Curso 2018-2019

Entrega 1. Prácticas IPC

IPC

UPV - DSIC

Contenido

1. Caso de Estudio

La clínica de atención primaria Servicios Médicos Avanzados desea obtener una nueva aplicación de gestión de citas que permita a su personal de recepción la gestión de las citas de los pacientes de la clínica. El nuevo sistema manejará los datos de los pacientes, de los médicos que atienden en la clínica, así como las citas médicas que los pacientes conciertan.

Tras realizar un análisis de los requisitos del nuevo sistema, éstos se han plasmado en una serie de escenarios de uso que se describen a continuación (se describen solo los requisitos que deberán tenerse en cuenta).

Por otra parte, debido a la limitación del sistema de gestión de información a utilizar, no se contemplan operaciones de modificación de la información, de forma que solo será posible crear o eliminar los registros de las personas, doctores y citas médicas.

1.1. Añadir a un paciente

María, personal de recepción, atiende a un paciente de la clínica que le está solicitando una cita. Tras comprobar que no está presente en el listado actual de pacientes, accede a la funcionalidad que le permite darlo de alta aportando todos sus datos:

- Número de identificación personal (dni/nif/nie/pasaporte)
- Nombre
- Apellidos
- Teléfono
- Fotografía (opcional)

María pregunta al paciente si consiente añadir su foto a la ficha. El paciente le indica que no tiene ningún problema, así que María saca una foto del paciente con una pequeña cámara web y la almacena en el propio disco duro del ordenador donde reside el sistema de citas.

El sistema comprueba la información, añadiendo al paciente a la lista de pacientes de la clínica, e informando a María de que el proceso ha terminado con éxito. María comprueba que los datos introducidos son correctos, eligiendo el nuevo paciente de la lista de clientes, para obtener todos sus datos y así verificarlos con el paciente.

1.2. Eliminar a un paciente

Tras recibir la llamada de un nuevo cliente de la clínica, Sara, personal de recepción, había añadido al nuevo paciente José Carlos García al listado de pacientes de la clínica. Sin embargo, tras verificar sus datos, se da cuenta de que se ha equivocado en su número de teléfono, por lo que debe eliminarlo del sistema para volver a añadirlo después. Sara, selecciona la opción de borrar tras seleccionar al paciente de la lista de pacientes. El sistema comprueba **que el paciente no tiene ninguna cita asignada.** Desgraciadamente, Sara ya había añadido también la cita que había pedido el cliente, por lo que el sistema informa a Sara **que no puede eliminar al paciente** García.

Sara elimina la cita creada y vuelve a seleccionar al cliente de la lista de pacientes, y vuelve a acceder a la opción de borrar paciente. El sistema comprueba **que no hay ninguna cita registrada** para el paciente y lo elimina, informando a Sara de que se ha eliminado del sistema. Sara ya puede terminar de solventar su error, añadiendo de nuevo al paciente.

1.3. Mostrar datos detallados de un paciente

El doctor Ramírez necesita contactar con una de sus pacientes para poder modificarle las pautas de medicación que le había recetado, puesto que se había equivocado. Para ello se dirige a recepción donde encuentra a María. El doctor le da a María el nombre y apellidos de la paciente. María, busca al paciente en la lista de pacientes y tras

seleccionarla, pulsa en la opción correspondiente que le permite conocer todos los datos personales de la paciente. María anota el teléfono en un pósit y se lo proporciona al doctor.

1.4. Añadir a un médico

La clínica ha cerrado un nuevo convenio de colaboración con una nueva médica de atención primaria. La responsable de la clínica le indica a la nueva médica que haga el favor de pasar por recepción para darse añadirse al cuadro médico. Pedro la recibe, preguntándole si desea que se añada su foto a la ficha. La médica no desea que se añada su fotografía, así que no se añade esa información. Posteriormente, accede a la funcionalidad que le permite añadirla al cuadro médico, preguntándole a la médica los datos que necesita el sistema:

- Número de identificación personal (DNI/NIF/NIE/pasaporte)
- Nombre
- Apellidos
- Teléfono
- Fotografía (opcional)
- Días de la semana en la que pasará consulta
- Hora de inicio de la consulta (la hora de inicio es común a todos los días de consulta)
- Hora de fin de la consulta (la hora de fin es común a todos los días de consulta)
- Sala en la que pasará consulta, elegida de entre las salas disponibles en la clínica

El sistema comprueba la información y añade al nuevo paciente, informado a Pedro de que se ha añadido a la nueva médica al listado de médicos de la clínica. Pedro comprueba que no se ha equivocado en ningún dato, seleccionando al médico de la lista de médicos y accediendo a la opción de mostrar los detalles.

1.5. Eliminar un médico

Tras introducir los datos de un nuevo médico, Pedro se da cuenta de que se ha equivocado con su número de teléfono, por lo que tiene que eliminarlo. Selecciona el doctor de la lista, y tras ello, accede a la opción de eliminar al doctor. El sistema comprueba que **el doctor no tiene ninguna cita concertada,** y como no tiene ninguna cita, lo elimina e informa a Pedro de que el médico ha sido eliminado.

1.6. Mostrar los datos detallados de un médico

María necesita conocer el email del médico Pedro Martinez para enviarle los resultados de un paciente, así que accede a la opción para ver el listado de los médicos. Selecciona el médico de la lista, y accede a la opción de mostrar sus detalles. Tras ello, puede acceder a todos los datos del doctor (número de identificación, nombre, apellidos, teléfono, fotografía, días de la semana en la que pasará consulta, horas de inicio y fin de la consulta y la sala en la que pasará consulta).

1.7. Añadir una cita médica

Laura Soriano, paciente de la clínica, llama por teléfono para concertar una cita. María recibe la llamada, y le pregunta a Laura con qué médico desea la cita. Laura le indica que desea concertar la cita con su médico, Pedro Martínez. María accede a la opción de crear una cita médica. Tras ello, selecciona a Laura Soriano de la lista de Clientes, después a Pedro Martínez de la lista de Médicos e informa a Laura de las posibles citas disponibles, puesto que puede visualizar que días de la semana tiene citas, desde qué hora y hasta qué hora. Es posible concertar una cita en intervalos de quince minutos. Laura le indica que fecha y hora le viene bien, y María la selecciona y guarda la cita.

1.8. Eliminar una cita médica

A Laura Soriano le ha surgido un imprevisto que le impide acudir a su próxima cita, por lo que se decide a anularla. Llama a la clínica, donde le atiende María. María accede al listado de citas, y selecciona la cita a eliminar, pulsando la opción correspondiente. El sistema comprueba que **la cita todavía no ha sucedido**, así que la elimina e informa al usuario de que ha sido eliminada.

1.9. Mostrar una cita médica

Juan Hernandez había confirmado una cita médica en la clínica, pero no recuerda ni el día ni el médico que le tiene que atender, así que decide acercarse a la clínica para que le den de nuevo la cita en papel. Acude a recepción donde le atiende Pedro. Pedro accede al listado de citas de Juan, y selecciona la próxima planificada. Saca la información de la cita, indicando a Juan en qué fecha es la cita, la hora, el doctor y sala donde debe acudir.

2. Persistencia de los datos

La persistencia de la información se realizará a través de un fichero XML, utilizando la librería JAXB de java. En concreto, se os ha proporcionado una librería de acceso (*clinicDBAccess.jar*) capaz de almacenar y recuperar toda la información de la clínica: pacientes, doctores, salas de visita y citas. Para que la librería funcione correctamente, el archivo XML *clinicDB.xml* debe ubicarse en el home del usuario, que en sistemas Windows se encuentra en:

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

Donde `userName` es el nick del usuario con el que se ha realizado el inicio de sesión en Windows.

Si trabajáis en los equipos el laboratorio o accedéis desde el escritorio virtual, el home del usuario se ubica directamente en:

```
W:\
```

El archivo *clinicDB.xml* no debe manipularse manualmente, sino a través del API que la librería *clinicDBAccess.jar* proporciona. Como punto de partida, se os proporciona un archivo *clinicDB.xml* que contiene el nombre de varios médicos, pacientes, citas y todas las salas de visita de la clínica.

2.1. Inclusión de la librería *clinicDBAccess.jar* en el proyecto.

En primer lugar, descarga y guarda en la carpeta de tu proyecto el fichero *clinicDBAccess.jar,* que contiene la librería de acceso.

Los proyectos creados por Netbeans para aplicaciones FXML tienen una carpeta `Libraries` donde añadir las librerías externas que se desee. Para ello, basta situarse sobre esa carpeta, y desde el menú contextual (botón derecho del ratón), seleccionar la opción `Add JAR/Folder`, tal y como se muestra en la Figura 1.

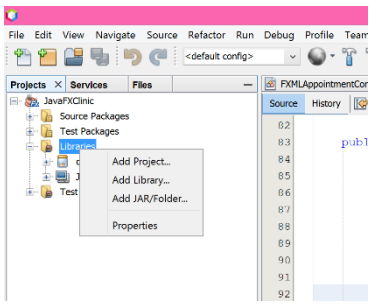

Figura 1. Seleccionar la opción de incluir la librería

Tras seleccionar la opción, aparecerá un diálogo donde se debe seleccionar el fichero jar que contiene la librería, que en este caso debería estar almacenado en la carpeta del proyecto (donde se había descargado). Tras aceptar, la librería se carga, mostrando todas las clases disponibles (Figura 2).

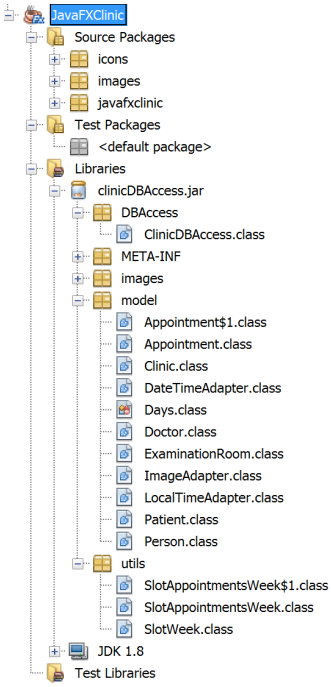

Figura 2. Librería ClinicDBAccess.jar incorporada al proyecto

2.2. API de *clinicDBAccess.jar*

Tal y como se ha mencionado anteriormente, no hay que manipular directamente el fichero XML que contiene la información de la clínica, sino acceder a ella a través del API proporcionado. Así, la librería proporciona una clase de acceso denominada ClinicDBAccess, que permite cargar la información de la clínica desde el fichero, así como almacenarla en él. Esta clase implementa el patrón Singleton, por lo que sólo puede instanciarse a un objeto de esta clase en una aplicación.

Tras obtener el objeto de la clase ClinicDBAccess, es posible acceder a toda la información de la clínica a través de diferentes métodos públicos, que pueden consultarse en la Tabla 1.

Tabla 1. API de la clase ClinicDBAccess

public static ClinicDBAccess getSingletonClinicDBAccess()	Crea un objeto de la clase ClinicDBAccess, si no había sido instanciado previamente. Si ya se había instanciado, devuelve el mismo objeto. Cuando lo crea por primera vez, comprueba si existe el archivo XML en el home del usuario y carga la información de la clínica. Si no existe el fichero, crea un nuevo fichero XML vacío, creando una clínica sin información.
public String getClinicName()	Devuelve el nombre de la Clínica
public void setClinicName(String newName)	Cambia el nombre de la clínica por el string proporcionado.
public ArrayList<Patient> getPatients()	Devuelve un ArrayList con todos los pacientes de la clínica.
public ArrayList<Patient> getPatients()	Devuelve un ArrayList con todos los doctores de la clínica.
public ArrayList<Appointment> getAppointments()	Devuelve un ArrayList con todas las citas registradas en la clínica.
public ArrayList<Appointment> getPatientAppointments(String patientId)	Devuelve un ArrayList con todas las citas registradas en la clínica del paciente con el identificador *patientId*
public ArrayList<Appointment> getDoctorAppointments(String doctorId)	Devuelve un ArrayList con todas las citas registradas en la clínica del doctor con el identificador *doctorId*
public boolean hasAppointments(Patient patient)	Devuelve TRUE si el paciente tiene alguna cita registrada en el sistema
public boolean hasAppointments(Doctor doctor)	Devuelve TRUE si el doctor tiene alguna cita registrada en el sistema
public ArrayList<ExaminationRoom> getExaminationRooms()	Devuelve un ArrayList con todas las salas de visitas disponibles en la clínica

public boolean saveDB()	Guarda el estado actual de la clínica en el fichero XML, almacenando los doctores, pacientes, citas y salas de visita. Devuelve TRUE si los datos son grabados correctamente. FALSE en caso contrario.

2.3. Clases del modelo

Clinic

La clase ClinicDBAccess tiene como atributo un objeto de la clase Clinic, que se corresponde con el nodo raíz de la persistencia de información en el XML. El API no proporciona una forma directa de recuperar este objeto, pero si te ofrece métodos públicos (Tabla 1) para recuperar la información necesaria de las listas con la información que maneja. En concreto, dentro de Clinic tenemos:

- **doctors**: lista con el cuadro médico de la clínica.
- **patientes**: lista con los pacientes de la clínica.
- **appointments**: lista con las citas registradas en la clínica.
- **examinationRooms**: lista con las salas de visitas de la clínica.

```
@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

Esta clase maneja los atributos compartidos entre doctores y pacientes, a fin de la reutilización de código. Así, para doctores y pacientes tenemos los siguientes atributos a través de métodos getter y setter:

- String **identifier**: Número identificativo de la persona, como por ejemplo el DNI, NIE o número de pasaporte.
- String **name**: Nombre de la persona
- String **surname:** Apellido de la persona
- String **telephon**: número de teléfono de contacto.
- Image **photo**: foto de la persona.

Doctor

Almacena toda la información sobre un doctor de la clínica, ofreciendo métodos setter y getter para acceder a los datos que se gestionan. Estos datos, son todos los de la clase Person más:

- ArrayList<Days> **visitDays**: lista con los días de la semana en la que este doctor pasa consulta. Days es un enumerado que puede tener los siguientes valores: {Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday}
- ExaminationRoom **examinationRoom**: sala en la que el doctor pasa consulta.
- LocalTime **visitStartTime**: hora y minutos de inicio de la consulta (común para todos los días en la que el doctor pasa consulta), por ejemplo 9:15. Los minutos válidos son: 00, 15, 30 y 45.
- LocalTime **visitEndTime**: hora y minutos correspondientes al fin de la consulta (común para todos los días en la que el doctor pasa consulta), por ejemplo 10:30. Los minutos válidos son: 00, 15, 30 y 45.

Patient

Almacena toda la información sobre un paciente de la clínica, ofreciendo métodos setter y getter para acceder a los datos que se gestionan. Los datos disponibles son los mismos que los explicados para la clase Person.

Appointment

Corresponde a los datos de una cita médica, siendo posible acceder a su información a partir de los métodos setter y getter correspondientes. En concreto, están disponibles:

- LocalDateTime **appointmentDateTime**: Fecha y hora de la cita médica.
- Doctor **doctor**: Doctor asignado a la cita médica
- Patient **patient**: paciente asistente a la cita médica.

La sala de visita donde se realiza la cita puede conocerse desde el doctor de la cita.

Además, la clase posee dos métodos públicos que permiten conocer a qué día de la semana y a qué semana del año corresponde la cita (Tabla 2).

Tabla 2. Métodos públicos de Appointment para facilitar la gestión del tiempo

public Days getAppointmentDay()	Devuelve el día de la semana a la que corresponde la cita, es decir uno de los siguientes valores: Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday
public int getAppointmentWeek()	Devuelve el número de semana a la que corresponde la cita.

ExaminationRoom

Sala de visita médica. Es posible conocer y establecer a través de los métodos getter y setter los siguientes atributos:

- int **identNumber**: número de la sala.
- String **equipmentDescription**: descripción del equipamiento de la sala.

Las clases DateTimeAdapter, ImageAdapter y LocalTimeAdapter se utilizan de forma interna por la librería para almacenar y recuperar del fichero XML los atributos de los tipos DateTime, Image y LocalTime respectivamente, por lo que no es necesario utilizarlas directamente.

2.4. Clases de Utils

SlotWeek

Para una hora concreta, indica la disponibilidad de asignar una nueva cita a un médico a lo largo de la semana. Es posible acceder a los siguientes atributos a través de sus correspondientes getters y setters:

- LocalTime **slot**: hora a la que corresponde el slot (por ejemplo 10:15)
- String **mondayAvailability**: disponibilidad en lunes.
- String **tuesdayAvailability**: disponibilidad en martes.
- String **wednesdayAvailability**: disponibilidad en miércoles.
- String **thursdayAvailability**: disponibilidad en jueves.
- String **fridayAvailability**: disponibilidad en viernes.
- String **saturdayAvailability**: disponibilidad en sábado.
- String **sundayAvailability**: disponibilidad en domingo.

SlotAppointmentsWeek

Esta clase ofrece un método estático que devuelve un ArrayList con los slots de disponibilidad de un médico para una semana del año en concreto, a partir de la información de citas de ese método. En concreto, la cabecera del método es:

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

donde **week** es la semana del año de la se obtendrá la disponibilidad; **visitDays**, los días de semana en los que atiende el médico; **visitStartTime**, la hora en la que empieza la consulta; **visitEndTime**, hora a la que termina la consulta; **appointments**, lista con todos las citas registradas para el doctor en el sistema.

El método comprueba el listado de citas, si alguna cita corresponde con la semana pedida, se añade el identificador del paciente en el slot correspondiente a la hora y día indicados. Si hay una combinación de hora y día disponible para añadir una nueva cita, escribe en el campo correspondiente "Free". Sin embargo, si el médico no visita en ese día y hora, retorna el valor "Not Available". En el siguiente ejemplo, se muestra como este método rellenaría un elemento del ArrayList correspondiente a las 10:30 de la mañana, para un doctor que visita de martes y jueves, de 10:30 a 12:30, y que tiene una visita ya registrada en la semana pedida para el jueves a las 10:30:

- Slot: 10:30
- mondayAvailability: "Not Available"
- tuesdayAvailability: "Free"
- wednesdayAvailability: "Not Available"
- thursdayAvailability: "4567833D"
- fridayAvailability: "Not Available"
- saturdayAvailability: "Not Available"
- sundayAvailability: "Not Available"

Esta clase puede utilizarse, si se desea, para visualizar la disponibilidad semanal de los médicos.

2.5. Uso de la librería ClinicDBAccess desde el proyecto

Acceso a la librería

Para acceder a los métodos de la librería, es necesario instanciar primero un objeto de la clase ClinicDBAccess, utilizando para ello el método estático *getSingletonClinicDBAccess*(). Después, puede obtenerse la información registrada o modificarla a través del API proporcionada. Por ejemplo, en el siguiente código se añade un nuevo paciente a la lista de pacientes de la clínica, almacenando el cambio en la base de datos:

```
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()
```

Utilización con componentes de la interfaz gráfica

Ya se ha comentado que es posible obtener desde un objeto de tipo ClinicDBAccess diferentes 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:

```
ClinicDBAccess clinicDBAccess = ClinicDBAccess.getSingletonClinicDBAccess();
ObservableList<Patient> patientsObservableList;
//Envolvemos el ArrayList de pacientes de de la clínica en una ObservableList
patientsObservableList = FXCollections.observableList(clinicDBAccess.getPatients());
// lPatients es una ListView<Patient>
lPatients.setItems(patientsObservableList); //Se conecta la lista con el ListView
Patient patient= new Patient("5307867J","Juan","Cafe Grandes","9376543", "juan@cafe.upv.es",avatar);
patientsObservableList.add(patient); //Se añade un Nuevo paciente
clinicDBAccess.saveDB(); //Se almacena el cambio en el fichero XML
```

Carga de imágenes almacenadas en la base de datos en una ImageView

Los médicos y los pacientes pueden tener almacenada una foto en su ficha. Para cargarla en una image View solo es necesario recuperar dicho campo del objeto correspondiente. Por ejemplo, para un paciente:

```
ClinicDBAccess clinicDBAccess = ClinicDBAccess.getSingletonClinicDBAccess();
int index = 0;
clinicDBAccess.getPatients().get(index);
//imagePhoto es una ImaveView
imagePhoto.imageProperty().setValue(patients.get(index).getPhoto());
```

Persistencia de los cambios

Para que todos los cambios realizados por el programa en los datos se persistan, es decir, se almacenen en el fichero XML, es necesario llamar al método saveDB(). Este método cuesta mucho tiempo, por lo que es aconsejable añadir la llamada cuando se cierre la aplicación:

```
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. Carga de imágenes desde disco duro

Los médicos y pacientes almacenan una imagen 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);
```

4. Manejo de fechas y del tiempo

En la aplicación se deben gestionar atributos de tipo LocalDateTime 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(2018, 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 o 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.

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

5. Instrucciones de entrega

El proyecto debe implementar los escenarios de usos planteados en el punto Caso de Estudio. Debe diseñarse una interfaz usable, que permita realizar las funcionalidades reflejadas en los diferentes casos de uso.

Debido a las limitaciones del formato XML no se contemplan ninguna funcionalidad de modificación, siendo necesario eliminar y agregar un nuevo objeto cuando se desee modificar alguno de los sus datos. Además, no se pueden eliminar doctores o pacientes que tengan citas pendientes. Tampoco se puede eliminar una cita que esté situada en el pasado, puesto que se supone que ya se realizó.

Con respecto a la entrega:

- Exporta el proyecto Netbenas a un fichero zip (opción `Fichero>Exportar Proyecto> A Zip`).
- 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 13 de abril de 2018.

6. 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, BordePane, GridPane, etc.).