LABORATORY CASE

GreenBall Tennis Club

Interfaces
Persona
Computador

IPC – DSIC

UPV

Curso 2022-2023

Table of content

1. Case Study

The tennis club *GreenBall* would like to offer a new application for its members to manage the tennis courts booking. The application must render well on any device and ensure usability and satisfaction.

The club has 6 courts; members can book any court in slots of one hour, from 9:00AM until and including 10:00PM (the facilities are open until 10:45PM).

Only users that have signed up for this service and have provided their contact details can book a court. The club intends to provide an easy and intuitive booking process, unnecessary information exchange must be avoided for the user to book a court rapidly. In addition, considering each user has different preferences, the system should display the availability for each court in the club.

On the other hand, users often forget which court was booked; sometimes because it was not booked by themselves, but by a colleague. For that reason, the club would like the application also to display who booked the court using a nick name, so their privacy is protected.

The booking can be cancelled only with a minimum of 24-hour notice. This rule aims to avoid the court remains unused because of a last-minute cancelation.

To avoid overbooking, a single user can book a maximum of 2 consecutive hours for the same court.

The following sections describe the use scenarios obtained after the system requirements analysis. These scenarios must be considered to design and implement the application accordingly.

1.1. User sign-up

Alberto has realized that it is possible to book tennis courts at *Greenball* using a free application at any time. He usually forgets to call during the office hours, so he is fascinated with the idea and decides to sign up. After downloading it, he accesses the SIGN-UP option. From that option, Alberto opens a form where he can enter his contact details:

- Name
- Family name
- Phone number
- Nickname (it is mandatory to sign in and might be displayed to the other users. It cannot contain spaces. It cannot be already in use)
- Password (any combination of, at least, 6 characters, either letters or numbers).
- Credit card number (16 digits) and card security code CSC (3 digits) (optional)
- Profile picture (optional)

After entering all contact details and choosing a tennis racquet as profile picture, the system is checking that the information is correct and no field contains data incorrect data format. All the information entered by Alberto is correct, the system displays a message indicating he has been successfully signed up and he should log in to start booking tennis courts.

1.2. Login

Pilar has just talked to his fried José and they have decided to book a tennis court for Thursday afternoon. Pilar accesses the *GreenBall* application so she can check whether there is any court available. The first thing she can find after launching the application is the login form, she enters her nickname and password. After pressing the ENTER button, the application checks whether the user is registered in the system.

Pilar made a mistake when entering the password, the application cannot match the data and displays a message to the user. Pilar is advised to try again, and she enters the nickname and password again. This time the data is correct, the system authenticates and allows the user to access the rest of functionalities.

1.3. Book a tennis court

Mario would like to book a tennis court. After login, he enters the "Book a tennis court" option. Mario can see all the courts that are available and booked for today. However, he would like to make his reservation for tomorrow, so he presses on the menu to change the date. After choosing the next day, he realized that court number 3 is free at 6PM, so he chooses it and enters the "Book now" option.

After he completes the reservation, the system shows that court number 3 is booked from 6PM to 7PM. Moreover, the nickname of Mario is also displayed together with the reservation. Because Mario entered his credit card number when signing up, the cost of this service is directly charged 1. The application registers the transaction as already paid.

1.4. Look up my bookings

Miguel has a reservation this afternoon at 3PM, but he does not remember the court number. As he foresees that they will be in a hurry, he prefers to check it out before going and avoid unnecessary questions at the club. He launches the application and after login, he goes into *My Bookings*. The next view list the last 10 courts booked by him; he can easily check the most recent ones. For each reservation he can check the date, the court, and the booking start/end time, together with a transaction status (whether it service has been already paid or not).

1.5. Cancel a booking

Marisa had a court booked for Thursday afternoon, she wanted to play with her friend at the *GreenBall* club. However, today is Tuesday and her son told her that he will be playing football at that time, and she must bring him. Although she is slightly bothered, she calls her friends and asks them to cancel the plan.

In the last moment, Marisa launches the application and cancels the reservation, so she does not need to pay for it because she is still within the timeframe allowed by the club. After login, she accesses "My reservations". First, she can see the reservation that planned, so she selects it and presses on "Delete".

The system checks that the reservation date is later than today plus 24 hours, so it can be cancelled with no cost associated. The tennis court is then available for that slot.

If the reservation was made for a date that is already in the past, it cannot be cancelled because it means that the court was already used.

1.6. Look up a tennis court availability

Juan and Marcos planned to meet their mates to play tennis; they are supposed to book the tennis court for today. Juan and Marcos will meet to go together, but neither of them can recall which court was booked. In a moment, Marcos launches the application and checks the availability of the tennis courts for today. This feature does not require authentication. In just a few seconds, he can filter by time slot and figure out that "junior33", his opponent has booked court number 2 at 5 PM.

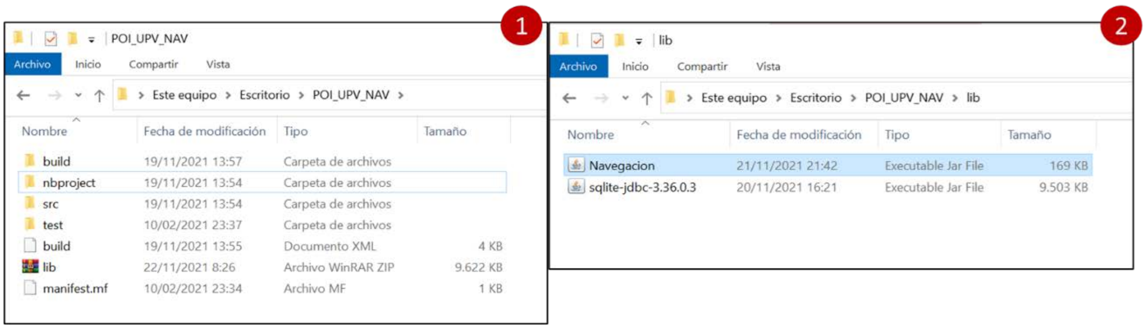
1.7. Update profile data

Juan would like to update his profile data. After login, he accesses a specific option to do that. He can update any information for his profile, except his nickname that is blocked. After he performs the changes, he presses on SAVE and the system checks that all new values are aligned with the system requirements, so the application updates the data and says goodbye to Juan.

2. Data model

The data persistence should be implemented using an SQLite[1] database. This is completely transparent for the developer. The access library *tenisClub.jar* is provided. It enables you to store and retrieve all the information that needs to be persisted. Also, it defines the data model required. To get the library working properly, it is mandatory to include in the project the sqlite-jdbc-3.41.2.1.jar. Both files are available for download at Poliformat.

It is recommended that these libraries get unzipped in a new folder "lib" inside the NetBeans project directory. See Figure I.


1
POI_UPV_NAV
Archivo
Inicio
Compartir
Vista
Este equipo > Escritorio > POI_UPV_NAV >
2
lib
Archivo
Inicio
Compartir
Vista
Este equipo > Escritorio > POI_UPV_NAV > lib

Nombre	Fecha de modificación	Tipo	Tamaño
build	19/11/2021 13:57	Carpeta de archivos	
nbproject	19/11/2021 13:54	Carpeta de archivos	
src	19/11/2021 13:54	Carpeta de archivos	
test	10/02/2021 23:37	Carpeta de archivos	
build	19/11/2021 13:55	Documento XML	4 KB
lib	22/11/2021 8:26	Archivo WinRAR ZIP	9.622 KB
manifest.mf	10/02/2021 23:34	Archivo MF	1 KB

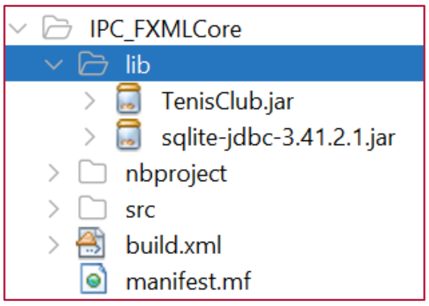
Nombre	Fecha de modificación	Tipo	Tamaño
Navegacion	21/11/2021 21:42	Executable Jar File	169 KB
sqlite-jdbc-3.36.0.3	20/11/2021 16:21	Executable Jar File	9.503 KB

Figure I. Unzip the library in the project folder

[1] https://www.sqlite.org/index.html

The projects created by NetBeans for Java applications brings in a `Libraries` package that we can use to add third-party libraries that are necessary for the project. We can do right-click on the `Libraries` package and from the context menu click on `Add JAR/Folder`, see Figure 2.

In the Add Jar/Folder dialog, select the lib folder we created earlier. Then, select both .jar files and click on the Open button. NetBeans will add the libraries to the project and as you can see in Figure 3.



```
IPC_FXMLCore
  lib
    TenisClub.jar
    sqlite-jdbc-3.41.2.1.jar
  nbproject
  src
  build.xml
  manifest.mf
```

Figure 2. Lib folder

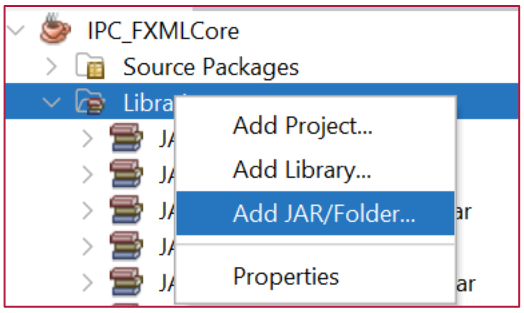
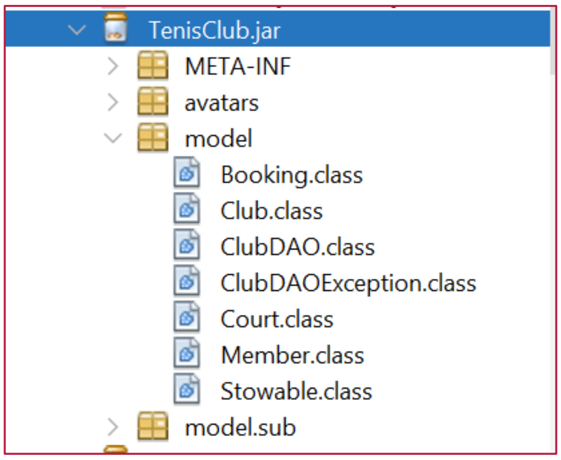


Figure 1 Add JAR/Folder option



```
TenisClub.jar
  META-INF
  avatars
  model
    Booking.class
    Club.class
    ClubDAO.class
    ClubDAOException.class
    Court.class
    Member.class
    Stowable.class
  model.sub
```

Figure 4 Classes within the jar TenisClub

2.1. Java classes

The data model defines multiple classes. The ones that you will need to develop the application are **Club**, **Court**, **Member** and **Booking**

Club

The class stores all system information and includes the following attributes:

- String **name**: name of the club
- int **bookingDuration**: the booking slot time. It is 60 minutes by default as it is explained at the Case Study.
- int **bookingSlots**; maximum number of bookings per day and court. It is 11 by default, default as it is explained at the Case Study.
- **courts**: list of tennis courts in the facilities. It is 6 courts by default.

- **members**: map with the club members.
- **bookings**: list with all reservations made.

This java class implements the Singleton pattern; it means that only one object of this class can be created in the application at the same time. This is the reason why the constructor has been defined as private, and there is a static method in the class that only creates this object and returns it. This method is **getInstance().**

Apart from that, there are some other methods to modify the attributes of the class and add club members, as well as reservations coming from those club members. At section 2.2, the methods defined as part of this class are described in detailed.

Court

This class stores the information related to the tennis court. It does only contain as attribute name, that can be accessed using the setter and getter methods.

- String **name**: String that represents the name of the club.

By default, 6 tennis courts will be created for the club (court 1, court 2, …, court 6). No tennis court can be added or deleted; however, the class allows court names update.

Member

This class stores the information related to the members of the club. The attributes of the class are:

- String **name**: name of the member.
- String **surname**: family name of the member.
- String **telephone**: string that represents the phone number of the member.
- String **nickName**: username used to authenticate in the system.
- String **password**: password used to authenticate in the system.
- String **creditCard**: string made of the 16 digits of the member's credit card number. It might be empty if the user did not provide this information during the sign-up process.
- Int **svc**: integer of 3 digits that represents the verification code of the member's credit card.
- Image **image**: member's profile picture. If it is not used by the constructor (null), a default one gets automatically added.

Every attribute can be updated except **nickName**. To do so, the class should contain different *setters*.

The constructor of this class is not defined as public. To create a new member, it is necessary to invoke the Club class method, registerMember(). This method will create a new member and register it in the club. The method returns an object with type Member

Booking

This class stores the information related to a reservation and includes the following attributes:

- LocalDateTime **bookingDate**: date and time when the user made the reservation.
- LocalDate **madeForDay**: date of the reservation, in other words, the day the member would like to rent the tennis court.
- LocalTime **fromTime**: time of the reservation.
- Boolean **paid**: flag for a credit-card-paid reservation.
- Court **court**: tennis court of the reservation.
- Member **member**: the user that made the reservation.

The only field that could be modified in an object type Booking is **paid**; for that purpose, there is dedicated method: setPaid(Boolean b).

The constructor class is not defined as public; so, in order to create a reservation, it is necessary to invoke the method registerBooking(…) from Club class. This method will create the reservation and register it in the club. The method returns an object with type Booking.

If we want to cancel a reservation, it is necessary to invoke the method removeBooking(Booking b) from Club class. This method will delete the reservation; no specific consideration will be taken (see Cancel a booking section from this document).

2.2. The Club class

As was previously mentioned, this java class implements the Singleton pattern. Using the method **getInstance()** we can get the object with type Club.

This object enables access to all stored data using public methods. Also, the associated setter and getter methods have been already implemented to facilitate the development of the project. Table 1 describes the functionality for all these methods.

Table 1. API for Club class

public static Club getInstance()	Creates an object of the Club class, if it was not previously instantiated. If it has already been instantiated, it returns the same object. When it creates it for the first time, it checks if the DB exists and loads its information. If the DB does not exist, it creates one with all tables that are required as well as populate them with the default data.
public String getClubName()	Returns the club name.
public int getClubBookingDuration()	Returns the number of minutes for any reservation.
public int getClubBookingSlots()	Returns the number of times the same user can book a tennis court per day.
public ArrayList<Member> getMembers()	Returns an ArrayList with all members of the club. The ArrayList cannot be modified.
public ArrayList<Court> getCourts()	Returns an ArrayList with all the tennis courts in the club. The ArrayList cannot be modified.
public ArrayList<Booking> getBookings()	Returns an ArrayList with all the reservations. The ArrayList cannot be modified. The reservations are linked to a specific date and time. The ArrayList is returned following an order: the oldest reservations are first on the list.
public ArrayList<Booking> getUserBookings(String login)	Returns an ArrayList with **a copy** of every reservation made by a given user (login). If the

	ArrayList is updated, the original list **remains the same**.
public ArrayList<Booking> getCourtBookings(String courtName, LocalDate madeForDay)	Returns an ArrayList with a **copy** of every reservation made for a given court (courtName) and date (madeForDay). If the ArrayList is updated, the original list **remains the same.**
public ArrayList<Booking> getForDayBookings(LocalDate forDay)	Returns an ArrayList with a copy of every reservation made for a given Date (forDay). If the ArrayList is updated, the original list **remains the same.**
public boolean existsLogin(String login)	Returns True when there is already a logged user for a given username (login)
public Member getMemberByCredentials(String nickname, String password)	Returns the object Member for a given *nickname* and *password*. If there is no match found, it returns *null*.
public Court getCourt(String name)	Returns the object Court for a given name.
public boolean hasCreditCard(String nickname)	Returns True if the Credit Card information was provided by the given user (*nickname*) during sign-up. If either the user does not exist or the information was not provided, it returns False.
public Member registerMember(String name, String surname, String telephon, String login, String password, String creditCard, int svc, Image image)	Returns an object type Member that matches with the parameters (*name, surname, telephone, login, password, creditCart, svc, image*). The method also stores the data in the database and adds the user to the map of members.
public boolean registerBooking(Booking booking)	Adds the reservation (*booking)* to the list of club reservations. If the operation fails because an error is found at the database, it returns False.
public Boolean removeBooking(Booking booking)	Deletes the reservation (*booking*) from the list of reservations. If an error is found the method returns False.
public void setInitialData()	Deletes all data stored and creates a fresh start of the system (default values).
public void addSimpleData()	Creates initial data for a demo: 5 users (nickname = “user1”, password = “123456x”), random number of reservations for the following day after the method is invoked and the next five days.

3. Programming tips

3.1. Loading images from hard disk

The club members can store as part of their profiles an image. 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 project's src directory, 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 disk, and we have the complete path of the image:

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

3.2. Date and time management

The application should be able to manage attributes type *LocalDateTime, LocalDate* and *DateTim*. In this section, you can find some methods that might be useful.

Getting the week number for a given Date.

The following code gets the current week number, as well as the number of the day within 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

You should check out the *LocalDate* and *LocalTime* API to learn all methods that are available for creating an object of this class. Among them, these two might be useful:

```
LocalDate sanJose = LocalDate.of(2020, 3,19);
LocalTime mascleta = LocalTime.of(14,0);
```

Updating a Date

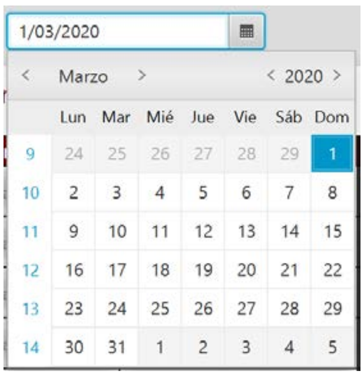
It is possible to increment or decrement a *LocalTime*, *LocalDate*, o *LocalDateTime* by days, months, years, minutes, hours, etc. using their own API. For example, the following code increments by 7 days the current date and updates a new variable with the result. Also, it increments by one month the current date, and updates another variable with the result. Finally, it increments the current time by 90 minutes, and updates a third variable, *endedTime*, with the result.

```
LocalDateTime nextWeekDay= LocalDateTime.now().plusDays(7);
LocalDateTime nextMontDay = LocalDateTime.now().plusMonths(1);
LocalTime endedTime = LocalTime.now().plusMinutes(90);
```

3.3. DatePicker configuration

The DatePicker components allow the user to select a date, being able to access the selected value through its valueProperty() property. It is possible to configure the way in which each day of the calendar is displayed by default, being possible to disable some days, change the background color, etc. The way this display is configured is like the way we use to configure a ListView or a TableView. The difference is that in this case we must extend the DateCell class and use the setDayCellFactory method. For example, the following code configures a DatePicker to be disabled the days before March 1, 2020 (see Figure 5).

```
dpBookingDay.setDayCellFactory((DatePicker picker) -> {
            return new DateCell() {
                @Override
                public void updateItem(LocalDate date, boolean empty) {
                    super.updateItem(date, empty);
                    LocalDate today = LocalDate.now();
                    setDisable(empty || date.compareTo(today) < 0 );
                }
            };
        });
```



	Lun	Mar	Mié	Jue	Vie	Sáb	Dom
9	24	25	26	27	28	29	1
10	2	3	4	5	6	7	8
11	9	10	11	12	13	14	15
12	16	17	18	19	20	21	22
13	23	24	25	26	27	28	29
14	30	31	1	2	3	4	5

Figura 5. DataPicker configured to disable days in the past

4. Delivery instruction

The project must be implemented taking into consideration every scenario from Case Study (page 2). The project should follow the principles and patterns that have been explained during the course lectures.

Please read carefully the following bullet-points before sending the final project delivery:

- Export the NetBeans Project to a zip file (option `File>Export Project> To ZIP`). As an alternative, you can zip the project folder. If you want to reduce the size of the file, the folder dist can be deleted from the ZIP file.
- Only one of the working group members should upload the zip file to Poliformat. Aa part of the process, he/she will fill out the comments text field with the name of every member of the group.
- The deadline for every group is the same one: **28th May 2023**.

5. Evaluation

- Projects that do not compile or do not show the main screen at startup will be scored with a zero.
- Confirmation dialogues, errors, etc. should be included as deemed 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 properly adjusted to use the available space (use the containers seen in class: *VBox*, *HBox*, *BorderPane*, *GridPane*, etc.).
- The UPV Academic Integrity Regulations and the ETSInf Academic Honesty Regulations will be applied. Anti-plagiarism tools are available in the course.