Trabajo Práctico Integrador

Integrantes:

-Agüero, Sebastián
-Mujica, Juan Manuel
-Navarro, Pablo
-Vucetic, Ivo

Caja Negra

El informe con la elaboración de los datos de Prueba de Caja Negra de test de unidad se encuentra subido al GitHub del grupo un Excel con los datos de prueba / casos de uso.

Caja Blanca

Con respecto a caja blanca podemos observar luego de hacer el grafo de control que hay 7 regiones. Por lo que la complejidad ciclomática es igual a 7.

Podemos concluir entonces que hay una cota máxima de 7 caminos para recorrer la totalidad del código.

Los caminos posibles son:

C1={{683 a 686},688,719,721}

C2={{683 a 686},688,{690,691},692,700,704,708,711,712,721}

C3={{683 a 686},688,{690,691},692,700,704,705,711,712,721}

C4={{683 a 686},688,{690,691},692,693,694,695,697,704,705,711,712,721}

C5={{683 a 686},688,{690,691},692,700,704,705,711,712,713,714,713,714,721}

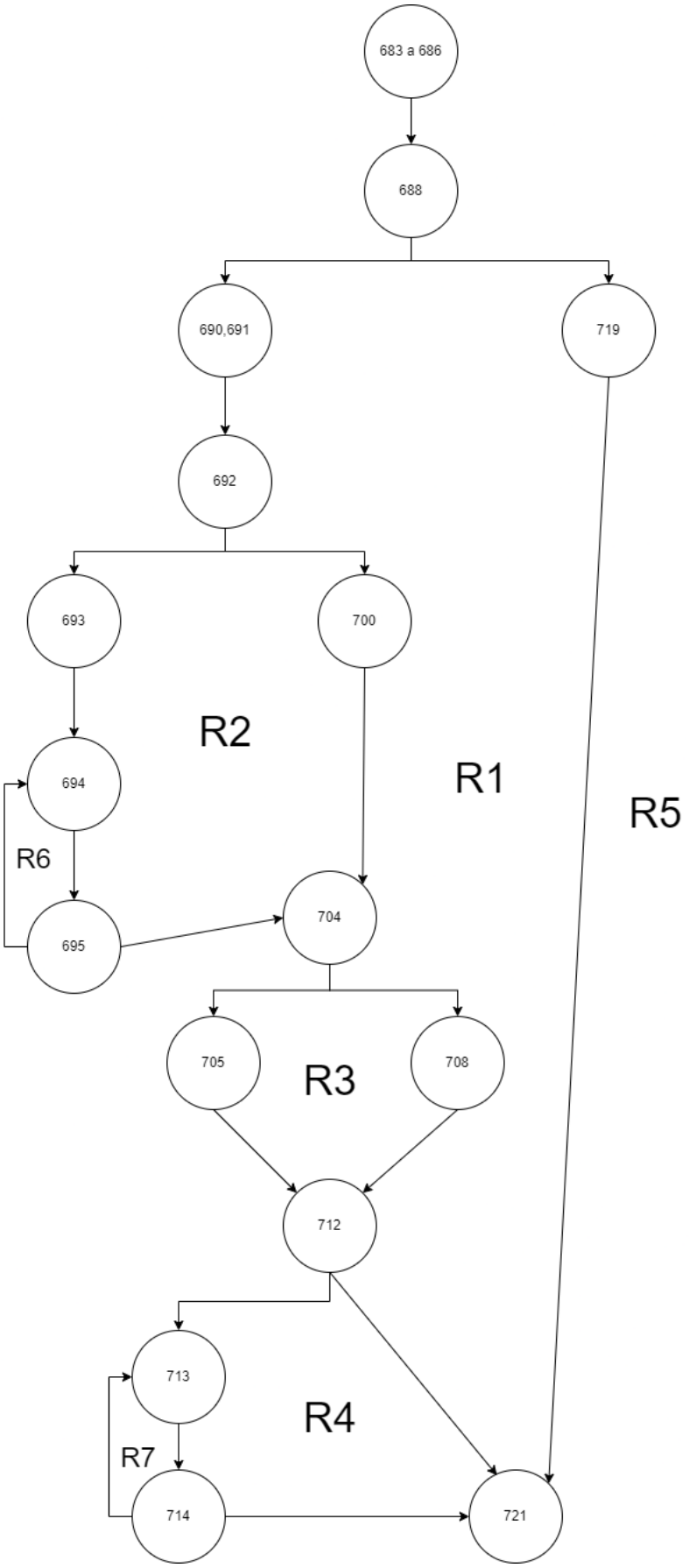
C6={{683 a 686},688,{690,691},692,700,704,705,711,712,713,714,721}

C7={{683 a 686},688,{690,691},692,693,694,695,694,695,697,704,705,711,712,713,714,713,714,713,714,721}

Podemos recorrer 3 caminos para abarcar la totalidad del código:

C1, C2 y C7

Grafo de Control:



Test de cobertura:
Para la cobertura se utilizó la cobertura de decisión para el total de los caminos posibles de decisión.

C1:

```
682    public double calculoImporteAdicionales(int numeroDeFactura, Calendar fechaDeSolicitud, ArrayList<Double> listaDeInsumos) {
683        double importeTotal, subTotalImpar, importeParcial;
684        GregorianCalendar fechaDeFactura;
685        int dias,i;
686        double A=0.7, B=0.4, C=1.3, D=0.85;
687
688        if(numeroDeFactura <= this.facturas.size()) { //si numeroDeFactura existe
689
690            fechaDeFactura = this.facturas.get(numeroDeFactura-1).fecha;
691            dias =  daysBetween(fechaDeFactura.getTime(),fechaDeSolicitud.getTime());
692            if(dias<10) {
693                subTotalImpar=0;
694                for(i=0;i<this.facturas.get(numeroDeFactura-1).getPrestaciones().size();i=i+2) {
695                    subTotalImpar+=this.facturas.get(numeroDeFactura-1).getPrestaciones().get(i).getSubTotal();
696                }
697                importeParcial = this.facturas.get(numeroDeFactura-1).totalFactura()-subTotalImpar*A;
698            }
699            else {
700                importeParcial = this.facturas.get(numeroDeFactura-1).totalFactura()*B;
701            }
702
703
704            if(this.facturas.get(numeroDeFactura-1).getPaciente().getClass().getName()=="modelo.Mayor") {
705                importeTotal = importeParcial * C;
706            }
707            else {
708                importeTotal = importeParcial * D;
709            }
710
711            double aleatorio = Math.random()*30+1;
712            if(aleatorio!=fechaDeFactura.get(Calendar.DATE)) {
713                for(i=0;i<listaDeInsumos.size();i++) {
714                    importeTotal+=listaDeInsumos.get(i);
715                }
716            }
717        }
718        else
719            importeTotal=0;
720
721        return importeTotal;
722
723    }
724
725    public static int daysBetween(Date d1, Date d2){
726            return (int)( (d2.getTime() - d1.getTime()) / (1000 * 60 * 60 * 24));
727     }
```

C2:

```
    public double calculoImporteAdicionales(int numeroDeFactura, Calendar fechaDeSolicitud, ArrayList<Double> listaDeInsumos) {
        double importeTotal, subTotalImpar, importeParcial;
        GregorianCalendar fechaDeFactura;
        int dias,i;
        double A=0.7, B=0.4, C=1.3, D=0.85;

        if(numeroDeFactura <= this.facturas.size()) { //si numeroDeFactura existe

            fechaDeFactura = this.facturas.get(numeroDeFactura-1).fecha;
            dias =  daysBetween(fechaDeFactura.getTime(),fechaDeSolicitud.getTime());
            if(dias<10) {
                subTotalImpar=0;
                for(i=0;i<this.facturas.get(numeroDeFactura-1).getPrestaciones().size();i=i+2) {
                    subTotalImpar+=this.facturas.get(numeroDeFactura-1).getPrestaciones().get(i).getSubTotal();
                }
                importeParcial = this.facturas.get(numeroDeFactura-1).totalFactura()-subTotalImpar*A;
            }
            else {
                importeParcial = this.facturas.get(numeroDeFactura-1).totalFactura()*B;
            }


            if(this.facturas.get(numeroDeFactura-1).getPaciente().getClass().getName()=="modelo.Mayor") {
                importeTotal = importeParcial * C;
            }
            else {
                importeTotal = importeParcial * D;
            }

            double aleatorio = Math.random()*30+1;
            if(aleatorio!=fechaDeFactura.get(Calendar.DATE)) {
                for(i=0;i<listaDeInsumos.size();i++) {
                    importeTotal+=listaDeInsumos.get(i);
                }
            }
        }
        else
            importeTotal=0;

        return importeTotal;

    }

    public static int daysBetween(Date d1, Date d2){
            return (int)( (d2.getTime() - d1.getTime()) / (1000 * 60 * 60 * 24));
     }
```

C7:

```
    public double calculoImporteAdicionales(int numeroDeFactura, Calendar fechaDeSolicitud, ArrayList<Double> listaDeInsumos) {
        double importeTotal, subTotalImpar, importeParcial;
        GregorianCalendar fechaDeFactura;
        int dias,i;
        double A=0.7, B=0.4, C=1.3, D=0.85;

        if(numeroDeFactura <= this.facturas.size()) { //si numeroDeFactura existe

            fechaDeFactura = this.facturas.get(numeroDeFactura-1).fecha;
            dias =  daysBetween(fechaDeFactura.getTime(),fechaDeSolicitud.getTime());
            if(dias<10) {
                subTotalImpar=0;
                for(i=0;i<this.facturas.get(numeroDeFactura-1).getPrestaciones().size();i=i+2) {
                    subTotalImpar+=this.facturas.get(numeroDeFactura-1).getPrestaciones().get(i).getSubTotal();
                }
                importeParcial = this.facturas.get(numeroDeFactura-1).totalFactura()-subTotalImpar*A;
            }
            else {
                importeParcial = this.facturas.get(numeroDeFactura-1).totalFactura()*B;
            }


            if(this.facturas.get(numeroDeFactura-1).getPaciente().getClass().getName()=="modelo.Mayor") {
                importeTotal = importeParcial * C;
            }
            else {
                importeTotal = importeParcial * D;
            }

            double aleatorio = Math.random()*30+1;
            if(aleatorio!=fechaDeFactura.get(Calendar.DATE)) {
                for(i=0;i<listaDeInsumos.size();i++) {
                    importeTotal+=listaDeInsumos.get(i);
                }
            }
        }
        else
            importeTotal=0;

        return importeTotal;

    }

    public static int daysBetween(Date d1, Date d2){
            return (int)( (d2.getTime() - d1.getTime()) / (1000 * 60 * 60 * 24));
     }
```

Como observación podemos mencionar la utilización de un mock en TestCalcularImporteAdicionalC7. Esto se debe a que se tenía que imponer la condición de que el atributo aleatorio del método sea diferente al día de la fecha de la factura. Por este motivo se hizo un mock de clase GregorianCalendar para que el método get(Calendar.DATE) devolviera 0 y por lo tanto siempre será diferente a un número aleatorio entre 1 y 31.

Test de persistencia

Módulo seleccionado: Médicos.

Para la realización del test de persistencia se consideraron los siguientes aspectos:

Escenarios:
E1: Colección de médicos vacío y archivo "Medicos.bin" inexistente
E2: Colección de médicos con al menos un médico y archivo "Medicos.bin" inexistente

1. Que el método guardarMedicos() cree un archivo con el nombre Medicos.bin
2. Verificar la lectoescritura de la colección de médicos, E1 y E2
3. Verificar que no despersista si no hay archivo creado E1 y E2

Con respecto al punto 1, se verificó que el método crea el archivo correspondiente correctamente, aunque cabe aclarar que no se utilizó la excepción FileNotFoundException ya que el método no la arrojaba.

Con respecto al punto 2, se tuvo en cuenta dos situaciones distintas:
-Una donde lo que se quería persistir era una colección vacía de médicos, en la cual se llegó efectivamente a un resultado correcto donde en primera instancia se persistió la colección vacía para luego despersistir y obtener nuevamente dicha colección vacía.
-Otra donde se tenía una colección con al menos un médico, y de la misma manera se realizó la persistencia y luego la despersistencia, llegando nuevamente al resultado esperado.

Con respecto al punto 3, no se pudo realizar el test ya que el método que se encargaba de despersistir no arrojaba la excepción FileNotFoundException.

Test de GUI para interfaces gráficas

Para la realización de este test se utilizó el método automático.
En primera instancia, simulamos los diferentes casos de prueba con sus respectivos casos de uso para abarcar los diferentes estados que tiene la interfaz de Agregar Pacientes, que fue la que seleccionamos para testear. Para esto se hizo uso de la clase java.awt.Robot.
Dado que para este tipo de tests la cantidad de casos posibles es enorme, decidimos utilizar los casos más representativos, los cuales son:

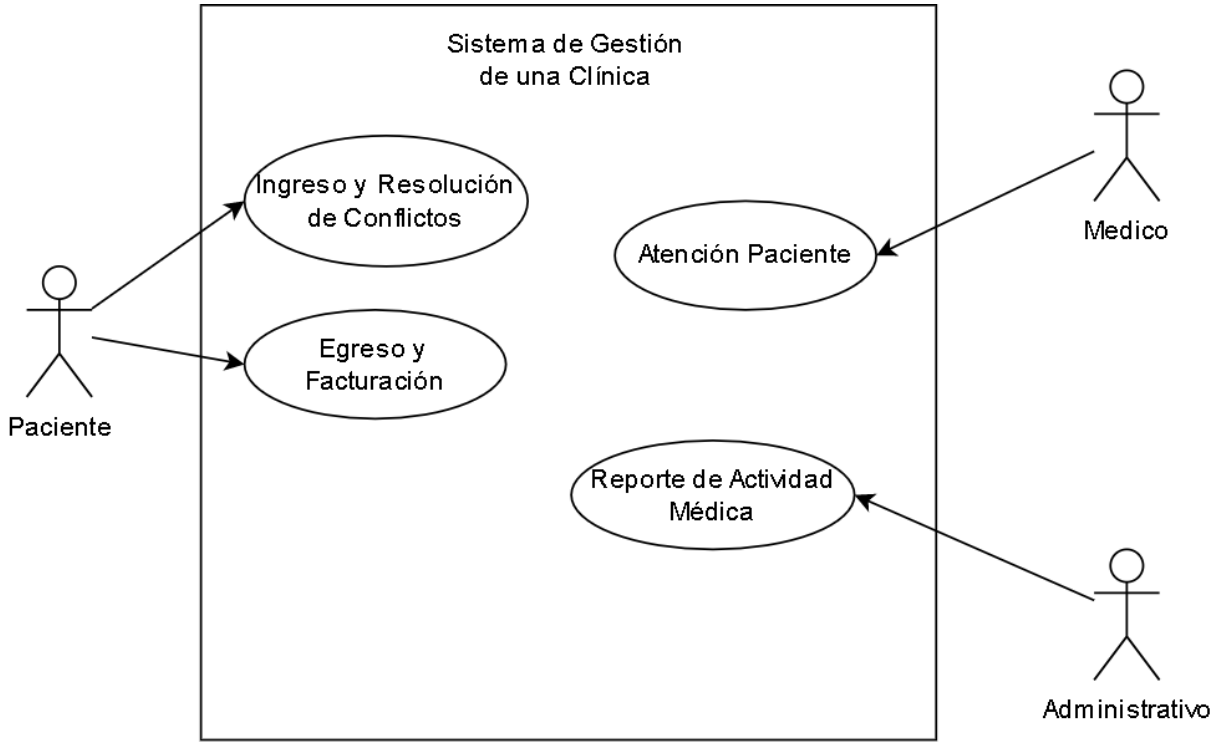
-----Escenario vacío------
1-Todo vacío -> botón deshabilitado
-----Pruebas solo con un campo-----
2-Solo nombre y todo vacío --> botón deshabilitado
3-Solo apellido y todo vacío --> botón deshabilitado
4-Solo dni y todo vacío --> botón deshabilitado
5-Solo domicilio y todo vacío --> botón deshabilitado
6-Solo telefono y todo vacío --> botón deshabilitado
7-Solo ciudad y todo vacío --> botón deshabilitado
-----Pruebas agregando de a un campo-----
8- Nombre, Apellido y el resto vacío --> botón deshabilitado
9- Nombre, Apellido,dni y el resto vacío --> botón deshabilitado
10- Nombre, Apellido,dni,domicilio y el resto vacío --> botón deshabilitado
11- Nombre, Apellido,dni,domicilio,teléfono y el resto vacío --> botón deshabilitado
12- Nombre, Apellido,dni,domicilio,teléfono,ciudad y el resto vacío --> botón deshabilitado
-----Pruebas de los errores que puede arrojar-----
13- Todos campos completos y dni < 8 --> No se agrega paciente por dni incorrecto
14- Todos campos completos y dni > 10 No se agrega paciente por dni incorrecto
15- Todos campos completos y dni con String con distinto de números -->No se agrega paciente por dni incorrecto
16- Todos los campos completos y nombre con al menos un caracter con numero --> No se agrega paciente por nombre incorrecto
17-Todos los campos completos y apellido con al menos un caracter con numero --> No se agrega paciente por apellido incorrecto
18-Todos los campos completos y ciudad con al menos un caracter con numero --> No se agrega paciente por ciudad incorrecto
19- Todos los campos completos y al menos una letra --> No se agrega paciente por teléfono incorrecto
20- Todos los campos completos y teléfono con menos de 9 números --> No se agrega paciente por teléfono incorrecto
21 - Todos los campos completos y teléfono con más de 16 números --> No se agrega paciente por teléfono incorrecto
--Escenario con al menos un paciente ---
22- No se debe ingresar el paciente a la colección si ya se encuentra. --> No se agrega paciente repetido
23- Agrega un paciente que no se encuentra

Aclaraciones:

- Debido a la necesidad de poseer referencias a los distintos componentes de la interfaz gráfica, se debió realizar una pequeña modificación en el código original, la cual fue agregarle un nombre a cada componente, ya que sin estos era imposible realizar el test de GUI.
- Dado que el código original no poseía la estructura necesaria para testear las ventanas modales con una respuesta al usuario, ya que no se tenía nada que pueda almacenar el mensaje que estas arrojan, y dado que agregar esta funcionalidad al código implicaba una enorme modificación del mismo, que incluía creación de nuevas clases y cambiar mucho de clases ya existentes, no se realizó esa parte específica del test.

Test de Integración

Se decidió utilizar las pruebas de integración orientadas a objetos ya que era la mejor opción de poder describir los objetos-conceptos del dominio del problema. De esta manera se pudo describir los diferentes actores involucrados y los casos de uso de los cuales formaban parte. Por medio de los diagramas de secuencias de sistemas propios del análisis orientado a objetos se pudo representar gráficamente la integración con las diferentes clases y módulos.

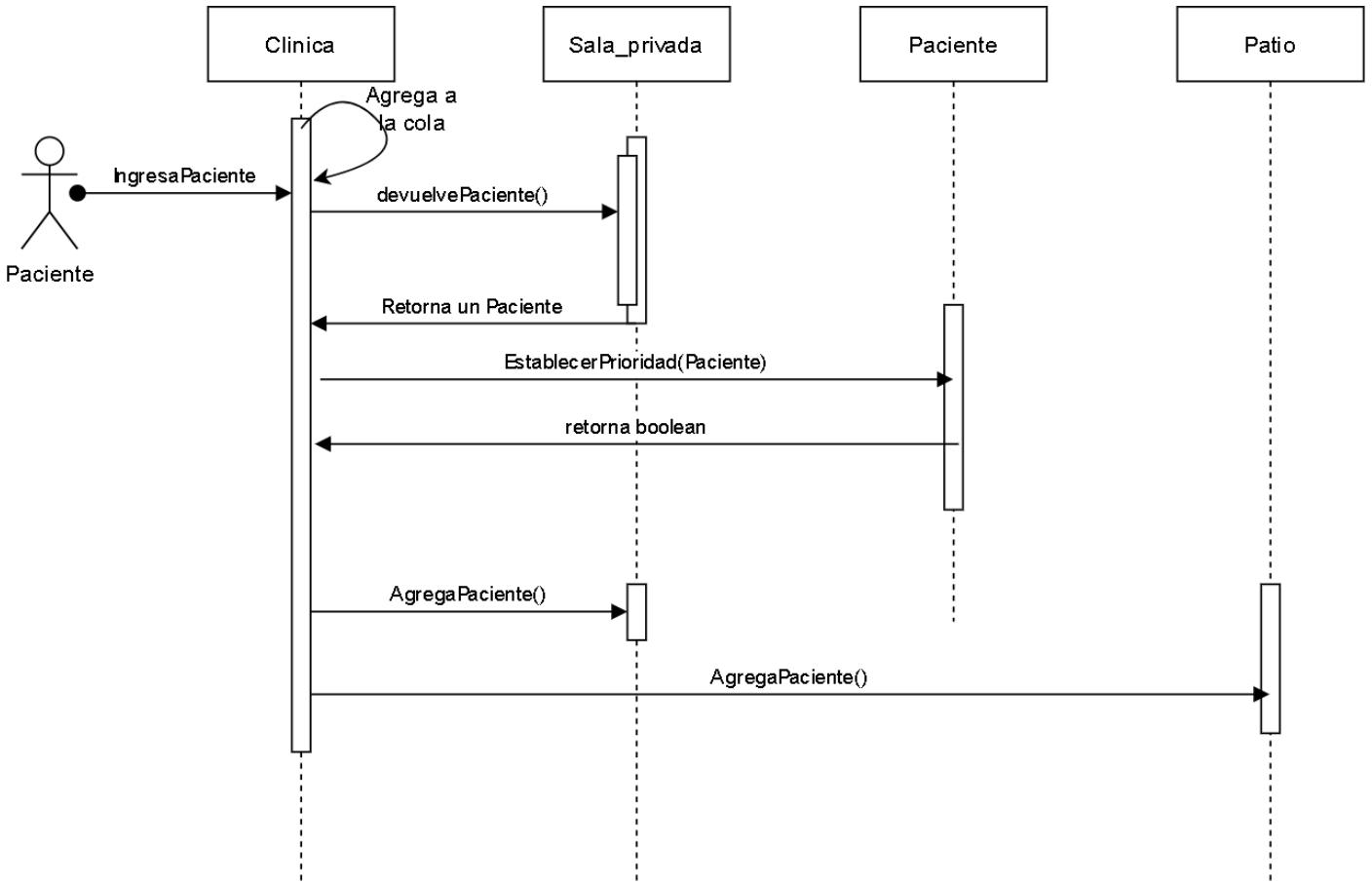


Casos de Uso:

Casos de uso **Ingreso del paciente a la clínica**:
El paciente llega a la ventana de atención al cliente de la clínica y se le otorga un número de orden. Luego es derivado a la Sala de Espera Privada o al Patio, según corresponda. A partir de ese momento el paciente es atendido cuando lo llamen.

El módulo de Ingreso y Módulo de Resolución de Conflictos
El módulo de ingreso dará de alta al paciente (si es la primera vez que ingresa a la clínica) o lo ubicará de la lista de pacientes y lo pondrá en la lista de espera según su orden de llegada, también determinará si se lo deriva a la Sala de Espera Privada o al Patio. En la sala de espera privada puede haber una sola persona. Si la sala está vacía, la persona ingresante se ubica en la Sala, si está ocupada y llega un nuevo paciente, la Sala se asignará de la siguiente forma:

Entre un niño y un joven, la sala queda para el niño y el otro se va al patio
Entre un joven y un mayor, la sala queda para el joven y el otro se va al patio.
Entre un mayor y un niño, la sala queda para el mayor y el otro se va al patio.



Casos de Prueba

Escenario:
E1: Sala privada vacía
E2: Sala privada con un Paciente

Tabla de particiones:

Condición	Clases Correctas	Clases Erróneas

Parámetro paciente	paciente válido (1) (Niño/Joven/Mayor)	
Estado Sala privada	vacía, con un paciente (Niño/Joven/Mayor)	

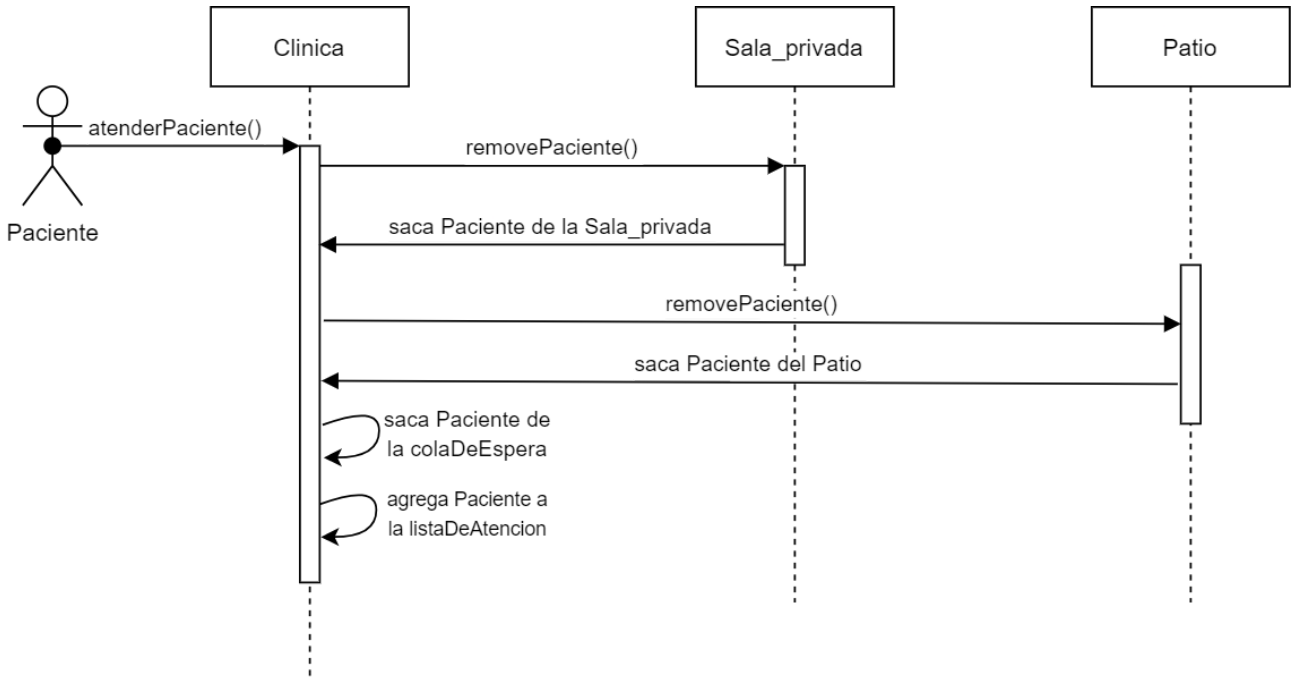
Batería de Pruebas:

Caso de Pruebas	Entrada	Salida esperada	Salida Obtenida
1, Escenario 1	El parámetro paciente de cualquier rango etario.	El paciente se agregó a la sala privada y a la cola de espera.	No hay errores
2, Escenario 2	El parámetro paciente de tipo Mayor. En la sala privada hay un niño.	El paciente mayor se agregó a la sala privada y a la cola de espera. El niño fue desplazado al patio.	No hay errores
3, Escenario 2	El parámetro paciente de tipo Mayor. En la sala privada hay un joven.	El paciente mayor se agregó al patio y a la cola de espera. El joven no fue desplazado.	No hay errores
4, Escenario 2	El parámetro paciente de tipo Mayor. En la sala privada hay un mayor.	El parámetro paciente es agregado al patio y a la cola. El paciente que estaba en la sala privada no fue desplazado.	No hay errores

Caso de uso **Atención del paciente**:

El paciente es atendido y se lo retira de la lista de espera. También se lo retira de la Sala de Espera o del Patio.

El módulo Atención
El módulo solamente retira al paciente de la espera y lo ubica en la Lista de Pacientes en Atención.



<u>Casos de Prueba</u>

Escenarios:

E1 Patio con al menos un Paciente, los pacientes también están en la cola de espera.

E2 Patio sin Paciente y Sala Privada con un Paciente, el paciente también está en la cola de espera.

E3 Patio Sin pacientes, Sala Privada sin Pacientes y Cola de Espera sin Pacientes

Batería de Pruebas:

Casos de Prueba	Entrada	Salida Esperada	Salida Obtenida
1, Escenario 1	Hay un Paciente en el Patio Sala Privada sin Paciente	Saca paciente de la colaDeEspera Agrega Paciente a la listaDeAtendidos Saca Paciente del Patio	No hay Errores
2, Escenario 2	Hay un Paciente en la Sala Privada Patio sin Pacientes	Saca paciente de la colaDeEspera Agrega Paciente a la listaDeAtendidos Saca Paciente de la Sala Privada	No hay Errores
3, Escenario 3	Patio y Sala Privada Sin Pacientes	Mensaje:”No quedan pacientes por ser atendidos”	No hay Errores

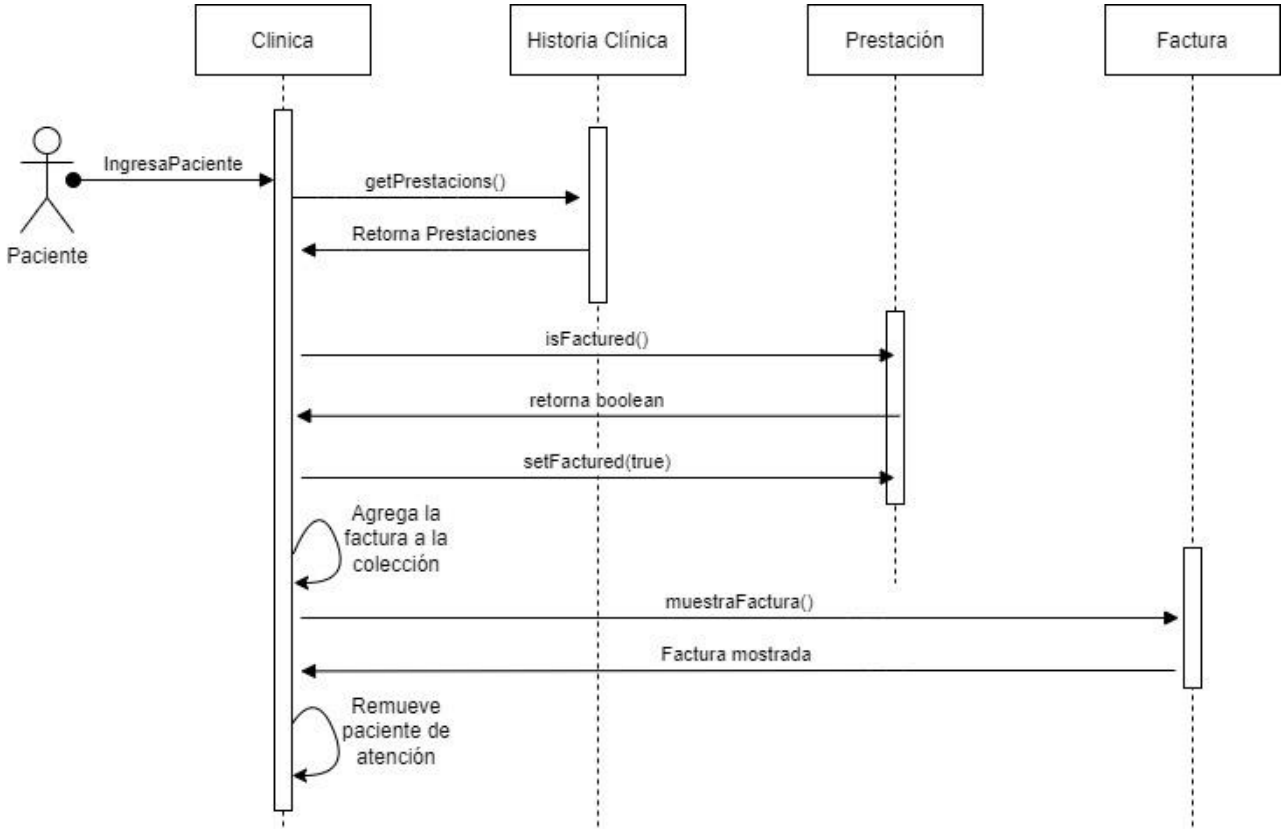
<u>Caso de uso **Egreso del paciente de clínica y Facturación al paciente**:</u>

Módulo de Egreso y Módulo de Facturación.

El módulo de egreso permitirá elegir un paciente de la Lista de Pacientes en Atención, y confeccionará la factura correspondiente, ingresando todas las prestaciones recibidas: Médico que lo atendió, días de internación, habitación. Luego se lo retirará de la Lista.

La factura de cada paciente tendrá la siguiente información: número de factura (autoincrementable), fecha, paciente, listado de prestaciones recibidas cada una con un importe (honorario médico, internación), importe total. El honorario médico que se le cobra al paciente sufrirá un 20% de incremento por sobre lo que el Médico cobre.

Pretación	Valor	Cantidad	subtotal
Nombre médico	Valor de la consulta	Cantidad de consultas	Subtotal
Habitación	Costo	Cantidad de días de internación	Subtotal



Casos de Prueba

Escenario:

E1 El paciente no tiene prestaciones en la Historia Clínica.
E2 El paciente tiene al menos una prestación en la Historia Clínica.

Parámetro Paciente
//Paciente no encontrado Exception, si es que no está en la lista de atención

Tabla de particiones:

Condición	Clases Correctas	Clases Erróneas
Parámetro paciente	paciente válido	
Estado de la colección Historias Clínicas	Sin prestaciones, con al menos una prestación.	
Estado de la lista de atención	Con ese paciente, Sin ese paciente	

Batería de Prueba:

Casos de Prueba	Entrada	Salida Esperada	Salida Obtenida
1, Escenario 1	El paciente no tiene prestaciones en la Historia Clínica.	Muestra la factura vacía	Error: no logra ejecutar porque hay un nullpointer a la lista de Prestaciones
2, Escenario 2	El paciente tiene al menos una prestación en la Historia Clínica.	Muestra la factura con las prestaciones	No hay errores