of the clinic, storing the change in the database:

```
ClinicDBAccess clinicDBAccess = ClinicDBAccess.getSingletonClinicDBAccess();
clinicDBAccess.setClinicName("IPC Medical Services Clinic");
String url = System.getProperty("user.dir")+File.separator+ "src"+ File.separator+ "images"+ File.separator+
                    "men2.PNG";
Image avatar = new Image(new FileInputStream(url));
Patient patient= new Patient("5307867J","Juan","Cafe Grandes","9376543", "juan@cafe.upv.es",avatar);
clinicDBAccess.getPatients().add(patient);
clinicDBAccess.saveDB()
```

Use with graphic interface components

Ya se ha comentado que es posible obtener desde un objeto de tipo ClinicDBAccess a ArrayLists con los datos de los médicos, pacientes, citas médicas y salas de visita. Todas estas listas pueden conectarse a los elementos gráficos de la interfaz a través de envolverlas en listas observables **`ObservableList`**, de forma que las nuevas inserciones y eliminaciones de elementos se transmitan a los ArrayLists otiginales del objeto ClinicDBAccess. **En ningún caso** debe utilizarse una **`ObservableArrayList`,** puesto que los cambios no se reflejan en las listas originales. A modo de ejemplo, se muestra el código necesario para conectar la lista de pacientes de la clínica con una ListView:

It has already been commented that it is possible to obtain from an object of type ClinicDBAccess some ArrayLists with the data of doctors, patients, medical appointments and visiting rooms. All these lists can be connected to the graphical elements of the interface by wrapping them in **ObservableList** objects, so that the new inserts and deletions of elements are transmitted to the original ArrayLists of the ClinicDBAccess object. In no case should wrap in the original lists into an ObservableArrayList, since the changes are not reflected in the original lists. As an example, the code needed to connect a clinic's patient list with a ListView is shown:

```
ClinicDBAccess clinicDBAccess = ClinicDBAccess.getSingletonClinicDBAccess();
ObservableList<Patient> patientsObservableList;
// We wrap the patient's ArrayList from the clinic in an ObservableList
patientsObservableList = FXCollections.observableList(clinicDBAccess.getPatients());
// lPatients is a ListView<Patient>
lPatients.setItems(patientsObservableList); //The list is connected to the ListView
Patient patient= new Patient("5307867J","Juan","Cafe Grandes","9376543", "juan@cafe.upv.es",avatar);
patientsObservableList.add(patient); //A new patient is added
clinicDBAccess.saveDB(); //the change is stored in the XML file
```

Load images stored in the database in an *ImageView*

Doctors and patients have a photo stored in their file. To load it in an image View it is only necessary to recover this field from the corresponding object. For example, for a patient:

```
ClinicDBAccess clinicDBAccess = ClinicDBAccess.getSingletonClinicDBAccess();
int index = 0;
clinicDBAccess.getPatients().get(index);
//imagePhoto is a ImageView
imagePhoto.imageProperty().setValue(patients.get(index).getPhoto());
```

Persistence of changes

So that all the changes made by the program in the data are persisted, that is, they are stored in the XML file, it is necessary to call the saveDB () method. This method costs a lot of time, so it is advisable to add the call when the application is closed:

```
stage.setOnCloseRequest((WindowEvent event) ->{
      Alert alert = new Alert(AlertType.INFORMATION);
      alert.setTitle(clinicDBAccess.getClinicName());
      alert.setHeaderText("Saving data in DB");
      alert.setContentText("The application is saving the changes in the data into the database. This action
can expend some minutes.");
      alert.show();
      clinicDBAccess.saveDB();
});
```

3. Loading images from hard disk

Doctors and patients can store an image on their personal data sheet. There are several ways to load an image from the hard disk and display it in an ImageView:

1. The image is in a subdirectory of the src directory of the project, for example called images

```
String url = File.separator+"images"+File.separator+"woman.PNG";
Image avatar = new Image(new FileInputStream(url));
myImageView.imageProperty().setValue(avatar);
```

2. The image is anywhere on the hard drive and we have the complete path of it:

```
String url = “c:"+File.separator+"images"+File.separator+"woman.PNG";
Image avatar = new Image(new FileInputStream(url));
myImageView.imageProperty().setValue(avatar);
```

4. Handling dates and time

Attributes of type LocalDateTime and type DateTime must be managed in the application. Below we explain some useful methods.

Obtaining the week of the year to which a date belongs

The following code gets the week to which it belongs today, as well as the number of the day of the week (1 for Monday, 2 for Tuesday, etc.)

```
WeekFields weekFields = WeekFields.of(Locale.getDefault());
int currentWeek = LocalDate.now().get(weekFields.weekOfWeekBasedYear());
int numDayNow=LocalDate.now().get(weekFields.dayOfWeek());
```

Creating a date or a time field

onsult the LocalDate API and LocalTime to know all the methods it provides to create an object of your classes, but some that you may find useful are:

```
LocalDate sanJose = LocalDate.of(2018, 3,19);
LocalTime mascleta = LocalTime.of(14,0);
```

Udating a date

It is possible to increase or decrease days, months, years, minutes, hours, etc. to the fields of type LocalTime or LocalDate, or LocalDateTime using its own API. For example, the following code increases the current date by seven days, saving the result in a new variable. It also increases the current date by one month.

```
LocalDateTime nextWeekDay= LocalDateTime.now().plusDays(7);
LocalDateTime nextMontDay = LocalDateTime.now().plusMonths(1);
```

5. Delivery instructions

The project must implement the scenarios of uses proposed in the Case Study point. A usable interface must be designed, which allows to perform the functionalities reflected in the different use cases.

Due to the limitations of the XML format, no modification functionality is contemplated, and it is necessary to delete and add a new object when you wish to modify any of their data. In addition, doctors or patients with pending appointments cannot be eliminated. Nor can an appointment that is located in the past be deleted, since it is assumed that it was already made.

With regard to delivery:

- Export the Netbeans project to a zip file (File > Export Project> To Zip).
- Only one member of the group uploads the zip file to the corresponding task and ,in the comments field, the names of the group members should be typed in
- The delivery date for all groups is April 13, 2018.

6. Evaluation

- Those projects that do not compile or that do not show the main screen when starting will be scored with a zero
- Confirmation dialogs, errors, etc. should be included when it was considered necessary.
- To evaluate the design of the application interface, the guidelines studied in theory class will be taken into consideration.
- It should be possible to resize the main screen, whose controls will be adjusted appropriately to use the available space (use the containers seen in class (VBox, HBox, BordePane, GridPane, etc.).