Taller de Programación I
Trabajo Práctico Grupal Final

Grupo Nº 4

Alumnos:
Echeverría, Noelia
Ércoli, Franco
Gutierrez, Matias
Mateos, Juan Cruz

Video: https://youtu.be/7-jR3yZo1h4
GitHub donde está subido el TP:
https://github.com/NoeliaEch/TestJava
(atención! cambiamos de GitHub)

Índice

Paquete Paciente

Clase Niño

Constructor

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
DNI	String o null (1)	Cualquier otro tipo de dato
Nombre	String o null (2)	Cualquier otro tipo de dato
Teléfono	String o null (3)	Cualquier otro tipo de dato
Domicilio	String o null (4)	Cualquier otro tipo de dato
Ciudad	String o null (5)	Cualquier otro tipo de dato
nro Historia clínica	Entero o null (6)	Cualquier otro tipo de dato

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	(“123456”, “Juan Perez”, “987654”, “casa”, “Mar del plata”, 1)	Se crea un niño con: DNI = 123456 Nombre = Juan Perez Teléfono = 987654 Domicilio = casa Ciudad = Mar del plata Historia clinica = 1	(1), (2), (3), (4), (5), (6)	Se crea un niño con: DNI = 123456 Nombre = Juan Perez Teléfono = 987654 Domicilio = casa Ciudad = Mar del plata Historia clinica = 1

Clase Joven

Constructor

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
DNI	String o null (1)	Cualquier otro tipo de dato
Nombre	String o null (2)	Cualquier otro tipo de dato
Teléfono	String o null (3)	Cualquier otro tipo de dato
Domicilio	String o null (4)	Cualquier otro tipo de dato
Ciudad	String o null (5)	Cualquier otro tipo de dato
nro Historia clínica	Entero o null (6)	Cualquier otro tipo de dato

Batería de pruebas			
	Entradas	Salidas	Clases Cubiertas
Clases correctas	(“123456”, “Juan Perez”, “987654”, “casa”, “Mar del plata”, 1)	Se crea una joven con: DNI = 123456 Nombre = Juan Perez Teléfono = 987654 Domicilio = casa Ciudad = Mar del plata Historia clinica = 1	(1), (2), (3), (4), (5), (6)

Clase Mayor

Constructor

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
DNI	String o null (1)	Cualquier otro tipo de dato

Nombre	String o null (2)	Cualquier otro tipo de dato
Teléfono	String o null (3)	Cualquier otro tipo de dato
Domicilio	String o null (4)	Cualquier otro tipo de dato
Ciudad	String o null (5)	Cualquier otro tipo de dato
nro Historia clínica	Entero o null (6)	Cualquier otro tipo de dato

Batería de pruebas			
	Entradas	Salidas	Clases Cubiertas
Clases correctas	(“123456”, “Juan Perez”, “987654”, “casa”, “Mar del plata”, 1)	Se crea una persona mayor con: DNI = 123456 Nombre = Juan Perez Teléfono = 987654 Domicilio = casa Ciudad = Mar del plata Historia clinica = 1	(1), (2), (3), (4), (5), (6)

Clase PacienteFactory

Método get Paciente

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
DNI	String o null (1)	Cualquier otro tipo de dato
Nombre	String o null (2)	Cualquier otro tipo de dato
Teléfono	String o null (3)	Cualquier otro tipo de dato
Domicilio	String o null (4)	Cualquier otro tipo de dato
Ciudad	String o null (5)	Cualquier otro tipo de dato
nro Historia clínica	Entero o null (6)	Cualquier otro tipo de dato
Rango Etareo	“Niño”, “Joven” o “Mayor” (7)	Cualquier otro dato o null (8)

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	(“123456”, “Juan Perez”, “987654”, “casa”, “Mar del plata”, 1, “Joven”)	Se crea una persona joven con: DNI = 123456 Nombre = Juan Perez Teléfono = 987654 Domicilio = casa Ciudad = Mar del plata Historia clinica = 1	(1), (2), (3), (4), (5), (6), (7)	Se crea una persona joven con: DNI = 123456 Nombre = Juan Perez Teléfono = 987654 Domicilio = casa Ciudad = Mar del plata Historia clinica = 1
Clases incorrectas	(“123456”, “Juan Perez”, “987654”, “casa”, “Mar del plata”, 1, “Anciano”)	Se lanza una excepción de tipo PacienteInvalidoException	(8)	Se lanza una excepción de tipo PacienteInvalidoException

Double dispatch

Para testear el double dispatch entre los diferentes rangos etarios para determinar quien permanecía en la sala privada teniendo en cuenta quien se encontraba previamente se realizó cada una de las comparaciones descritas en la siguiente tabla

Un / Contra	Niño	Joven	Mayor
Niño	queda el que estaba	queda el niño	queda mayor
Joven	queda niño	queda el que estaba	queda joven
Mayor	queda mayor	queda joven	queda el que estaba

El sistema respondió satisfactoriamente a las pruebas demostrando el correcto funcionamiento del doble dispatch.

Paquete médicos

Clase Cirujano

Método: constructor

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
DNI	String o null (1)	Cualquier otro tipo de dato
Nombre	String o null (2)	Cualquier otro tipo de dato
Teléfono	String o null (3)	Cualquier otro tipo de dato
Domicilio	String o null (4)	Cualquier otro tipo de dato
Ciudad	String o null (5)	Cualquier otro tipo de dato
nro matricula	Entero o null (6)	Cualquier otro tipo de dato
Honorario	Double o null (7)	Cualquier otro tipo de dato

Batería de pruebas				
	Entradas	Salidas esperada	Clases Cubiertas	Salidas obtenidas
Clases correctas	(“Juan Perez”, “123456”, “casa”, “Mar del plata”, “987654”, 1, 100)	Se crea un Cirujano con: DNI = 123456 Nombre = Juan Perez Teléfono = 987654 Domicilio = casa Ciudad = Mar del plata Matrícula = 1 Honorario = 100	(1), (2), (3), (4), (5), (6), (7)	Se creó un Cirujano con: DNI = 123456 Nombre = Juan Perez Teléfono = 987654 Domicilio = casa Ciudad = Mar del plata Matrícula = 1 Honorario = 100

Método calcula honorario

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
Cantidad de parámetros	Ningun parametro (1)	Uno o más parámetros de entrada (2)

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	()	Se devuelve el sueldo base del médico multiplicado por 1.1	(1)	Se devuelve el sueldo base del médico multiplicado por 1.1
Clases incorrectas	(“Carlos”)	Error	(2)	Error

Clase Pediatra

Método: Constructor

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
DNI	String o null (1)	Cualquier otro tipo de dato
Nombre	String o null (2)	Cualquier otro tipo de dato
Teléfono	String o null (3)	Cualquier otro tipo de dato
Domicilio	String o null (4)	Cualquier otro tipo de dato
Ciudad	String o null (5)	Cualquier otro tipo de dato
nro matricula	Entero o null (6)	Cualquier otro tipo de dato
Honorario	Double o null (7)	Cualquier otro tipo de dato

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	(“Juan Perez”, “123456”, “casa”, “Mar del plata”, “987654”, 1, 100)	Se crea un Pediatra con: DNI = 123456 Nombre = Juan Perez Teléfono = 987654 Domicilio = casa Ciudad = Mar del plata Matrícula = 1 Honorario = 100	(1), (2), (3), (4), (5), (6), (7)	Se creó un Pediatra con: DNI = 123456 Nombre = Juan Perez Teléfono = 987654 Domicilio = casa Ciudad = Mar del plata Matrícula = 1 Honorario = 100

Método calcula honorario

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
Cantidad de parámetros	Ningun parametro (1)	Uno o más parámetros de entrada (2)

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	()	Se devuelve el sueldo base del médico multiplicado por 1.07	(1)	Se devuelve el sueldo base del médico multiplicado por 1.07
Clases incorrectas	(“Carlos”)	Error	(2)	Error

Clase Clínico

Método: Constructor

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
DNI	String o null (1)	Cualquier otro tipo de dato
Nombre	String o null (2)	Cualquier otro tipo de dato
Teléfono	String o null (3)	Cualquier otro tipo de dato
Domicilio	String o null (4)	Cualquier otro tipo de dato
Ciudad	String o null (5)	Cualquier otro tipo de dato
nro matricula	Entero o null (6)	Cualquier otro tipo de dato
Honorario	Double o null (7)	Cualquier otro tipo de dato

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	(“Juan Perez”, “123456”, “casa”, “Mar del plata”, “987654”, 1, 100)	Se crea un Clínico con: DNI = 123456 Nombre = Juan Perez Teléfono = 987654 Domicilio = casa Ciudad = Mar del plata Matrícula = 1 Honorario = 100	(1), (2), (3), (4), (5), (6), (7)	Se creó un Clínico con: DNI = 123456 Nombre = Juan Perez Teléfono = 987654 Domicilio = casa Ciudad = Mar del plata Matrícula = 1 Honorario = 100

Método calcula honorario

Tabla de partición

Condición	Clases correctas	Clases incorrectas
Cantidad de parámetros	Ningun parametro (1)	Uno o más parámetros de entrada (2)

Batería de pruebas			
	Entradas	Salidas	Clases Cubiertas
Clases correctas	()	Se devuelve el sueldo base del médico multiplicado por 1.05	(1)
Clases incorrectas	(“Carlos”)	Error	(2)

Clase Magister

Método: Constructor

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
Encapsulado	IMedico o null (1)	Cualquier otro tipo de dato

Batería de pruebas			
	Entradas	Salidas	Clases Cubiertas
Clases correctas	(Un objeto de tipo IMedico con los siguientes datos: DNI = 123456 Nombre = Juan Perez Teléfono = 987654 Domicilio = casa Ciudad = Mar del plata Matrícula = 1 Honorario = 100)	Se crea un Magíster con: DNI = 123456 Nombre = Juan Perez Teléfono = 987654 Domicilio = casa Ciudad = Mar del plata Matrícula = 1 Honorario = 100	(1)

Método calcula honorario

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
Cantidad de parámetros	Ningun parametro (1)	Uno o más parámetros de entrada

Batería de pruebas			
	Entradas	Salidas	Clases Cubiertas
Clases correctas	()	Se devuelve el sueldo del encapsulado multiplicado por 1.05	(1)

Clase Doctorado

Método: Constructor

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
Encapsulado	IMedico o null (1)	Cualquier otro tipo de dato

Batería de pruebas			
	Entradas	Salidas	Clases Cubiertas
Clases correctas	(Un objeto de tipo IMedico con los siguientes datos: DNI = 123456 Nombre = Juan Perez Teléfono = 987654 Domicilio = casa Ciudad = Mar del	Se crea un Doctorado con: DNI = 123456 Nombre = Juan Perez Teléfono = 987654 Domicilio = casa Ciudad = Mar del plata	(1)

	plata Matrícula = 1 Honorario = 100)	Matrícula = 1 Honorario = 100	

Método calcula honorario

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
Cantidad de parámetros	Ningun parametro (1)	Uno o más parámetros de entrada

Batería de pruebas			
	Entradas	Salidas	Clases Cubiertas
Clases correctas	()	Se devuelve el sueldo del encapsulado multiplicado por 1.1	(1)

Clase PlantelPermanente

Método: Constructor

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
Encapsulado	IMedico o null (1)	Cualquier otro tipo de dato

Batería de pruebas			
	Entradas	Salidas	Clases Cubiertas
Clases correctas	(Un objeto de tipo IMedico con los siguientes datos:	Se crea un Permanente con: DNI = 123456	(1)

	DNI = 123456 Nombre = Juan Perez Teléfono = 987654 Domicilio = casa Ciudad = Mar del plata Matrícula = 1 Honorario = 100)	Nombre = Juan Perez Teléfono = 987654 Domicilio = casa Ciudad = Mar del plata Matrícula = 1 Honorario = 100	

Método calcula honorario

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
Cantidad de parámetros	Ningun parametro (1)	Uno o más parámetros de entrada

Batería de pruebas			
	Entradas	Salidas	Clases Cubiertas
Clases correctas	()	Se devuelve el sueldo del encapsulado multiplicado por 1.1	(1)

Clase Residente

Método: Constructor

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
Encapsulado	IMedico o null (1)	Cualquier otro tipo de dato

Batería de pruebas

	Entradas	Salidas	Clases Cubiertas
Clases correctas	(Un objeto de tipo IMedico con los siguientes datos: DNI = 123456 Nombre = Juan Perez Teléfono = 987654 Domicilio = casa Ciudad = Mar del plata Matrícula = 1 Honorario = 100)	Se crea un Residente con: DNI = 123456 Nombre = Juan Perez Teléfono = 987654 Domicilio = casa Ciudad = Mar del plata Matrícula = 1 Honorario = 100	(1)

Método calcula honorario

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
Cantidad de parámetros	Ningun parametro (1)	Uno o más parámetros de entrada

Batería de pruebas			
	Entradas	Salidas	Clases Cubiertas
Clases correctas	()	Se devuelve el sueldo del encapsulado multiplicado por 1.05	(1)

Clase MedicoFactory

Método: get Médico

Tabla de partición

Condición	Clases correctas	Clases incorrectas
DNI	String o null (1)	Cualquier otro tipo de dato
Nombre	String o null (2)	Cualquier otro tipo de dato
Teléfono	String o null (3)	Cualquier otro tipo de dato
Domicilio	String o null (4)	Cualquier otro tipo de dato
Ciudad	String o null (5)	Cualquier otro tipo de dato
nro matricula	Entero o null (6)	Cualquier otro tipo de dato
Honorario	Double o null (7)	Cualquier otro tipo de dato
Especialidad	“Cirujano”, “Pediatra” o “Clínico” (8)	null o cualquier otro string (9)
Contratación	“Permanente” o “Residente” (10)	null o cualquier otro string (11)
Grado	“Magíster” o “Doctorado” (12)	null o cualquier otro string (13)

Batería de pruebas			
	Entradas	Salidas	Clases Cubiertas
Clases correctas	(“Juan Perez”, “123456”, “casa”, “Mar del plata”, “987654”, 1, “Cirujano”, “Permanente”, “Doctorado”, 100)	Se crea un objeto de tipo IMedico	(1), (2), (3), (4), (5), (6), (7), (8), (10), (12)
Clases incorrectas	(“Juan Perez”, “123456”, “casa”, “Mar del plata”, “987654”, 1, “Carnicero”, “Permanente”, “Doctorado”, 100)	Lanza excepción “EspecialidadInvalidaException”	(9)
	(“Juan Perez”, “123456”, “casa”, “Mar del plata”, “987654”, 1, “Cirujano”, “Turno noche”, “Doctorado”, 100)	Lanza excepción “ContratacionInvalidaException”	(11)

	(“Juan Perez”, “123456”, “casa”, “Mar del plata”, “987654”, 1, “Cirujano”, “Permanente”, “Estudiante”, 100)	Lanza la excepción “PosgradoInvalidoException”	(13)

Paquete habitación

Clase Habitacion compartida

Método calcula costo

Condición	Clases correctas	Clases incorrectas
Días	Debe ser un número natural (1)	Cualquier otro tipo de dato

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	(1)	El costo de asignación de la habitación multiplicado por la cantidad de días	(1)	El costo de asignación de la habitación por la cantidad de días

Método agrega paciente

Escenario con habitación vacía

Condición	Clases correctas	Clases incorrectas
Paciente	Debe ser un objeto de tipo IPaciente o null (1)	Cualquier otro tipo de dato

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas

Clases correctas	(Objeto de tipo IPaciente)	Se agrega al paciente pasado por parámetro y devuelve true	(1)	Se agrega al paciente pasado por parámetro y devuelve true

Escenario con habitación llena

Condición	Clases correctas	Clases incorrectas
Paciente	Debe ser un objeto de tipo IPaciente o null (1)	Cualquier otro tipo de dato

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	(Objeto de tipo IPaciente)	No se agrega al paciente pasado por parámetro y devuelve false	(1)	No se agrega al paciente pasado por parámetro y devuelve false

Método elimina paciente

Escenario con habitación vacía

Condición	Clases correctas	Clases incorrectas
Paciente	Debe ser un objeto de tipo IPaciente o null (1)	Cualquier otro tipo de dato

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas

Clases correctas	(Objeto de tipo IPaciente)	No se elimina al paciente pasado por parámetro	(1)	No se elimina al paciente pasado por parámetro

Escenario con habitación llena

Condición	Clases correctas	Clases incorrectas
Paciente	Debe ser un objeto de tipo IPaciente o null (1)	Cualquier otro tipo de dato

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	(Objeto de tipo IPaciente)	Se elimina al paciente pasado por parámetro	(1)	Se elimina al paciente pasado por parámetro

Clase Terapia intensiva

Método calcula costo

Condición	Clases correctas	Clases incorrectas
Días	Debe ser un número natural (1)	Cualquier otro tipo de dato

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases	(1)	El costo de	(1)	El costo de

correctas		asignación de la habitación elevado a la cantidad de días		asignación de la habitación elevado a la cantidad de días

Método agrega paciente

Escenario con habitación vacía

Condición	Clases correctas	Clases incorrectas
Paciente	Debe ser un objeto de tipo IPaciente o null (1)	Cualquier otro tipo de dato

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	(Objeto de tipo IPaciente)	Se agrega al paciente pasado por parámetro y devuelve true	(1)	Se agrega al paciente pasado por parámetro y devuelve true

Escenario con habitación llena

Condición	Clases correctas	Clases incorrectas
Paciente	Debe ser un objeto de tipo IPaciente o null (1)	Cualquier otro tipo de dato

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas

Clases correctas	(Objeto de tipo IPaciente)	No se agrega al paciente pasado por parámetro y devuelve false	(1)	No se agrega al paciente pasado por parámetro y devuelve false

Método elimina paciente

Escenario con habitación vacía

Condición	Clases correctas	Clases incorrectas
Paciente	Debe ser un objeto de tipo IPaciente o null (1)	Cualquier otro tipo de dato

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	(Objeto de tipo IPaciente)	No se elimina al paciente pasado por parámetro	(1)	No se elimina al paciente pasado por parámetro

Escenario con habitación llena

Condición	Clases correctas	Clases incorrectas
Paciente	Debe ser un objeto de tipo IPaciente o null (1)	Cualquier otro tipo de dato

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases	(Objeto de tipo	Se elimina al	(1)	Se elimina al

correctas	IPaciente)	paciente pasado por parámetro		paciente pasado por parámetro

Clase Habitación privada

Método calcula costo

Condición	Clases correctas	Clases incorrectas
Días	Debe ser un número natural (1)	Cualquier otro tipo de dato

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	(1)	El costo de asignación de la habitación multiplicado por la cantidad de días	(1)	El costo de asignación de la habitación por la cantidad de días

Método agrega paciente

Escenario con habitación vacía

Condición	Clases correctas	Clases incorrectas
Paciente	Debe ser un objeto de tipo IPaciente o null (1)	Cualquier otro tipo de dato

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	(Objeto de tipo IPaciente)	Se agrega al paciente pasado por parámetro y devuelve true	(1)	Se agrega al paciente pasado por parámetro y devuelve true

Escenario con habitación llena

Condición	Clases correctas	Clases incorrectas
Paciente	Debe ser un objeto de tipo IPaciente o null (1)	Cualquier otro tipo de dato

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	(Objeto de tipo IPaciente)	No se agrega al paciente pasado por parámetro y devuelve false	(1)	No se agrega al paciente pasado por parámetro y devuelve false

Método elimina paciente

Escenario con habitación vacía

Condición	Clases correctas	Clases incorrectas

Paciente	Debe ser un objeto de tipo IPaciente o null (1)	Cualquier otro tipo de dato

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	(Objeto de tipo IPaciente)	No se elimina al paciente pasado por parámetro	(1)	No se elimina al paciente pasado por parámetro

Escenario con habitación llena

Condición	Clases correctas	Clases incorrectas
Paciente	Debe ser un objeto de tipo IPaciente o null (1)	Cualquier otro tipo de dato

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	(Objeto de tipo IPaciente)	Se elimina al paciente pasado por parámetro	(1)	Se elimina al paciente pasado por parámetro

Paquete Clínica

Contaremos con dos escenarios para los médicos, uno sin ningun medico y otro con algunos médicos agregados

Escenarios

Nro Escenario	Descripción
1	Escenario Con Pacientes
2	Escenario Sin Pacientes
3	Escenario sin Médicos
4	Escenario con Médicos
5	Clinica Con Medicos y Pacientes

Método Agrega médico

Escenario con médicos

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
Médico	IMedico que no se encuentra en el arreglo (1)	Cualquier otro tipo de dato o IMedico que se encuentra en el arreglo (2)

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	Un objeto de tipo IMedico que no se encuentra en arreglo	Se lo agrega al arreglo de médicos	(1)	Médico agregado al arreglo
Clases incorrectas	Un objeto de tipo IMedico que ya se	Se lanza una excepcion del tipo	(2)	Excepcion lanzada

	encuentra en el arreglo	“MedicoRepetidoException”		

Escenario sin médicos

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
Medico	IMedico (1)	Cualquier otro tipo de dato

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	Un objeto de tipo IMedico	Se lo agrega al arreglo de médicos	(1)	Médico agregado al arreglo

Metodo elimina medico

Escenario con médicos

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
Médico	IMedico (1)	Cualquier otro tipo de dato

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	Un objeto de tipo IMedico	Se lo elimina del arreglo de médicos	(1)	El médico fue eliminado

Escenario sin médicos

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
Medico	IMedico (1)	Cualquier otro tipo de dato

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	Un objeto de tipo IMedico	El arreglo de médicos tiene la misma longitud que previa la eliminación	(1)	El arreglo de médicos tiene la misma longitud que previa la eliminación

Método ingresar paciente

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
Paciente	IPaciente que no se encuentra ingresado (1)	IPaciente que ya se encuentra ingresado (2) o se ingresa el mismo paciente (3)

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	Un objeto de tipo IPaciente que no se encuentra ingresado	Se lo agrega a la lista de espera	(1)	Se lo agrega a la lista de espera
Clases incorrectas	Un objeto de tipo IPaciente	Lanza PacienteIngresa	(2)	Lanza PacienteRepeti

	que ya se encuentra ingresado	doException		doException
	Un objeto de tipo IPaciente que se encuentra repetido	Lanza PacienteRepetidoException	(3)	Lanza PacienteRepetidoException

Método ingresa nuevamente

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
Paciente	IPaciente que se encuentra en el histórico pero no el lista de espera (1)	IPaciente que se encuentra en el histórico y también el lista de espera (2)

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	Un objeto de tipo IPaciente	Se lo agrega a la lista de espera	(1)	El paciente es agregado a la lista de espera
Clases incorrectas	Un objeto de tipo IPaciente	Lanzamiento de excepcion	(2)	PacienteYaIngresadoException

Metodo agregar prestaciones

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
Habitación	Objeto de tipo IHabitacion que no esté llena (1)	Objeto de tipo IHabitacion llena (5)
Cantidad	Entero (2)	Cualquier otro tipo de dato
Paciente	Objeto de tipo IPaciente que se encuentre en atención	Objeto de tipo IPaciente que se encuentre en atención

	(3)	(6)
Médico	IMedico (4)	Cualquier otro tipo de dato

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	Un objeto de tipo IMedico, 1, un objeto de tipo IPaciente que se encuentre en atencion y un objeto de tipo IHabitacion no llena	Se le agrega una prestación al paciente enviado por parámetro	(1), (2), (3), (4)	Se le agrega una prestación al paciente enviado por parámetro
Clases incorrectas	Un objeto de tipo IMedico, 1, un objeto de tipo IPaciente que no se encuentre en atención y un objeto de tipo IHabitacion	Lanza excepción NoFueLlamadoException	(6)	Lanza excepción NoFueLlamadoException
	Un objeto de tipo IMedico, 1, un objeto de tipo IPaciente que se encuentre en atención y un objeto de tipo IHabitacion llena	Lanza excepción NoHayEspacioException	(5)	Lanza excepción NoHayEspacioException

Método elimina prestaciones

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
Paciente	Objeto de tipo IPaciente (1)	Cualquier otro tipo de dato

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas

Clases correctas	Un objeto de tipo IPaciente	Se le eliminan todas las prestaciones que poseía	(1)	Se le eliminan todas las prestaciones que poseía

Método llamar paciente

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
Paciente	Objeto de tipo IPaciente en espera (1)	Sala de Espera vacía (2)

Escenario con pacientes

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas		Se lo agrega a la lista de atencion eliminando de la lista de espera y del lugar donde estaban esperando	(1)	Se lo agrega a la lista de atencion eliminando de la lista de espera y del lugar donde estaban esperando

Escenario sin pacientes

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases incorrectas		Se lanza una excepción de tipo NoHayPacienteEsperandoException		Se lanza una excepción de tipo NoHayPacienteEsperandoException

Método cuenta ocurrencias

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
Paciente	Objeto de tipo IPaciente (1)	Cualquier otro tipo de dato
Médico	Objeto de tipo IMedico (2)	Cualquier otro tipo de dato

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	Un objeto de tipo IPaciente y uno de tipo IMedico	La cantidad de prestaciones que brindó el médico a ese paciente	(1), (2)	La cantidad de prestaciones que brindó el médico a ese paciente

Método agrega paciente histórico

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
Paciente	Objeto de tipo IPaciente que no se encuentra en el registro (1)	Objeto de tipo IPaciente que ya se encuentra en el registro (2)

Escenario con pacientes

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	Un objeto de tipo IPaciente que no se encuentra en el registro	Es agregado al registro histórico	(1)	Es agregado al registro histórico
Clases	Un objeto de tipo	Lanza excepción	(2)	Lanza excepción

incorrectas	IPaciente que ya se encuentra en el registro	PacienteRepetido Exception		PacienteRepetid oException

Escenario sin pacientes

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
Paciente	Objeto de tipo IPaciente que no se encuentra en el registro (1)	Objeto de tipo IPaciente que ya se encuentra en el registro

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	Un objeto de tipo IPaciente que no se encuentra en el registro	Es agregado al registro histórico	(1)	Es agregado al registro histórico

Método elimina histórico

Escenario con datos

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
Paciente	Objeto de tipo IPaciente que se encuentra en el registro y no debe pagar facturas (1)	Objeto de tipo IPaciente que se encuentra en el registro y debe pagar facturas (2)

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas

Clases correctas	Objeto de tipo IPaciente que se encuentra en el registro y no debe pagar facturas	Se borra de la lista de los registros historicos	(1)	Se borra de la lista de los registros historicos
Clases incorrectas	Un objeto de tipo IPaciente que se encuentra en el registro pero debe facturas	Lanza excepción JugoRobinhoException	(2)	Lanza excepción JugoRobinhoException

Método agregar a lista de espera

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
Paciente	Objeto de tipo IPaciente que no se encuentra ingresado (1)	Objeto de tipo IPaciente que ya se encuentra ingresado (2)

Escenario con pacientes

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	Un objeto de tipo IPaciente que no se encuentra ingresado	Es agregado a la lista de espera	(1)	Es agregado a la lista de espera
Clases incorrectas	Un objeto de tipo IPaciente que ya se encuentra ingresado	Lanza excepción PacienteYaIngresadoException	(2)	Lanza excepción PacienteYaIngresadoException

Escenario sin pacientes

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
Paciente	Objeto de tipo IPaciente que no se encuentra ingresado	Objeto de tipo IPaciente que ya se encuentra ingresado

	(1)	

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	Un objeto de tipo IPaciente que no se encuentra ingresado	Es agregado a la lista de espera	(1)	Es agregado a la lista de espera

Método hay prestación

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
Paciente	Un objeto de tipo IPaciente (1)	

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	Un objeto de tipo IPaciente que posee líneas de factura	true	(1)	true

Método ingresa nuevamente

Tabla de partición		
Condición	Clases correctas	Clases incorrectas
Paciente	Un objeto de tipo IPaciente que no se encuentra en las salas de espera (1)	Un objeto de tipo IPaciente que se encuentra en las salas de espera (2)

Batería de pruebas				
	Entradas	Salidas esperadas	Clases Cubiertas	Salidas obtenidas
Clases correctas	Un objeto de tipo IPaciente que no se encuentra en las salas de espera	El paciente es reingresado a la línea de espera	(1)	El paciente es reingresado a la línea de espera
Clases incorrectas	Un objeto de tipo IPaciente que se encuentra en las salas de espera	Excepcion de tipo PacienteYaIngresadoException	(2)	Excepcion de tipo PacienteYaIngresadoException

Caja Blanca

Se presenta el código correspondiente al método para calcular el importe adicional a la factura, cabe destacar que se debieron agregar parámetros al método por la forma en la que estaba tratado el tema de las facturas, por lo que se debieron agregar datos extras al método. Esto se destaca y se aclara por el hecho de que la complejidad climática es menor al tener directamente ciertos datos como parámetros, sin embargo fue la mejor solución posible al problema externo encontrado.

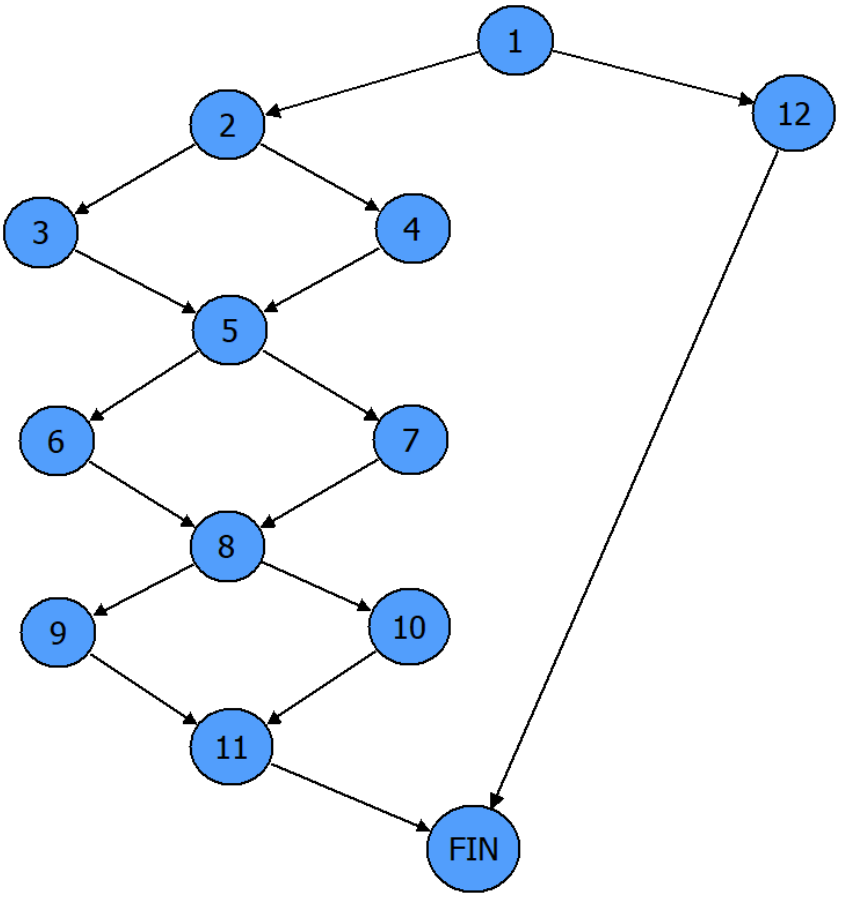
```
double importeParcial = 0;
double importeTotal = 0;
double respuesta = 0;
double A = 0.8, B = 0.4, C = 1.5, D = 0.9;
int diaDeLaFechaDeFacturacion = fechaDeFacturacion.get(Calendar.DAY_OF_WEEK);
Random mock = Mockito.mock(Random.class);
if(totalFacturado == 1000.0)
    when(mock.nextInt(30)).thenReturn(diaDeLaFechaDeFacturacion);
else
    when(mock.nextInt(30)).thenReturn(-1);
int aleatorio = mock.nextInt(30);

if (numeroDeFactura <= this.nroFactura) {  // 1
    long t1 = fechaDeSolicitud.getTimeInMillis();
    long t2 = fechaDeFacturacion.getTimeInMillis();

    if ((Math.abs(t2 - t1) / 1000.0) / (3600 * 24) < 10) {  //2
        importeParcial = totalFacturado - (subTotalImpar * A);  //3
    } else {
        importeParcial = totalFacturado * B; //4
    }
    if (paciente.getRangoEtareo().equalsIgnoreCase("Mayor")) { //5
        importeTotal = importeParcial * C; //6
    } else {
        importeTotal = importeParcial * D; //7
    }
    if (aleatorio == diaDeLaFechaDeFacturacion) {  //8
        respuesta = importeTotal; //9
    }else
        respuesta = importeTotal + sumaValores(listaDeInsumos); //10
    return respuesta; //11

} else
    return importeParcial; //12
```

Grafo Ciclomático



Aclaración: número de nodo = número de línea

Cálculo de la complejidad ciclomática

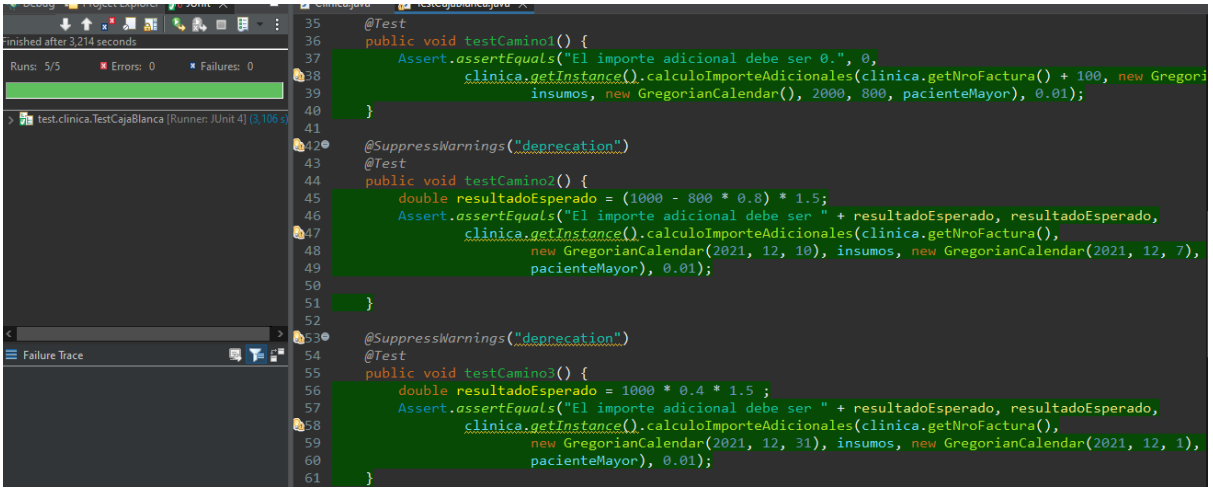
- Número de regiones cerradas + 1 = 5
- Número de arcos - Número de nodos + 2 = 16 - 13 + 2 = 5
- Número de nodos condición + 1 = 4 + 1 = 5

Conjunto de caminos básicos independientes

1) 1 - 12 - FIN
2) 1 - 2 - 3 - 5 - 6 - 8 - 9 - 11 - FIN
3) 1 - 2 - 4 - 5 - 6 - 8 -9 - 11 - FIN
4) 1 - 2 - 4 - 5 - 7 - 8 - 9 - 11 - FIN
5) 1 - 2 - 4 - 5 - 7 - 8 - 10 - 11 - FIN

Nº de camino	Parámetros de entrada	Resultado esperado

1	numeroDeFactura inexistente	0
2	numeroDeFactura existente fecha de solicitud - fecha de facturación < 10 Rango Etáreo de paciente = Mayor aleatorio = día de la fecha de facturación	(totalFacturado - (subTotalImpar * A)) * C
3	numeroDeFactura existente fecha de solicitud - fecha de facturación > 10 Rango Etáreo de paciente = Mayor aleatorio = día de la fecha de facturación	totalFacturado * B * C
4	numeroDeFactura existente fecha de solicitud - fecha de facturación > 10 Rango Etáreo de paciente ≠ Mayor aleatorio = día de la fecha de facturación	totalFacturado * B * D
5	numeroDeFactura existente fecha de solicitud - fecha de facturación > 10 Rango Etáreo de paciente ≠ Mayor aleatorio ≠ día de la fecha de facturación	totalFacturado * B * D + suma de valores de la lista de insumos




```
35      @Test
36      public void testCamino1() {
37          Assert.assertEquals("El importe adicional debe ser 0.", 0,
38                  clinica.getInstance().calculoImporteAdicionales(clinica.getNroFactura() + 100, new Gregori
39                          insumos, new GregorianCalendar(), 2000, 800, pacienteMayor), 0.01);
40      }
41
42      @SuppressWarnings("deprecation")
43      @Test
44      public void testCamino2() {
45          double resultadoEsperado = (1000 - 800 * 0.8) * 1.5;
46          Assert.assertEquals("El importe adicional debe ser " + resultadoEsperado, resultadoEsperado,
47                  clinica.getInstance().calculoImporteAdicionales(clinica.getNroFactura(),
48                          new GregorianCalendar(2021, 12, 10), insumos, new GregorianCalendar(2021, 12, 7),
49                          pacienteMayor), 0.01);
50
51      }
52
53      @SuppressWarnings("deprecation")
54      @Test
55      public void testCamino3() {
56          double resultadoEsperado = 1000 * 0.4 * 1.5 ;
57          Assert.assertEquals("El importe adicional debe ser " + resultadoEsperado, resultadoEsperado,
58                  clinica.getInstance().calculoImporteAdicionales(clinica.getNroFactura(),
59                          new GregorianCalendar(2021, 12, 31), insumos, new GregorianCalendar(2021, 12, 1),
60                          pacienteMayor), 0.01);
61      }
```

```java
@SuppressWarnings("deprecation")
@Test
public void testCamino4() {
    double resultadoEsperado = 1000 * 0.4 * 0.9;
    Assert.assertEquals("El importe adicional debe ser " + resultadoEsperado, resultadoEsperado,
            clinica.getInstance().calculoImporteAdicionales(clinica.getNroFactura(),
                    new GregorianCalendar(2021, 12, 31), insumos, new GregorianCalendar(2021, 12, 1),
                    pacienteJoven), 0.01);
}

@SuppressWarnings("deprecation")
@Test
public void testCamino5() {
    double resultadoEsperado = 2000 * 0.4 * 0.9 + 100;
    Assert.assertEquals("El importe adicional debe ser " + resultadoEsperado, resultadoEsperado,
            clinica.getInstance().calculoImporteAdicionales(clinica.getNroFactura(),
                    new GregorianCalendar(2021, 12, 31), insumos, new GregorianCalendar(2021, 12, 1),
                    pacienteJoven), 0.01);
}
```

```java
    double subTotalImpar, IPaciente paciente) {
double importeParcial = 0;
double importeTotal = 0;
double respuesta = 0;
double A = 0.8, B = 0.4, C = 1.5, D = 0.9;
int diaDeLaFechaDeFacturacion = fechaDeFacturacion.get(Calendar.DAY_OF_WEEK);
Random mock = Mockito.mock(Random.class);
if(totalFacturado == 1000.0)
    when(mock.nextInt(30)).thenReturn(diaDeLaFechaDeFacturacion);
else
    when(mock.nextInt(30)).thenReturn(-1);
int aleatorio = mock.nextInt(30);

if (numeroDeFactura <= this.nroFactura) {
    long t1 = fechaDeSolicitud.getTimeInMillis();
    long t2 = fechaDeFacturacion.getTimeInMillis();

    if ((Math.abs(t2 - t1) / 1000.0) / (3600 * 24) < 10) {
        importeParcial = totalFacturado - (subTotalImpar * A);
    } else {
        importeParcial = totalFacturado * B;
    }
    if (paciente.getRangoEtareo().equalsIgnoreCase("Mayor")) {
        importeTotal = importeParcial * C;
    } else {
        importeTotal = importeParcial * D;
    }
    if (aleatorio == diaDeLaFechaDeFacturacion) {
        respuesta = importeTotal;
    }else
        respuesta = importeTotal + sumaValores(listaDeInsumos);
    return respuesta;

} else
    return importeParcial;
```

Test de GUI

Para realizar el test de GUI se utilizó la clase Robot de java que tiene como propósito la automatización de pruebas y actividades que involucren el mouse o el teclado sin la presencia o interacción física del usuario.
Con el fin de poder generar distintos escenarios (clínica con y sin pacientes) se utilizó un mock de la clase controlador y en este mock se generó un nuevo constructor con dos parámetros extra (archivo de entrada, archivo de salida) para poder cargar distintas suites de datos.
Debido a la complejidad de la interfaz gráfica sólo se testearon casos relacionados con el agregado y la eliminación de pacientes.

Se testearon 2 escenarios distintos.

- Escenario 1: Clínica sin Pacientes.
- Escenario 2: Clínica con Pacientes.

Escenario 1

Para emular un escenario sin pacientes, en el método @Before setUp() se cargó un archivo binario de persistencia vacío (se realizó una copia para preservar el archivo original) mediante el mock del controlador. Y en el método @After tearDown() se eliminó la copia creada para descartar todos los posibles cambios sufridos durante la ejecución del test.

Para el escenario vacío se realizaron los siguientes test:

- agrega1PacienteOk()
- agrega3PacientesOk()
- agregaPacienteRepetido()

Test agrega1PacienteOk()

Para que este test tenga un resultado favorable se debe cumplir que la cantidad de elementos en la lista de pacientes luego de agregar un paciente tenga su longitud incrementada en 1 unidad.

```
Assert.assertEquals("Deberia haber un paciente mas en la lista",
(cantidadAntes+1), cantidadDespues);
```

Test agrega3PacientesOk()

Para que este test tenga un resultado favorable se debe cumplir que la cantidad de elementos en la lista al finalizar el test sea 3 (arranca sin pacientes y agrega 3).

```
Assert.assertEquals("Deberia haber tres pacientes en la lista", 3,
pacientes.getModel().getSize());
```

Test agregaPacienteRepetido()

Cuando se agrega un paciente repetido (misma historia clínica y mismo DNI) se lanza una excepción que despliega una ventana emergente (JOptionPane) que envía el mensaje “PACIENTE REPETIDO”.

Para poder testear el correcto despliegue del mensaje se utiliza una clase llamada FalsoOptionPane que captura el mensaje de la excepción y mediante un getter es posible recuperarlo.

Para que el test resulte favorable se debe cumplir que el mensaje capturado coincida con el mensaje que teóricamente se envía cuando hay un paciente repetido.

```
agregaUnPaciente("Pedrito Ramirez", "223 1563322", "33456888",
"11", "Elm Street 666", "Springwood", "mayor");

agregaUnPaciente("Pedrito Ramirez", "223 1563322", "33456888",
"11", "Elm Street 666", "Springwood", "mayor");

Assert.assertEquals("Mensaje incorrecto, deberia decir" +
Mensajes.Error_Paciente_Repetido.getValor(),
Mensajes.Error_Paciente_Repetido.getValor(), op.getMensaje());
```

Escenario 2

Para emular un escenario con pacientes, en el método @Before setUp() se cargó un archivo binario de persistencia con datos (se realizó una copia para preservar el archivo original) mediante el mock del controlador. Y en el método @After tearDown() se eliminó la copia creada para descartar todos los posibles cambios sufridos durante la ejecución del test.

Para el escenario con datos se realizaron los siguientes test:

- testEliminaOKUnoSolo()
- testEliminaOKTodos()
- testEliminaTodosMasUnoMal()

testEliminaOkUnoSolo()

Para que este test resulte favorable se debe cumplir que la cantidad de elementos luego de la eliminación difiera en uno con respecto a la cantidad de elementos al inicio de la ejecución del test.

```
Assert.assertEquals("La lista deberia tener un elemento menos", numeroDeElementosDespues, (numeroDeElementos-1));
```

testEliminaOKTodos()

Para que este test resulte favorable se debe cumplir que la cantidad de elementos al finalizar el test sea 0.

```
Assert.assertEquals("La lista deberia quedar vacia", 0, pacientes.getModel().getSize());
```

testEliminaTodosMasUnoMal()

Para que este test resulte favorable se deben eliminar, en primer lugar todos los pacientes de la lista y posteriormente, intentar eliminar uno más. Al intentar eliminar un paciente inexistente se debe lanzar una excepción de IndexOutOfBoundsException().

Un test correcto siempre llegará al catch que captura la excepción mencionada.

```
} catch (IndexOutOfBoundsException e)  {

          Assert.assertTrue("Siempre deberia saltar la excepcion al intentar borrar mas de la cantidad disponible", true);
}
```

Test de Persistencia

test lanzaExcpecionAlIntentarLeerArchivoInexistenteTest

Verifica el lanzamiento de una excepción de tipo IOException al intentar leer de un archivo inexistente.

test crearArchivoConExitoTest

Verifica la creación exitosa de un archivo nuevo, sin lanzamiento de excepciones.

test persisteClincaConPacientesTest

Verifica la persistencia exitosa de la Clínica en su totalidad, contando con médicos y pacientes.

test persisteClincaVacia

Verifica la persistencia exitosa de una clínica vacía, sin pacientes ni médicos.

Dificultades con las que nos enfrentamos

Durante el desarrollo de este trabajo práctico nos enfrentamos a varias dificultades que, en términos generales, supimos sortear.

En primer lugar, nos enfrentamos con la dificultad de tener clases y métodos que incluían la letra ñ. Debido a este inconveniente tuvimos que optar por utilizar el IDE Eclipse ya que en Intellij arrojaba errores. Incluso, en algunas ocasiones luego de hacer un pull desde GitHub se generaban nuevos errores relacionados al uso de este tipo de caracteres no presentes en el idioma inglés.

En segundo lugar nos enfrentamos a un gran problema relacionado con la falta documentación de la aplicación. Al no estar los contratos correctamente definidos (en muchos casos no estaban siquiera definidos) se hizo más dificultoso poder testear los métodos y los constructores. Gracias a estas dificultades a las que nos enfrentamos nos dimos cuenta lo vital que es, en primer lugar, programar una aplicación pensando en el futuro testeo, en segundo lugar lo imprescindible que es tener una correcta documentación para logar un testeo ameno y ágil y en tercer lugar lo importante que son las etapas de testeo estático (no realizadas en este trabajo práctico), con la presencia de las programadores para clarificar las dudas que pudieran surgir durante esta etapa previa.

En tercer lugar nos enfrentamos a una dificultad que pudimos sortear parcialmente y es la concerniente a la persistencia de la información. No hay métodos preparados para poder agregar y quitar de manera sencilla (sin tener que utilizar la GUI) pacientes y médicos. Para poder testear correctamente la persistencia tuvimos que modificar/agregar algunos métodos. No sabemos si es correcto o no tomarnos estas atribuciones de refactorizar parte del código recibido para testear, pero a fin de cumplimentar este requisito del TP, lo hicimos.

Por último, notamos algunos comportamientos erráticos durante el uso de la interfaz gráfica. Decimos erráticos, porque no son fácilmente reproducibles y debido a la falta de documentación no podemos aislar el problema. En algunas ocasiones, al intentar eliminar un paciente, salta una ventana de error que informa que no se puede eliminar el paciente hasta que haya liquidado las prestaciones. Cuando presionamos el botón de “facturar”, nos informa que no es posible facturar debido a que el paciente no ha recibido ninguna prestación.

Conclusiones

Durante la realización de este trabajo práctico nos enfrentamos a muchos desafíos y fue realmente gratificante poder sortearlos. Aprendimos a realizar distintos tipos de pruebas

sobre un código totalmente ajeno a nosotros. Tuvimos largas charlas, pensando las mejores estrategias (de las vistas durante la cursada) que podíamos aplicar para lograr los objetivos. Algo muy positivo es que mejoramos notablemente nuestro uso de GitHub.

Por último, no queremos dejar de mencionar que enfrentarnos a todos estos problemas también nos hizo pensar en el código que nosotros entregamos al otro grupo de test. Creemos que entregamos un código (bastante) apto para el testing, pero sería muy interesante escuchar las devoluciones de los chicos del grupo 3 para poder mejorar aún más.