

# **TP Grupal 2021**

## **Grupo 5**



### **Integrantes:**

- 1) Erik Reinoso
- 2) Nahuel Spatera
- 3) Agustín Patané

# **Índice:**

- [Caja Negra](#)
  - [Conclusiones](#)
- [Caja Blanca](#)
- [Test de Persistencia](#)
- [Test de GUI](#)
- [Test de Integración](#)

# Caja Negra

## Clase: Clinica

### Método: getInstance()

#### Escenarios

Nro escenario	Descripción
1	No existe ninguna instancia de Clínica.
2	Ya existe una instancia de Clínica

#### Tabla de particiones

Condición de entrada	Clases Válidas	Clases Inválidas
No tiene parámetros	Instancia clínica. Escenario 1 <b>1.1</b> Instancia clínica. Escenario 2 <b>1.2</b>	

#### Batería de pruebas

Tipo de clase	Valores entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Válida	() Escenario 1	Creación de nueva instancia de clínica	1.1	Se crea una nueva instancia de clínica
Válida	() Escenario 2	No crear una nueva instancia de Clínica	1.2	No se crea una nueva instancia de Clínica

### Método: addMedico(IMedico medico)

#### Escenarios

Nro escenario	Descripción
1	No hay médicos registrados en la clínica
2	Hay 3 médicos registrados la clínica con matrículas: 2345, 8888 y 9129

Tabla de particiones

Condición de entrada	Clases Válidas	Clases Inválidas
médico	<u>medico no pertenece a la clínica</u> Escenario 1 <b>1.1</b> Escenario 2 <b>1.2</b>	<u>médico ya está registrado en la clinica</u> Escenario 2 <b>2</b>

Batería de pruebas

Tipo de clase	Valores entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Válida	(médico) Escenario 1	Medico agregado a la clinica	1.1	Se agrega el médico a la clínica
Válida	(médico) Escenario 2 matrícula = 3333	Medico agregado a la clinica	1.2	Se agrega el médico a la clínica
Invalida	(médico) Escenario 2 matrícula = 8888	MedicoYaExisteException	2	ERROR(No arroja la excepcion)

Método: removeMedico(IMedico medico)

Escenarios

Nro escenario	Descripción
1	No hay médicos registrados en la clínica
2	Hay 3 médicos registrados la clínica con matrículas: 2345, 8888 y 9129

Tabla de particiones

Condición de entrada	Clases Válidas	Clases Inválidas
(médico)	<u>Médico pertenece a la clínica</u> Escenario 2 <b>1</b>	<u>Médico no pertenece a la clínica</u> Escenario 1 <b>2.1</b> Escenario 2 <b>2.2</b>

Batería de pruebas

Tipo de clase	Valores entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Válida	(médico) Escenario 2 matrícula = 8888	Médico removido de la clínica	1	Médico removido de la clínica
Inválida	(médico) Escenario 2 matrícula = 3333	Error en la eliminación del médico	2.2	No informa el error
Invalida	(médico) Escenario 1	Error en la eliminación del médico	2.1	No informa el error

Método: addPaciente(Paciente paciente)

Escenarios

Nro escenario	Descripción
1	No hay pacientes registrados en la clínica
2	Hay 3 pacientes registrados la clínica con DNI: "49231888", "39231888" y "23455123"

Tabla de particiones

Condición de entrada	Clases Válidas	Clases Inválidas
paciente	<u>paciente no registrado en la clínica</u> Escenario 1 <b>1.1</b> Escenario 2 <b>1.2</b>	<u>paciente ya registrado en la clínica</u> Escenario 2 <b>2</b>

Batería de pruebas

Tipo de clase	Valores entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Válida	(paciente) Escenario 1	Registro exitoso del paciente en la clínica	1.1	Registro exitoso del paciente en la clínica

Valida	(paciente) Escenario 2 dni="11149388"	Registro exitoso del paciente en la clínica	1.2	Registro exitoso del paciente en la clínica
Invalida	(paciente) Escenario 2 dni="49231888"	PacienteYaExisteException	2	PacienteYaExisteExc eption

## Método: removePaciente(Paciente paciente)

### Escenarios

Nro escenario	Descripción
1	No hay pacientes registrados en la clínica
2	Hay 3 pacientes registrados la clínica con DNI: "49231888", "39231888" y "23455123"

### Tabla de particiones

Condición de entrada	Clases Válidas	Clases Inválidas
(paciente)	<u>paciente registrado en la clínica</u> Escenario 2 <b>1</b>	<u>paciente no registrado en la clínica</u> Escenario 1 <b>2.1</b> Escenario 2 <b>2.2</b>

### Batería de pruebas

Tipo de clase	Valores entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Válida	(paciente) Escenario 2 dni="49231888"	Eliminación exitosa del paciente de la clínica	1	Eliminación exitosa del paciente de la clínica
Invalida	(paciente) Escenario 2 dni="222222"	NoEstaPacienteException	2.2	ERROR(No arroja la excepcion)
Invalida	(paciente) Escenario 1	NoEstaPacienteException	2.1	ERROR(No arroja la excepcion)

# Método: Ingreso(Paciente paciente)

## Escenarios

Nro escenario	Descripción
1	No hay pacientes registrados en la clínica
2	Hay 3 pacientes registrados la clínica: "49231888" Ninio ----> en sala privada "39231888" Joven "23455123". Mayor "12345678". Joven -----> en patio

## Tabla de particiones

Condición de entrada	Clases Válidas	Clases Inválidas
paciente	<u>paciente a ingresar no esta registrado</u> Escenario 1 <b>1.1</b> Escenario 2 <b>1.2</b>  <u>paciente a ingresar ya esta registrado</u> Escenario 2 • paciente es Mayor <b>1.3</b> • paciente es Joven <b>1.4</b>	

## Batería de pruebas

Tipo de clase	Valores entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Válida	(paciente) dni="44444444" Escenario 1	Paciente ingresado con éxito. Va a sala privada	1.1	Paciente ingresado con éxito. Va a sala privada
Válida	(paciente) dni="44444444" rango=Joven Escenario 2	Paciente ingresado con éxito a la lista de espera. Va al patio.	1.2	Paciente ingresado con éxito. Va a sala privada
Válida	(paciente) dni="23455123" rango=Mayor Escenario 2	Paciente ingresado con éxito a la lista de espera. Va a sala privada.	1.3	ERROR. El paciente que estaba en sala privada no se fue para el patio
Válida	(paciente) dni="39231888" rango=Joven	Paciente ingresado con éxito a la lista de espera. Va al	1.4	Paciente ingresado con éxito. Va a sala privada

	Escenario 2	patio.		
--	-------------	--------	--	--

Método: Atencion()

Escenarios

Nro escenario	Descripción
1	Hay 3 pacientes en la lista de espera: "49231888" Ninio 1er "39231888" Joven 2do "23455123". Mayor 3ero

Tabla de particiones

Condición de entrada	Clases Válidas	Clases Inválidas
listaEspera	Hay al menos un paciente en la lista de espera. <b>1</b>	

Batería de pruebas

Tipo de clase	Valores entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Válida	()	Primero en lista de espera pasa a ser atendido	1	Primero en lista de espera pasa a ser atendido

Método: EgresoYFacturacion(Paciente paciente, HashMap<String, Prestacion> prestaciones)

Escenarios

Nro escenario	Descripción
1	Lista de atencion vacia
2	Lista de atencion con los pacientes: DNI paciente1: "49231888" DNI paciente 2: "23455123"



**Tabla de particiones**

Condición de entrada	Clases Válidas	Clases Inválidas
paciente	paciente está en lista de atencion <b>1</b>	<u>paciente no está en lista de atencion</u> Escenario 1 <b>2.1</b> Escenario 2 <b>2.2</b>
prestaciones	prestaciones.size(>)0 <b>3.1</b> prestaciones.size() == 0 <b>3.2</b>	prestaciones==null <b>4</b>

**Batería de pruebas**

Tipo de clase	Valores entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Válida	(paciente,prestaciones) DNI paciente1: "49231888" prestaciones:internacion 1 y 3; consulta 1 y 2	Remover de lista de atencion el paciente y facturar	1, 3.1	Remueve de lista de atencion el paciente y factura
Válida	(paciente,prestaciones) DNI paciente1: "49231888" prestaciones:no tiene	Total a pagar =0 Remover de lista de atencion el paciente y facturar	3.2	Remueve de lista de atencion el paciente y factura
Invalida	(paciente,prestaciones) DNI paciente1: "77777777" prestaciones:internacion 1 y 3; consulta 1 y 2	NoEstaPacienteException	2.1	NoEstaPacienteException
Invalida	(paciente,prestaciones) DNI paciente1: "77777777" prestaciones:internacion 1 y 3; consulta 1 y 2	NoEstaPacienteException	2.2	NoEstaPacienteException
Invalida	(paciente,null) DNI paciente1: "49231888"	NullPointerException	4	ERROR(no lanza excepción). Crashea

**Método: ReporteActividadMedica(IMedico medico, Calendar fecha1, Calendar fecha2)**

**Escenarios**

Nro escenario	Descripción
1	HashMap de médicos vacío
2	Hay 3 médicos registrados la clínica con matrículas: 2345, 8888 y 9129

Tabla de particiones

Condición de entrada	Clases Válidas	Clases Inválidas
medico	<u>Médico registrado en hashmap.</u> Escenario 2 <b>1</b>	<u>Medico no registrado en hashmap</u> Escenario 1 <b>2.1</b> Escenario 2 <b>2.2</b>
fecha1,fecha2	fecha1<=fecha2 <b>3</b>	fecha1>fecha2 <b>4</b>

Batería de pruebas

Tipo de clase	Valores entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Válida	(medico,f1,f2) medico.matricula=8888 f1 antes que f2	Reporte de la actividad del médico entre f1 y f2	1,3	Reporte exitoso de la actividad del médico entre f1 y f2
Invalida	(medico,f1,f2) medico.matricula=8888 f1 antes que f2	Informar error de que no existe el medico	2.1	ERROR. (No arroja excepción)
Invalida	(medico,f1,f2) medico.matricula=9999 f1 antes que f2	Informar error de que no existe el medico	2.2	ERROR. (No arroja excepción)
Invalida	(medico,f1,f2) medico.matricula=8888 f1 después que f2	OrdenFechasIncorrectoExc eption	4	OrdenFechasIncorr ectoException

Método: agregaHabitacion(Habitacion habitacion)

Escenarios

Nro escenario	Descripción
1	Habitaciones registradas Habitación 1: numeroHabitacion=30 Habitación 2: numeroHabitacion=40

Tabla de particiones

Condición de entrada	Clases Válidas	Clases Inválidas
habitacion	<u>Agregar habitacion no registrada</u> <b>1</b>	<u>Agregar habitacion con número ya registrado</u> <b>2</b>

Batería de pruebas

Tipo de clase	Valores entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Válida	(habitacion) numHabitacion=50	Habitación registrada exitosamente	1	Habitación registrada exitosamente
Invalida	(habitacion) numHabitacion=40	ERROR. El número de habitación ya se encuentra registrado	2	ERROR(No arroja excepción)

## Clase: Joven

Método: Joven(String dni, String nombre, String apellido, String domicilio, String ciudad, String telefono, long nroHistoria)

Condición de entrada	Clases Válidas	Clases Inválidas
dni	dni.length>=7 && dni.lenght<=10 <b>1</b>	dni=NULL <b>2.1</b> dni.lenght>10 <b>2.2</b>
nombre	Solo compuesto por letras y espacios <b>3</b>	nombre==NULL <b>4.1</b> Contiene un carácter que no es letra ni espacio <b>4.2</b>
apellido	Solo compuesto por letras y espacios <b>5</b>	apellido==NULL <b>6.1</b> Contiene un carácter que no es letra ni espacio <b>6.2</b>

domicilio	Solo compuesto por letras,espacios y números <b>7</b>	domicilio==NULL <b>8.1</b> Contiene un carácter que no es letra ni espacio ni número. <b>8.2</b>
ciudad	Solo compuesto por letras y espacios <b>9</b>	ciudad==NULL <b>10.1</b> Contiene un carácter que no es letra ni espacio <b>10.2</b>
telefono	telefono.length>=7 && telefono.length <=13 <b>11</b>	teléfono==NULL <b>12.1</b> telefono.length >13 <b>12.2</b>

- ***Está mal el contrato. No tiene ningún tipo de sentido analizar los casos inválidos ya que todos los parámetros que recibe el constructor ya fueron validados en la ventana, por lo que lo correcto sería que por precondition todos los strings ya estén validados***

#### Batería de pruebas

Tipo de clase	Valores entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Válida	("42429388","Martin","Gomez Herrera","Avenida 230","Tandil","2257523265",190)	Joven creado correctamente	1,3,5,7,9	Joven creado correctamente

## Clase: Mayor

Método: Mayor(String dni, String nombre, String apellido, String domicilio, String ciudad, String telefono, long nroHistoria)

- ***Está mal el contrato. No tiene ningún tipo de sentido analizar los casos inválidos ya que todos los parámetros que recibe el constructor ya fueron validados en la ventana, por lo que lo correcto sería que por precondition todos los strings ya estén validados***

Condición de entrada	Clases Válidas	Clases Inválidas
dni	dni.length>=7 && dni.lenght<=10 <b>1</b>	dni=NULL <b>2.1</b> dni.lenght>10 <b>2.2</b>

nombre	Solo compuesto por letras y espacios <b>3</b>	nombre==NULL <b>4.1</b> Contiene un caracter que no es letra ni espacio <b>4.2</b>
apellido	Solo compuesto por letras y espacios <b>5</b>	apellido==NULL <b>6.1</b> Contiene un caracter que no es letra ni espacio <b>6.2</b>
domicilio	Solo compuesto por letras,espacios y numeros <b>7</b>	domicilio==NULL <b>8.1</b> Contiene un caracter que no es letra ni espacio ni numero. <b>8.2</b>
ciudad	Solo compuesto por letras y espacios <b>9</b>	ciudad==NULL <b>10.1</b> Contiene un caracter que no es letra ni espacio <b>10.2</b>
telefono	telefono.length>=7 && telefono.length <=13 <b>11</b>	telefono==NULL <b>12.1</b> telefono.length >13 <b>12.2</b>

#### Batería de pruebas

Tipo de clase	Valores entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Válida	("42429388","Martin","Gomez Herrera","Avenida 230","Tandil","2257523265",190)	Mayor creado correctamente	1,3,5,7,9	Mayor creado correctamente

## Clase: Ninio

Método: Ninio(String dni, String nombre, String apellido, String domicilio, String ciudad, String telefono, long nroHistoria)

#### Tabla de particiones

Condición de entrada	Clases Válidas	Clases Inválidas
dni	dni.length>=7 && dni.lenght<=10 <b>1</b>	dni=NULL <b>2.1</b> dni.lenght>10 <b>2.2</b>
nombre	Solo compuesto por letras y espacios <b>3</b>	nombre==NULL <b>4.1</b> Contiene un caracter que no es letra ni

		espacio <b>4.2</b>
apellido	Solo compuesto por letras y espacios <b>5</b>	apellido==NULL <b>6.1</b> Contiene un caracter que no es letra ni espacio <b>6.2</b>
domicilio	Solo compuesto por letras,espacios y numeros <b>7</b>	domicilio==NULL <b>8.1</b> Contiene un caracter que no es letra ni espacio ni numero. <b>8.2</b>
ciudad	Solo compuesto por letras y espacios <b>9</b>	ciudad==NULL <b>10.1</b> Contiene un caracter que no es letra ni espacio <b>10.2</b>
telefono	telefono.length>=7 && telefono.length <=13 <b>11</b>	telefono==NULL <b>12.1</b> telefono.length >13 <b>12.2</b>

- ***Está mal el contrato. No tiene ningún tipo de sentido analizar los casos inválidos ya que todos los parámetros que recibe el constructor ya fueron validados en la ventana, por lo que lo correcto sería que por precondition todos los strings ya estén validados***

Tipo de clase	Valores entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Válida	("42429388","Martin","Gomez Herrera","Avenida 230","Tandil","2257523265",190)	Niño creado correctamente	1,3,5,7,9	Niño creado correctamente

## Clase: Medico

Método: Medico(String dni, String nombre, String apellido, String domicilio, String ciudad, String telefono,long nroMatricula)

Tabla de particiones

Condición de entrada	Clases Válidas	Clases Inválidas
----------------------	----------------	------------------

dni	dni.length==8    dni.length==9 <b>(1)</b>	dni.length!=8 && dni.length!=9 <b>(2)</b>
nombre	Compuesto solo por letras alfabéticas y espacios <b>(3)</b>	Contener al menos un carácter que no sea una letra alfabética ni un espacio <b>(4)</b>
apellido	Compuesto solo por letras alfabéticas y espacios <b>(5)</b>	Contener al menos un carácter que no sea una letra alfabética ni un espacio <b>(6)</b>
domicilio	Contiene al menos una letra y un número <b>(7)</b>	No tiene ninguna letra <b>(8.1)</b> <b>(8)</b> No tiene ningún número <b>(8.2)</b>
ciudad	Compuesto solo por letras alfabéticas y espacios <b>(9)</b>	Contener al menos un carácter que no sea una letra alfabética ni un espacio <b>(10)</b>
teléfono	teléfono.length >=8 y telefono.length<=14 <b>(11)</b>	teléfono.length <8 <b>(12.1)</b> teléfono.length>14 <b>(12.2)</b>

- ***Está mal el contrato. No tiene ningún tipo de sentido analizar los casos inválidos ya que todos los parámetros que recibe el constructor ya fueron validados en la ventana, por lo que lo correcto sería que por precondition todos los strings ya estén validados***

#### Batería de pruebas

Tipo de clase	Valores entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Valida	("11234532", "Pepe", "Gonzalez", "San Luis 1234", "Batan", "5551234", 2345)	Médico creado correctamente	1,3,5,7,9,11	Médico creado correctamente

## Clase: MedicoFactory

Método                      getMedico(String                      especialidad,String contratacion,String posgrado,String dni, String nombre, String

apellido, String domicilio, String ciudad, String telefono,long nroMatricula)

Escenarios

Nro escenario	Descripción

Tabla de particiones

Condición de entrada	Clases Válidas	Clases Inválidas
especialidad	especialidad=="Clínica" (1.1) o especialidad=="Cirugía" (1.2) (1) o especialidad=="Pediatria" (1.3)	especialidad!="Clínica" (2.1) y especialidad!="Cirugía" (2.2) (2) y especialidad!="Pediatria" (2.3)
contratación	contratación=="Permanente" (3.1)  o (3) contratación=="Residente" (3.2)	contratación!="Permanente" (4.1)  y (4) contratación!="Residente" (4.2)
posgrado	posgrado=="Doctorado" (5.1)  o (5) posgrado=="Magister" (5.2)	posgrado!="Doctorado" (6.1)  y (6) posgrado!="Magister" (6.2)
dni	dni.length==8    dni.length==9 (7)	dni.length!=8 && dni.length!=9 (8)
nombre	Compuesto solo por letras alfabéticas y espacios (9)	Contener al menos un carácter que no sea una letra alfabética ni un espacio (10)
apellido	Compuesto solo por letras alfabéticas y espacios (11)	Contener al menos un carácter que no sea una letra alfabética ni un espacio (12)
domicilio	Contiene al menos una letra y un número (13)	No tiene ninguna letra (14.1)  (14)  No tiene ningún número (14.2)
ciudad	Compuesto solo por letras alfabéticas y espacios	Contener al menos un carácter que no sea una letra alfabética ni un espacio



teléfono	teléfono.length >=8 y teléfono.length<=14 (17)	teléfono.length <8 (18.1) teléfono.length>14 (18.2)
matricula	Compuesto únicamente por números (19)	Contiene al menos una letra u otro símbolo que no corresponde a un número (20)

- ***Está mal el contrato. No tiene ningún tipo de sentido analizar los casos inválidos ya que todos los parámetros que recibe el constructor ya fueron validados en la ventana, por lo que lo correcto sería que por precondition todos los strings ya estén validados.***

#### Batería de pruebas

Tipo de clase	Valores entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Válida	("Clínica", "Residente", "Magister" "11234532", "Pepe", "Gonzalez", "San Luis 1234", "Batan", "5551234", 2345)	Médico creado correctamente	1,3,5,7,9,11	Médico creado correctamente

## **Conclusiones de la caja negra**

Las pruebas realizadas con caja negra deslumbraron algunos errores en las salidas de los diversos métodos del sistema. A continuación pasamos a enumerar cada uno de los errores detectados en el sistema.

### 1. **addMedico(IMedico medico) - Clase(Clínica)**

En este método al recibir por parámetro un médico que ya está registrado en la clínica NO arroja la excepción YaExisteMedicoException.

### 2. **removeMedico(IMedico medico) - Clase(Clínica)**

Al querer eliminar un médico que no está registrado, si bien el método no altera nada, no se informa dicho error al usuario.

### 3.removePaciente(Paciente paciente) - Clase(Clínica)

Si se agrega un paciente que no está registrado en la clínica no se arroja la excepción NoEstaPacienteException.

### 4.Ingreso(Paciente paciente) - Clase(Clínica)

El paciente que estaba en la sala privada (Niño) no fue trasladado al patio al agregar un mayor. El mayor sí fue trasladado a la sala privada correctamente y a la lista de espera.

### 5.EgresoYFacturacion(Paciente paciente, HashMap<String, Prestacion> prestaciones) - Clase(Clinica)

Cuando el método recibe un hashmap de prestaciones nulo el sistema crashea. Debería arrojar NullPointerException.

### 6.Método: ReporteActividadMedica(IMedico medico, Calendar fecha1, Calendar fecha2) - Clase(Clinica)

El método al recibir un médico que no está registrado en la clínica no informa nada sobre dicho problema.

### 7.agregaHabitacion(Habitacion habitacion) - Clase(Habitacion)

Se pueden agregar dos habitaciones con el mismo número de habitación, lo cual no tiene sentido.

# Caja Blanca

El sistema debe proveer de un cálculo de importe adicional a la factura emitida al paciente, cuyo valor depende del total facturado, de la fecha de solicitud, de la fecha de facturación, de la lista de insumos.

Nota: el resto de la información no está documentada en la SRS y surgió por las conversaciones entre el stakeholder y el equipo de desarrollo.

Este método debe resolver el cálculo de un importe adicional que se le va a cobrar a ciertos pacientes. Lo que viene a continuación son consideraciones del código que no formarán parte del contrato, o sea, es información no disponible ni en la SRS ni en el contrato.

El valor del importe adicional se expresa en base al pseudocódigo:

Sí numeroDeFactura existe

Si la (fechaDeSolicitud – fecha de facturación) es menor a los 10 días

$\text{importeParcial} = \text{TotalFacturado} - (\text{SubTotalImpar} * A)$  //A: valor menor que 1

caso contrario

$\text{importeParcial} = \text{TotalFacturado} * B$  // B: valor menor que 1 y menor que A

Si el rango etario del paciente es MAYOR

$\text{ImporteTotal} = \text{importeParcial} * C$  // C: valor entre 1 y 2

caso contrario

$\text{ImporteTotal} = \text{importeParcial} * D$  // D: valor menor que 1 y mayor que A

Si ALEATORIO == día de la fecha de facturación

Respuesta = ImporteTotal

caso contrario

Respuesta = ImporteTotal + suma de valores de la listaDeInsumos.

caso contrario

Respuesta = 0

Cabe aclarar que como dice que no está documentado, sólo surgió a raíz de charlas con los stakeholder posiblemente deba ajustarse ciertos puntos, ya que no podría ser la forma correcta de aumentar o decrementar importes. También puede notarse que hay varios puntos ambiguos por ejemplo: “C: valor entre 1 y 2” no deja claro si incluye los límites y también si los valores A,B,C,D podrían modificarse en un futuro.

Se adjunta foto del código:

```

35 public double calculoImporteAdicionales(int numeroDeFactura, Calendar fechaDeSolicitud, ArrayList<Double> listaDeInsumos) {
36     double A=0.6, B=0.35, C=1.1, D=0.75;
37     double importeTotal = 0;
38
39     if(numeroDeFactura < Factura.getSiguienteNumero()) { //Si existe ese nro de factura.
40         boolean encontrado = false;
41         Iterator<Factura> iterator = facturas.iterator();
42         Factura f = null;
43         while (iterator.hasNext() && !encontrado) {
44             f = iterator.next();
45             encontrado = (f.getNroFactura() == numeroDeFactura);
46         }
47         int milisegundosEnUnDia = 86400000;
48         int dias = (int) (fechaDeSolicitud.getTime().getTime() - f.getFecha().getTime().getTime()) / milisegundosEnUnDia;
49         double importeParcial;
50         if(dias < 10){
51             double subTotalImpar=0;
52             int j = 1;
53             for(Prestacion i : f.getPrestaciones().values()) {
54                 if (j%2 == 0) {
55                     subTotalImpar += i.getSubtotal();
56                 }
57                 j++;
58             }
59             importeParcial = f.getTotal() - subTotalImpar * A;
60         }
61         else {
62             importeParcial = f.getTotal();
63         }
64         if(f.getPaciente().esMayor()) {
65             importeTotal = importeParcial * C;
66         }
67         else {
68             importeTotal = importeParcial * D;
69         }
70         if((int)(Math.random()*31) + 1 != f.getFecha().DAY_OF_MONTH) {
71             for(double i : listaDeInsumos) {
72                 importeTotal += i;
73             }
74         }
75     }
76     return importeTotal;
}

```

Al analizar detenidamente el código, se pueden notar una serie de errores en la línea 58 se olvidaron de multiplicar por B, haciendo que esto no cumpla con la especificación en caso de que la diferencia de días sea mayor a 10, también en la línea 53, debería ser si  $j \% 2 = 1$  ya se están evaluando las prestaciones pares, al contrario de lo pedido en las especificaciones. en la línea 63 el método esMayor() está mal implementado ya que busca pacientes por un índice erróneo. En la línea 70 el for que recorre la lista de insumos. No contempla que la misma puede ser Null, lanzando NullPointerException en dicho escenario a continuación se adjunta el link del análisis del grafo ciclotómico:

[https://drive.google.com/file/d/1\\_kzgV4zsqNumsGBaJz7IVMn07b\\_am5UY/view?usp=sharing](https://drive.google.com/file/d/1_kzgV4zsqNumsGBaJz7IVMn07b_am5UY/view?usp=sharing)

Para testear dicho método, se utilizó una Suite de Test, en la cual, debido a los errores de implementación resultaron fallidos todos aquellos que atravesaron la línea 38, ya que los valores por todas las ramas de ejecución tenían errores, tanto semánticos, como de implementación en métodos interiores. Para

poder avanzar más allá de línea 63, se tuvo que reemplazar el llamado a la función `f.getPaciente`, con un objeto Mock que simule los escenarios en que el paciente fuera mayor o no. También se empleó el uso de objeto Mock para simular el número aleatorio.

El test de cobertura fue corrido con la IDE IntelliJ con su herramienta de Test nativa la cual generó un reporte en formato HTML el cual se encuentra subido al repositorio, en la carpeta “test de cobertura”.  
[https://github.com/AgustinPatane/Taller\\_de\\_Programacion\\_1](https://github.com/AgustinPatane/Taller_de_Programacion_1)

## **Test de persistencia**

El código recibido persiste todos los módulos por separado, así que se eligió la persistencia del módulo de pacientes. A partir de lo visto en la teoría, se realizaron diversos métodos de test para comprobar que la persistencia funcione apropiadamente. Para dichos métodos se realizó el siguiente setup para que al momento de empezar la ejecución, el archivo que contiene a los pacientes sea eliminado (en caso de que no exista no se hace nada):

```
@Before
public void setUp() throws Exception
{
    Clinica.getInstance();
    Clinica.getInstance().vaciarPacientes();
    File archivo = new File("Pacientes.dat");
    if (archivo.exists())
        archivo.delete();
}
```

Luego, los métodos implementados fueron:

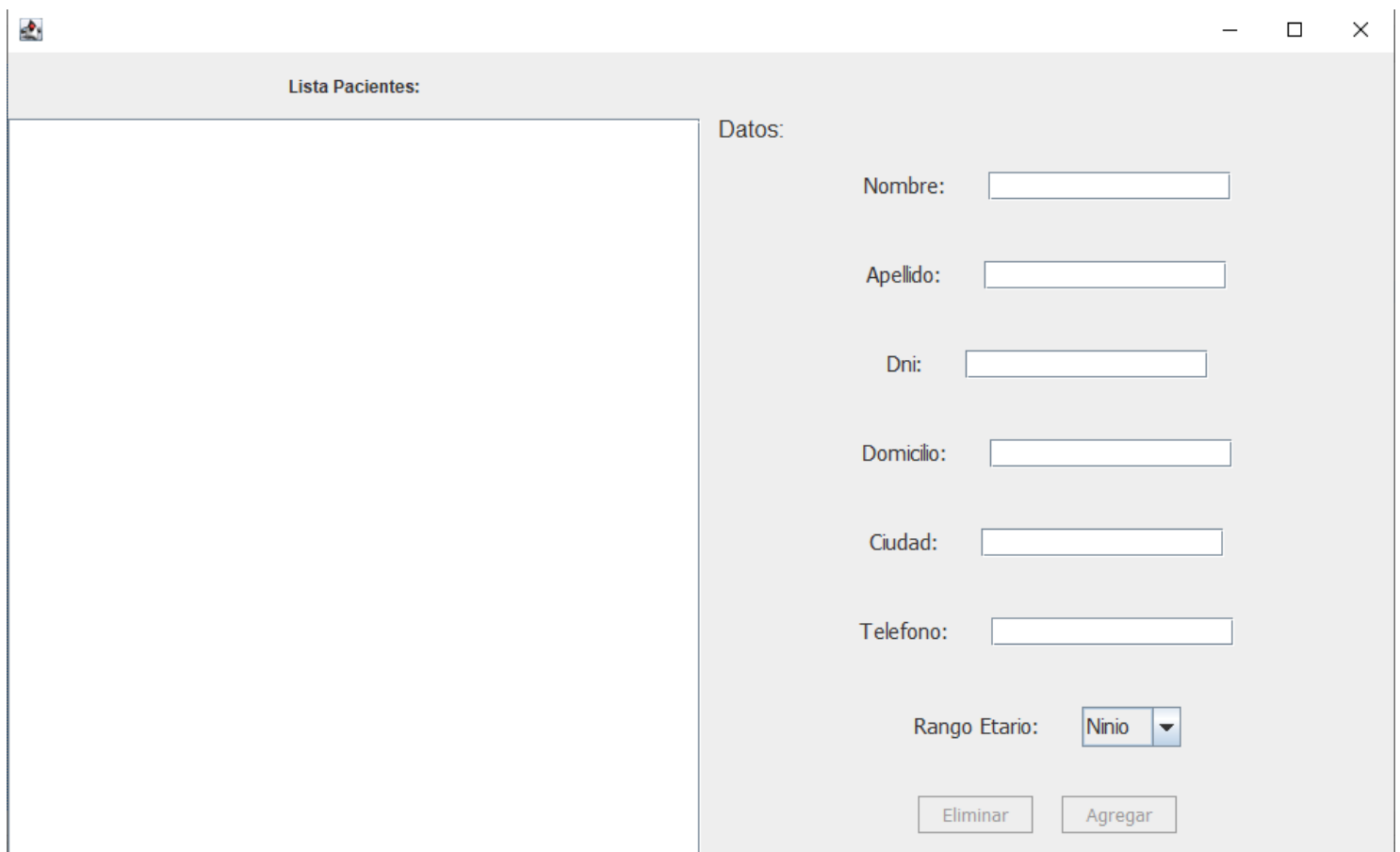
- **testCrearArchivo:** Aquí se verifica que el archivo “Pacientes.dat” se genere al invocar a `PersisteGeneral.guardaInformacionPacientes(Clinica.getInstance().getPacientesRegistrados())`
- **testPacientesVacioArchivo:** Aquí se comprueba que se persista un conjunto vacío correctamente (escribiendo la lista de pacientes vacía en el archivo para luego leerla, levantar los datos y comparando la colección guardada con la colección leída)
- **testClinicaConPacientes:** Aquí se comprueba que se persista un conjunto con tres pacientes correctamente (escribiendo la lista de pacientes en el archivo para luego leerla, levantar los datos y comparando la colección guardada con la colección leída)
- **testDespersistirSinArchivo:** Aquí se comprueba que la lectura lance una excepción ya que se intenta despersistir a los pacientes luego de haber eliminado el archivo “Pacientes.dat”.

A partir de los tests antes mencionados no se detectaron errores, por lo que aparentemente el módulo de persistencia de los pacientes funciona correctamente

# Test GUI

La técnica aplicada para el testeo de la interfaz gráfica fue la utilización de la clase robot ya que esta clase nos permite simular eventos de mouse y de teclado de manera automática (completar campos de texto, oprimir botones, mover el mouse, etc). Esta clase permite que, mediante muchas líneas de código, los testers puedan probar todas las combinaciones posibles dentro de la ventana que se quiere testear (por ejemplo: completar un único campo de texto y oprimir un botón de registro, completar todos los campos de texto correctamente y oprimir el botón para registrar, no completar ninguno e intentar registrar, etc).

La ventana seleccionada para el testing fue la de alta y baja de pacientes:



Lista Pacientes:

Datos:

Nombre:

Apellido:

Dni:

Domicilio:

Ciudad:

Telefono:

Rango Etario:

Dicha ventana posee, en su sector derecho, todos los textfields necesarios para el registro de un nuevo paciente (junto con una lista desplegable para elegir el rango etario), y luego, a la izquierda, una lista con todos los pacientes registrados actualmente en la clínica. También posee dos botones:

- Agregar: Botón que permite registrar a un paciente con los datos ingresados (se debe mantener deshabilitado hasta que se completen todos los campos y que el dni ingresado tenga siete caracteres o más).

- **Eliminar:** Botón que permite eliminar a un paciente seleccionado de la lista de pacientes del sector izquierdo de la pantalla (se debe mantener deshabilitado mientras no haya ningún paciente seleccionado).

Para realizar el testing se tuvieron que realizar diversos cambios en el código para poder cumplir con los requerimientos del test de GUI (setear el nombre de los componentes de la ventana, implementar los `JOptionPane` con los mensajes de confirmación de altas/bajas, agregar un método en la clase clínica para eliminar todos los pacientes registrados de una sola vez, en lugar de ir eliminando uno a uno, entre otros cambios). Luego, el testeo de la ventana fue separado en tres partes

- 1) **TestEnabledDisbled:** Aquí se verifica que los botones de Agregar y eliminar estén habilitados para su uso en los momentos correctos . Para llevar adelante este testeo se realizaron diversos métodos para probar todas las posibles combinaciones de ingreso de datos (por ejemplo: ingresar solo nombre, solo nombre y apellido, solo dni y domicilio, solo rango etario, etc) y de eliminación (por ejemplo no seleccionar ningún paciente, seleccionar uno correctamente y no seleccionar un paciente luego de la eliminación de otro e intentar eliminar).
- 2) **TestConDatos:** Aquí se testea que se realicen correctamente el alta y baja de pacientes (intentando registrar un paciente repetido, agregando un paciente correctamente, agregando dos, eliminando uno, eliminando tres, agregando y eliminando uno y agregando y eliminando tres) teniendo ya precargada la clínica con tres pacientes. Para comprobar que se hayan realizado correctamente los cambios se verifica que el número de pacientes antes y después del alta/baja coincida con lo esperado y también que el mensaje del `JOptionPane` coincida con lo correspondiente ("Paciente agregado con éxito", "Paciente eliminado con éxito", "ERROR: El dni del paciente ingresado coincide con uno ya registrado en la clínica").
- 3) **TestSinDatos:** Aquí se testea que se realicen correctamente el alta y baja de pacientes (o, agregando un paciente correctamente, agregando dos, agregando y eliminando uno y agregando y eliminando tres) teniendo la clínica sin ningún paciente. Para comprobar que se hayan realizado correctamente los cambios se verifica que el número de pacientes antes y después del alta/baja coincida con lo esperado y también que el mensaje del `JOptionPane` coincida con lo correspondiente ("Paciente agregado con éxito", "Paciente eliminado con éxito").

A partir de las tres secciones creadas para el testeo se encontró que el botón de eliminar quedó habilitado luego de eliminar a un paciente, por lo que se puede hacer click en el botón sin tener seleccionado ningún paciente, generando así una excepción del tipo `NullPointerException` ya que se invoca al método de la clase `Paciente` "`getDni()`" para una instancia nula. Por lo demás, la ventana responde correctamente a todos los estímulos planteados.

# **Test de integración**

Para realizar el test de integración se decidió abstraerse del total de las funcionalidades de la clínica, focalizándose únicamente en las que tienen que ver con los pacientes (registro de pacientes, ingreso a la salas de espera, atención del próximo paciente en fila y el egreso y facturación de un paciente). Se realizó una integración ascendente (se testea desde los módulos situados en los nodos finales de la jerarquía de control), por lo que no fue necesario la utilización de mocks en esta sección. Se adjunta gráfico con todas las ramas a modo ilustrativo:

<https://drive.google.com/file/d/14NAN7fdhxyi7Nuu-SPXOiqmrZiI0mVEH/view?usp=sharing>

Partiendo del gráfico se tienen 8 ramas:

- R1:** Agregar un paciente y que este no se encuentre registrado previamente en la clínica.
- R2:** Intentar agregar un paciente repetido a la clínica.
- R3:** Agregar un paciente a la lista de espera de atención en el patio (la sala privada está ocupada por otro paciente que tiene prioridad).
- R4:** Agregar un paciente a la lista de espera de atención en la sala privada (la sala privada está ocupada pero el paciente que quiere entrar tiene prioridad sobre el que está).
- R5:** Agregar un paciente a la lista de espera de atención en la sala privada (la sala privada está vacía).
- R6:** Se atiende al siguiente paciente en la lista de espera (por contrato tiene que haber por lo menos uno)
- R7:** Se intenta egresar y facturar a un paciente, pero este no se encuentra en la lista de atención, por lo que se lanza una excepción notificando esto
- R8:** Se egresa y factura a un paciente satisfactoriamente.