ENTREGA PRÁCTICAS

Club de Tenis GreenBall

Interfaces Persona Computador

IPC – DSIC

UPV

Curso 2022-2023

Índice

1. Caso de Estudio

El club de tenis *GreenBall* desea ofrecer una nueva aplicación que permita a sus socios gestionar el alquiler de las pistas. La aplicación se adaptará al tamaño del dispositivo donde se ejecute.

El club dispone de 6 pistas, ofreciendo su alquiler en tramos de una hora desde las 9 de la mañana hasta las 22:00h (las instalaciones cierran a las 22:45h, siendo los últimos 45 para uso de vestuarios).

Para poder reservar, los usuarios tendrán que haberse registrado previamente, indicando sus datos de contacto Desde el club desean que la reserva sea un proceso sencillo e intuitivo, de forma que el usuario pueda reservar rápidamente sin necesidad de aportar demasiada información. Además, como saben que los usuarios tienen preferencias por unas pistas u otras, el proceso deberá permitirles ver la disponibilidad de cada una de las pistas del club.

Por otra parte, muchos usuarios no recuerdan qué pista tenían reservada, a veces porque no la reservaron ellos directamente, sino algún compañero. Por ello, el club desea que la aplicación informe a los miembros de quién reservó cada pista mediante el nombre de usuario en el sistema (nickName), a fin de preservar su privacidad.

Finalmente, las reservas solo podrán anularse con 24h de antelación, para evitar que las pistas se queden sin utilizar a última hora.

Para evitar que un usuario acumule reservas, éste no podrá reservar más de dos horas consecutivas la misma pista.

Seguidamente se detallan los escenarios de uso que se han obtenido tras el análisis de requisitos del sistema. Estos se deben utilizarse para diseñar e implementar adecuadamente la aplicación requerida.

1.1. Registro de socio

Alberto ha visto que en el club *GreenBall* es posible reservar pistas en cualquier momento usando una aplicación gratuita. Como siempre se le olvida llamar en horas de oficina, le parece genial y decide registrarse en su aplicación. Tras descargarla, Alberto accede a la opción de registro. Desde la opción se abre un formulario donde Alberto puede introducir sus datos de contacto:

- Nombre
- Apellidos
- Teléfono
- Nickname (usado para acceder a la aplicación y para ser mostrado a los demás usuarios. No puede contener espacios. No puede repetirse en el club)
- Password (cualquier combinación de letras y números con más de 6 caracteres)
- Número de la tarjeta de crédito (16 números) y código de seguridad SVC (3 números) (opcional)
- Imagen de perfil (opcional)

Tras introducir todos sus datos de contacto y elegir una raqueta de tenis como imagen de perfil, el sistema comprueba que todos los datos son correctos y no se ha introducido ningún dato con un formato no permitido. Como todos los datos son correctos, informa

a Alberto de que ha sido añadido correctamente como usuario y que debe autenticarse para poder reservar pistas.

1.2. Autenticarse

Pilar acaba de hablar con su amigo José y han decidido reservar una pista para el jueves por la tarde, por lo que Pilar accede a la aplicación del club *GreenBall* para ver si hay pista disponible. Nada más abrir la aplicación, accede a la opción de autenticarse, donde aparece un formulario donde introduce su nickname y contraseña. Tras pulsar el botón de acceder, el sistema comprueba si el usuario está registrado en el sistema.

Pilar se ha confundido al introducir la contraseña, por lo que el sistema no lo encuentra y le informa que no está registrado en el sistema, permitiéndole introducir los datos de nuevo. Pilar vuelve a introducir sus datos, esta vez de forma correcta. El sistema la autentifica, permitiéndole el acceso al resto de funcionalidades.

1.3. Reservar una pista

Mario, quiere reservar una pista. Tras autenticarse, Mario accede a la opción de reservar pista. Mario puede ver todas las pistas disponibles y reservadas para hoy. Sin embargo, la reserva la quiere para mañana, así que pulsa en la opción para cambiar de día. Tras cambiar al día siguiente, observa que la pista 3 está libre a las 18:00, por lo que la selecciona y accede a la opción de reservar. Mario quiere aprovechar al máximo para jugar por lo que reserva las dos horas permitidas.

Tras reservarla, el sistema muestra la pista 3 como reservada desde las 18:00 hasta las 19:00 y desde las 19:00 a las 20:00. Además, se indica que es el dueño de las reservas mostrando su *nickname*. Como Mario indicó la tarjeta de crédito cuando se registró, el sistema de gestión cobra las reservas[1], por lo que desde la aplicación de reservas se guarda que las reservas ya han sido pagadas.

1.4. Ver mis reservas

Miguel tiene pista reservada para esta tarde a las 15:00, pero no recuerda exactamente el número de la pista. Como va a ir justito de tiempo, prefiere comprobarlo antes de ir y no tener que preguntar en el club al llegar. Accede a la aplicación del club *GreenBall*, y tras autentificarse, accede a la opción *Mis Reservas.* En la pantalla le salen las 10 últimas pistas que ha reservado, de forma que en primer lugar puede ver la más reciente. Para cada reserva puede consultar para qué día es, a qué pista corresponde, el horario de inicio de la reserva y el horario en el que tiene que abandonar la pista, así como si la reserva ha sido pagada.

1.5. Anular una reserva

Marisa tenía una pista reservada para el jueves por la tarde, para ir a jugar con sus amigas al club *GreenBall*. Sin embargo, hoy martes le ha dicho su hijo que le han puesto un partido de fútbol el jueves por la tarde, y que le tiene que llevar porque es en otro

[1] Se considera que la gestión y cobro de las reservas se realiza por una aplicación existente previamente.

colegio. Un poco molesta, Marisa avisa a sus compañeras de partida por teléfono que tienen que cancelar la partida.

Corriendo, Marisa accede a la aplicación del club para anular la reserva, de forma que no se la cobren porque aún está dentro del margen de 24h que le permite el club. Tras autentificarse, accede a la opción de sus reservas. En primer lugar, le aparece la reserva que tiene pendiente, así que la selecciona y pulsa en eliminar.

El sistema comprueba que la fecha de la reserva es posterior a la fecha actual por más de 24h, por lo que puede anular la reserva y poner la pista libre para ese horario.

No se podrá eliminar una reserva que esté situada en el pasado, puesto que se supone que ya se utilizó la instalación.

1.6. Ver disponibilidad de las pistas

Juan y Marcos han quedado para jugar con sus compañeros habituales de partida, quienes se encargaron de reservar la pista para hoy. Juan y Marcos han quedado para ir juntos, pero ninguno de los dos recuerda en qué pista estaba la reserva. En un momento, Marcos accede a la aplicación del club, donde hay una opción que no requiere autenticarse donde es posible ver la disponibilidad de las pistas para hoy. En un momento, busca en el tramo de la tarde el *nickname* de su contrincante "junior33", viendo que les toca la pista 2, a las 17:00.

1.7. Actualizar datos del socio

Juan quiere actualizar los datos de su perfil, después de autenticarse accede a la opción que le permite actualizar los datos de su perfil. Después de hacer los cambios que considera en cualquiera de los campos excepto en su nickName que no se puede modificar, indica al sistema que quiere guardar los cambios introducidos. Tras comprobar que los nuevos valores cumplen con los requisitos necesarios el sistema guarda la información y se despide de Juan.

2. Modelo de datos

La persistencia de los datos se realizará a través de una base de datos SQLite. El proceso es transparente para el programador. Para ello proporcionamos el fichero tenisClub.jar (librería) que contiene las clases del modelo que se deben de utilizar para realizar la entrega. Para que la librería funcione de manera adecuada es necesario incluir también en el proyecto la librería sqlite-jdbc-3.41.2.1.jar. Tienes disponibles ambos ficheros en poliformat.

Te recomendamos que crees la carpeta "lib" dentro de la carpeta donde se encuentra tu proyecto de netbeans para la entrega y después copia estos dos ficheros en dicha carpeta, Figura 1.

Los proyectos creados por Netbeans para aplicaciones Java tienen una carpeta `Libraries` donde añadir las librerías externas que son necesarias en el proyecto. Par añadir una librería basta con situarse sobre esa carpeta `Libraries`, y desde el menú contextual (botón

derecho del ratón), seleccionar la opción `Add JAR/Folder`, tal y como se muestra en la Figura 2.

Tras seleccionar la opción, aparecerá un diálogo donde se deben seleccionar los dos ficheros jar que hemos indicado y que deberían estar almacenado en la carpeta lib dentro del proyecto. Tras aceptar, las librerías se cargan, mostrando todas las clases disponibles (Figura 3).

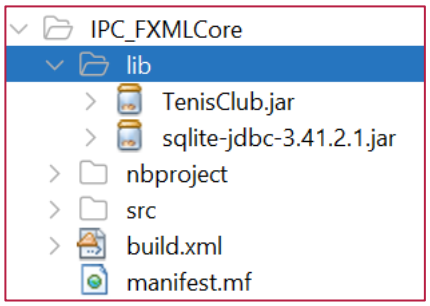

Figura 2. Carpeta lib

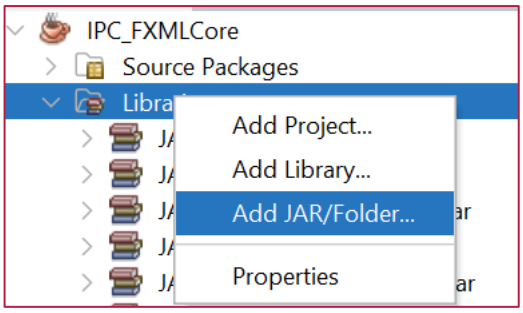

Figura 1 Seleccionar la opción de incluir la librería

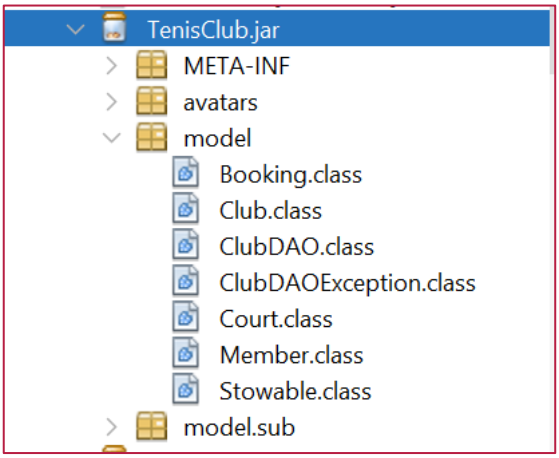

Figura 4 Clases en el jar TenisClub

2.1. Clases del modelo

En el modelo se definen varias clases, las que vas a necesitar utilizar para desarrollar la entrega son: **Club**, **Court**, **Member** y **Booking**

Club

La clase Club guarda toda la información del sistema, posee los siguientes atributos:

- String **name**: nombre del club
- int **bookingDuration**: duración en minutos de una reserva. 60 min por defecto, tal y como se indica en el enunciado.
- int **bookingSlots**; número de posibles reservas que se pueden realizar en un día y pista. 11 por defecto, tal y como se indica en el enunciado.
- **courts**: lista con las pistas del club. Por defecto se proporcionan 6 pistas.
- **members**: mapa con los miembros del club.
- **bookings**: lista con las reservas realizadas en el club.

Esta clase implementa el patrón Singleton, este patrón de programación obliga a que en toda la aplicación solo se pueda crear un objeto de esta clase. Para ello se define el constructor como privado, y se define un método estático en la clase que es el que se encarga de crear este solo objeto y proporcionarlo allí donde se necesite. El método es **getInstance().**

Además de este método se proporcionan métodos para modificar los atributos de la clase y poder añadir miembros al club y reservas de estos miembros. En el punto 2.2 se describen con detalle todos los métodos que proporciona esta clase.

Court

La clase Court almacena la información de una pista del club. Solo tiene el atributo name, que es accesible desde los correspondientes métodos *setter* y *getter*:

- String **name**: cadena que contiene el nombre del club.

Por defecto se crean 6 pistas en un club (pista 1, pista 2, …, pista 6). No se pueden añadir o eliminar pistas al club pero si que es posible modificar el nombre de cada una de ellas.

Member

La clase Member almacena toda la información sobre un miembro del club, los atributos de la clase son:

- String **name**: nombre del miembro.
- String **surname**: apellido del miembro.
- String **telephone**: cadena que contiene el teléfono del miembro.
- String **nickName**: credencial que usará el usuario para conectarse en el sistema.
- String **password**: contraseña que usará el usuario para conectarse en el sistema.
- String **creditCard**: cadena que contiene los 16 dígitos de la tarjeta de crédito del miembro. Puede estar vacía si no se proporcionó en el registro.
- Int **svc**: entero de 3 dígitos del código de verificación de la tarjeta.
- Image **image**: avatar con la imagen del miembro. Si en el constructor no se utiliza (null), se añade automáticamente un avatar por defecto.

Se pueden modificar todos los campos salvo **nickName**, para ello se dispone de los correspondientes *setter*.

El constructor de Member no es público, para crear un miembro nuevo hay que invocar al método de la clase Club, registerMember(). Este método se encargar de crear el nuevo miembro y de registrarlo en el club. El método retorna el objeto del tipo Member creado.

Booking

La clase Booking almacena la información de una reserva, los atributos de la clase son:

- LocalDateTime **bookingDate**: fecha y hora en la que el usuario realiza la reserva.
- LocalDate **madeForDay**: fecha para la que se hace la reserva, es decir, el día en el miembro del club quiere jugar.
- LocalTime **fromTime**: Hora de inicio de la reserva.

- Boolean **paid**: la reserva ha sido pagada con la tarjeta de crédito.
- Court **court**: pista del club para la que se ha reservado.
- Member **member**: miembro del club que ha reservado la pista.

El único campo que se puede modificar en un objeto de la clase Booking es **paid**, para ello disponemos del método setPaid(Boolean b).

El constructor de Booking no es público, para crear una reserva hay que invocar el método de la clase Club registerBooking(…). Este método se encargar de crear la reserva y de registrarla en el club. El método retorna el objeto del tipo Booking creado.

En el caso de querer anular una reserva hay que invocar al método de la clase Club removeBooking(Booking b). Este método borra la reserva sin tener en consideración las posibles restricciones expresadas en el escenario anular reserva.

2.2. La clase Club

Como hemos dicho esta clase implementa el patrón *Singleton*, mediante el método **getInstance()** recuperamos el objeto de la clase club.

Tras obtener el objeto de la clase *Club*, es posible acceder a toda la información almacenada a través de diferentes métodos públicos. Además de los correspondientes setter y getter se han implementado métodos que faciliten el desarrollo de la entrega. En la Tabla 1 se describe la funcionalidad aportada en todos estos métodos.

Tabla 1. API de la clase Club

public static Club getInstance()	Crea un objeto de la clase *Club*, si no había sido instanciado previamente. Si ya se había instanciado, devuelve el mismo objeto. Cuando lo crea por primera vez, se recupera toda la información guardada en la base de datos. Si la base de datos no existe se crea una base de datos con las tablas necesarias y se añade la información por defecto del club, así como 6 pistas.
public String getClubName()	Devuelve el nombre del club
public int getClubBookingDuration()	Devuelve los minutos que dura una reserva en el club.
public int getClubBookingSlots()	Devuelve el número de reservas que pueden realizarse en una pista cada día.
public ArrayList<Member> getMembers()	Devuelve un *ArrayList* no modificable con todos los miembros del club.
public ArrayList<Court> getCourts()	Devuelve un *ArrayList* no modificable con todas las pistas del club.
public ArrayList<Booking> getBookings()	Devuelve un *ArrayList* no modificable con todas las reservas realizadas. Las reservas se realizan para un día y hora concretos. Este *ArrayList* se devuelve ordenado por las reservas para los días más antiguos a los más actuales o más futuros.
public ArrayList<Booking> getUserBookings(String login)	**Copia** todas las reservas realizadas por el usuario con el *login* proporcionado en un *ArrayList* y lo devuelve. Si se modifica este *ArrayList*, **la lista original del club** de reservas **NO CAMBIA**.
public ArrayList<Booking> getCourtBookings(String courtName, LocalDate madeForDay)	**Copia** todas las reservas realizadas para una pista y día concretos en un *ArrayList* y lo devuelve. Si se modifica este *ArrayList*, **la lista original del club** de reservas **NO CAMBIA**.

public ArrayList<Booking> getForDayBookings(LocalDate forDay)	**Copia** todas las reservas realizadas para día concreto en un *ArrayList* y lo devuelve. Si se modifica este *ArrayList*, la **lista original del club** de reservas **NO CAMBIA**.
public boolean existsLogin(String login)	Devuelve True si existe algún miembro del club que esté utilizando el login dado, Falso en otro caso.
public Member getMemberByCredentials(String nickname, String password)	Devuelve el objeto *Member* correspondiente al miembro del club cuyo *nickname* y *password* coinciden con los dados. Si no existe, devuelve *null*.
public Court getCourt(String name)	Devuelve el objeto *Court* correspondiente a la pista cuyo nombre sea igual al nombre dado.
public boolean hasCreditCard(String nickname)	Devuelve si el miembro del club correspondiente al *nickname* proporcionó los datos de su tarjeta de crédito en el registro. Si no existe o no proporcionó los datos, devuelve false.
public Member registerMember(String name, String surname, String telephon, String login, String password, String creditCard, int svc, Image image)	Devuelve un objeto del tipo Member creado con los parámetros del método, Además de crear el objeto lo almacena en la base de datos y en el mapa de member
public boolean registerBooking(Booking booking)	Añade el objeto a la lista de bookings del club. El proceso puede fallar si hay un error en la base de datos, en este caso el valor de retorne es false
public Boolean removeBooking(Booking booking)	Elimina de la lista de bookings el bookking parámetro, en el caso de producirse algún error retorna false
public void setInitialData()	Función de utilidad que sirve para borrar toda la información guardada en la base de datos y dejar el sistema en su estado inicial (valores por defecto)
public void addSimpleData()	Función de utilidad que se puede utilizar para crear datos iniciales. Crea 5 usuarios (nicknme= user1, password 123456x).... Y crea un número aleatorio de bookings desde el día en el que se invoca el método y los siguientes 5 días

2.3. Uso de la librería desde el proyecto

Para acceder a los métodos de la librería, es necesario instanciar primero un objeto de la clase *Club*, utilizando para ello el método estático *getInstance*(). Después, puede obtenerse la información registrada o modificarla a través del API proporcionada. Por ejemplo, en el siguiente código se añade un nuevo miembro del club sin imagen:

```
String nickName = "user99";
String password = "123456X";
String name = "user";
String surname = "99";
String creditC = "0000111122223333";
int svc=123;
String tel=”666666666”;
Club club = Club.getInstance();
Meber result = club.registerMember(name, surname, tel, nickName, password, credit, svc, null);
```

3. Ayudas a la programación

3.1. Carga de imágenes desde disco duro

Los miembros del club pueden almacenar en su ficha de datos personales. Existen diversas formas de cargar una imagen desde el disco duro y mostrarla en una *ImageView*:

1. La imagen está en un subdirectorio del directorio src del proyecto, por ejemplo, llamado images:

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

2. La imagen está en cualquier parte del disco duro y tenemos el *path* completo de la misma:

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

3.2. Manejo de fechas y del tiempo

En la aplicación se deben gestionar atributos de tipo *LocalDateTime, LocalDate* y de tipo *DateTime*. A continuación, os explicamos algunos métodos de utilidad.

Obtención de la semana del año a la que pertenece una fecha

El siguiente código obtiene la semana a la que pertenece el día de hoy, así como el número del día de la semana (1 para lunes, 2 para martes, etc.)

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

Creación de una fecha o un campo de tiempo

Consulta el API de *LocalDate* y *LocalTime* para conocer todos los métodos que proporciona para crear un objeto de sus clases, pero algunos que te pueden resultar útiles son:

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

Actualizando una fecha

Es posible incrementar o decrementar días, meses, años, minutos, horas, etc. a los campos de tipo *LocalTime*, *LocalDate*, o *LocalDateTime* usando su propia API. Por ejemplo, el siguiente código incrementa en siete días la fecha actual, guardando el resultado en una nueva variable. También incrementa en un mes la fecha actual, salvando la información en otra variable. Finalmente, incrementa el tiempo actual en 90 minutos, guardando el resultado en otra variable de tipo *LocalTime*.

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

3.3. Configurar DatePicker

Los componentes DatePicker permiten seleccionar al usuario una fecha, pudiendo acceder al valor seleccionado a través de su propiedad valueProperty(). Es posible configurar la forma en la que se visualiza por defecto cada día del calendario, siendo posible deshabilitar algunos días, cambiar el color de fondo, etc. La forma en la que se configura esta visualización es similar a la que empleamos para configurar una ListView o una TableView. La diferencia es que en este caso debemos extender la clase DateCell, y usar el método setDayCellFactory. Por ejemplo, el siguiente código configura un DatePicker para que se inhabiliten los días anteriores al 1 de Marzo de 2020 (ver Figura 3).

```
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 );
                }
            };
        });
```

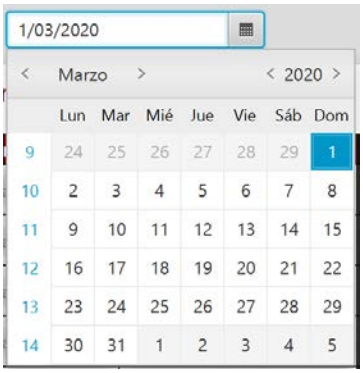

Figura 3. DataPicker configurado para inhabilitar días en el pasado

4. Instrucciones de Entrega

El proyecto debe implementar los escenarios de usos planteados en el punto Caso de Estudio (página 2). Debe diseñarse utilizando todo los principios, heurísticas y contenido visto en el apartado de teoría de la asignatura.

Con respecto a la entrega:

- Exporta el proyecto Netbeans a un fichero zip (opción `Fichero>Exportar Proyecto> A Zip`). Una alternativa es guardar toda la carpeta del proyecto en un fichero zip. Para reducir espacio puedes eliminar de este zip la carpeta dist
- Un único miembro del grupo sube el fichero zip a la tarea correspondiente, incluyendo en el campo de comentarios los nombres de los miembros del grupo.
- La fecha de entrega para todos los grupos es el **28 de mayo de 2023**

5. Evaluación

- Aquellos proyectos que no compilen o que no se muestren la pantalla principal al arrancar se calificarán con un cero.
- Se deberán incluir los diálogos de confirmación, errores, etc. que se considere necesario.
- Para evaluar el diseño de la interfaz de la aplicación se tendrán en consideración **las directrices estudiadas en clase de teoría**.
- Debe ser posible redimensionar la pantalla principal, cuyos controles se ajustarán adecuadamente para usar el espacio disponible (usa los contenedores vistos en clase: *VBox*, *HBox*, *BorderPane*, *GridPane*, etc.).