Universidad Nacional de Córdoba

Facultad de Ciencias Exactas, Física y Naturales

Diseño del Sistema

Docentes:
Miceli, Martín.
Bustos, Martin

Grupo:”Drink Team”

Integrantes:

- Depetris, Stefano
- Gauna, Macarena del Sol
- Nieto, Marcos

Diseño del sistema

Versión	Fecha	Resumen de cambio	Autor
1.0	19/06/2020		• Depetris, Stefano • Gauna, Macarena • Nieto, Marcos

INTRODUCCIÓN

En el siguiente informe detallaremos los patrones de diseño utilizados, acompañado con diagramas de paquetes, clase, objetos y secuencia que ayudarán a describir el diseño de nuestro proyecto.También se detallarán las pruebas de test unitarias.

DESARROLLO

Diagrama de Paquetes:

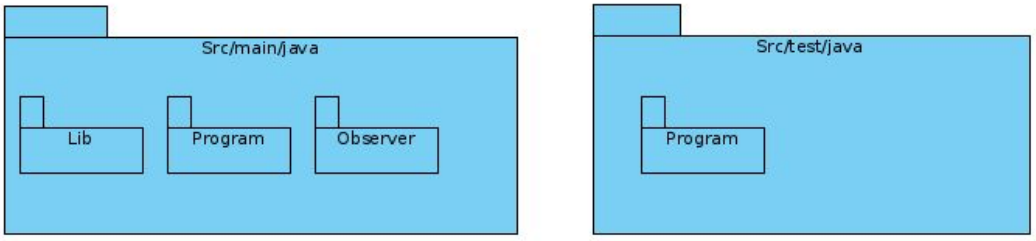


Diagrama de Clases

Este es un diagrama de clases simplificado que muestra el funcionamiento del proyecto.

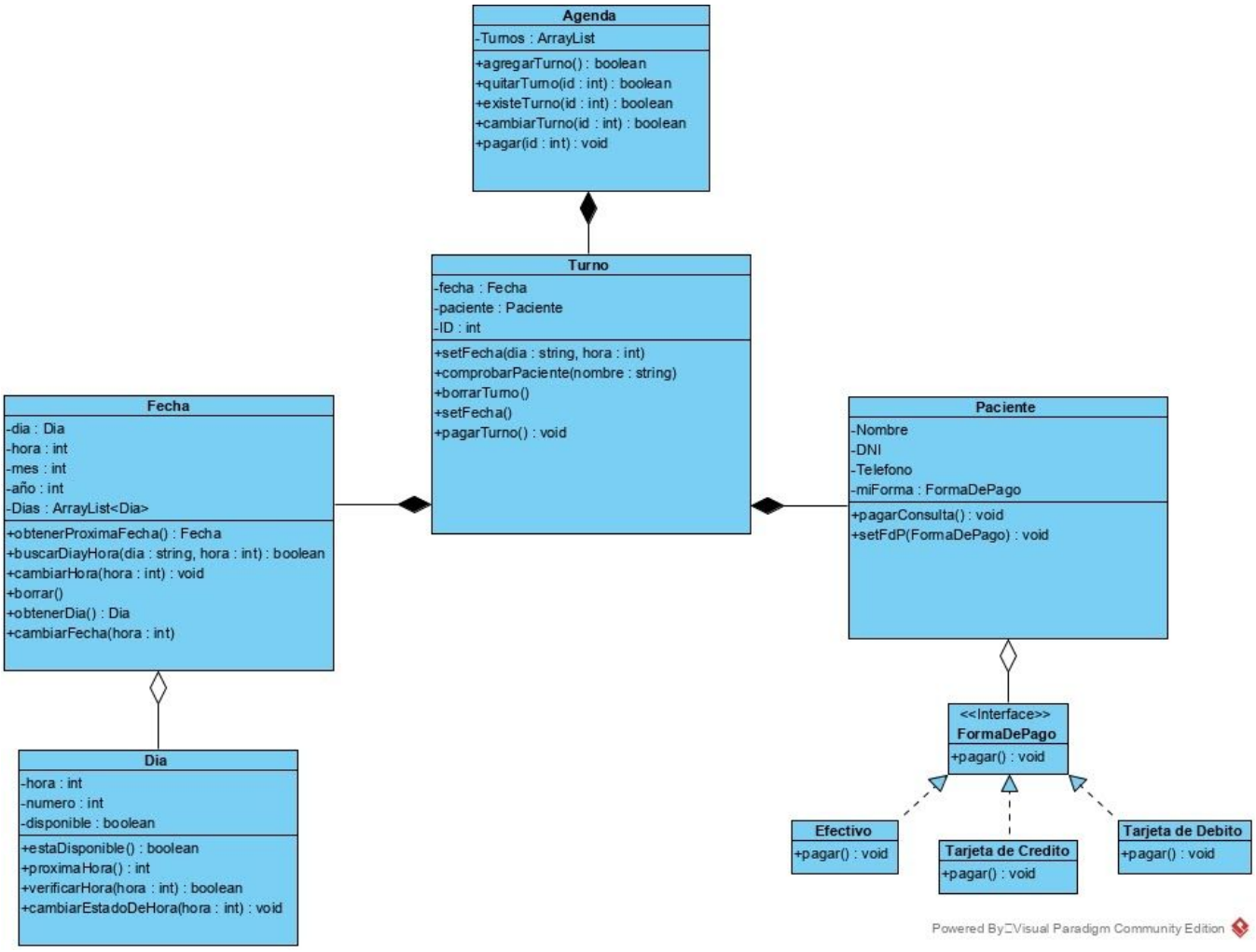


Diagrama de Objetos:

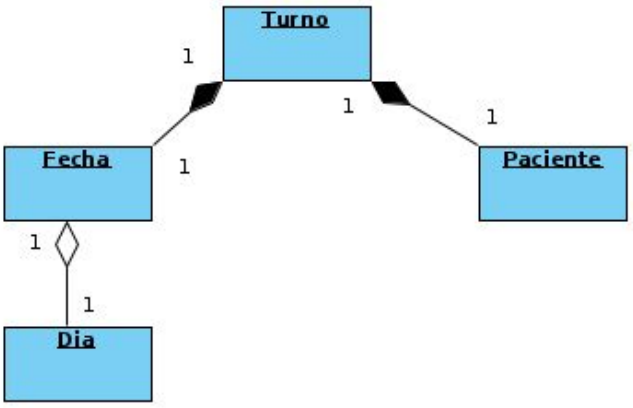
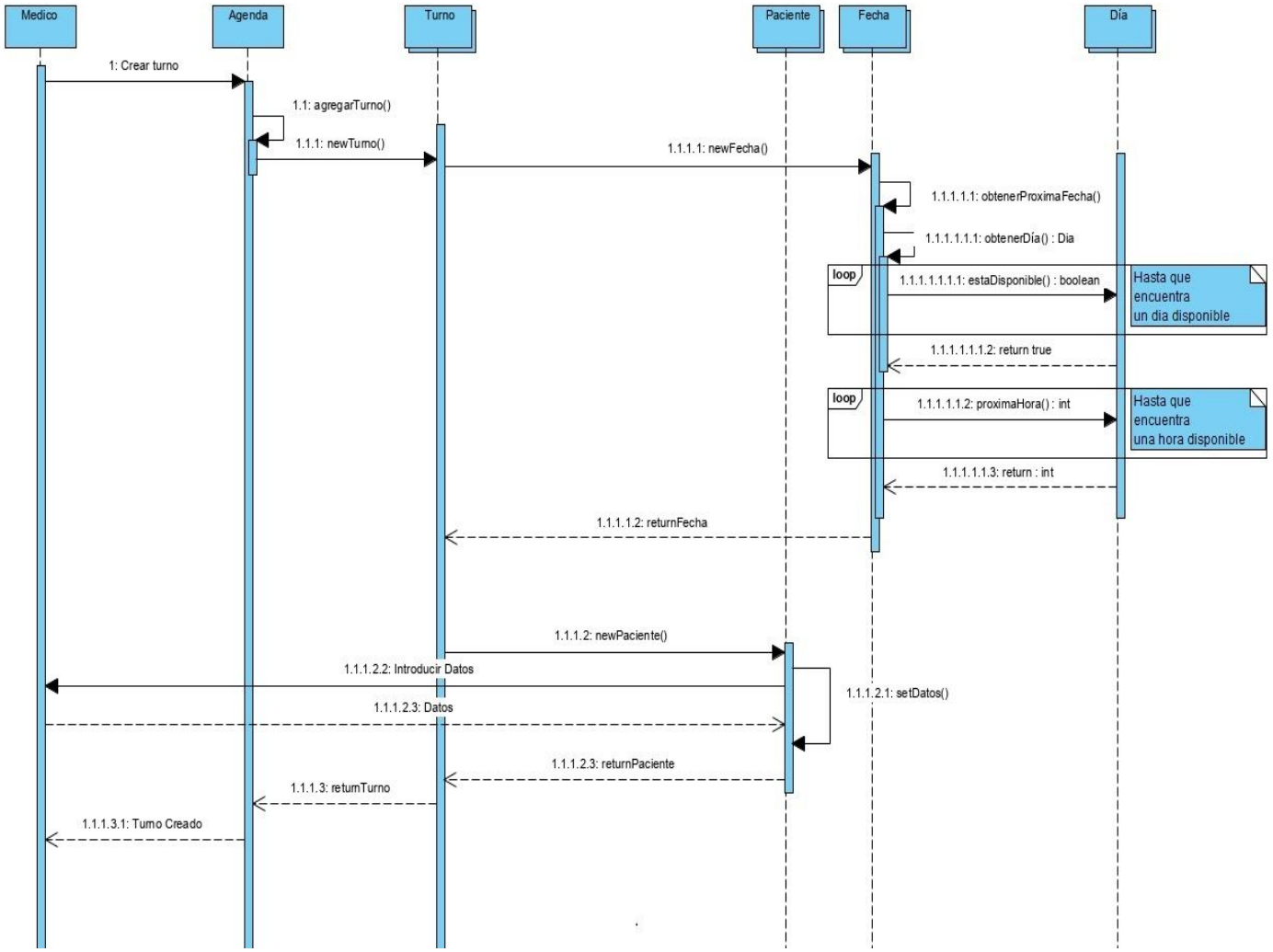
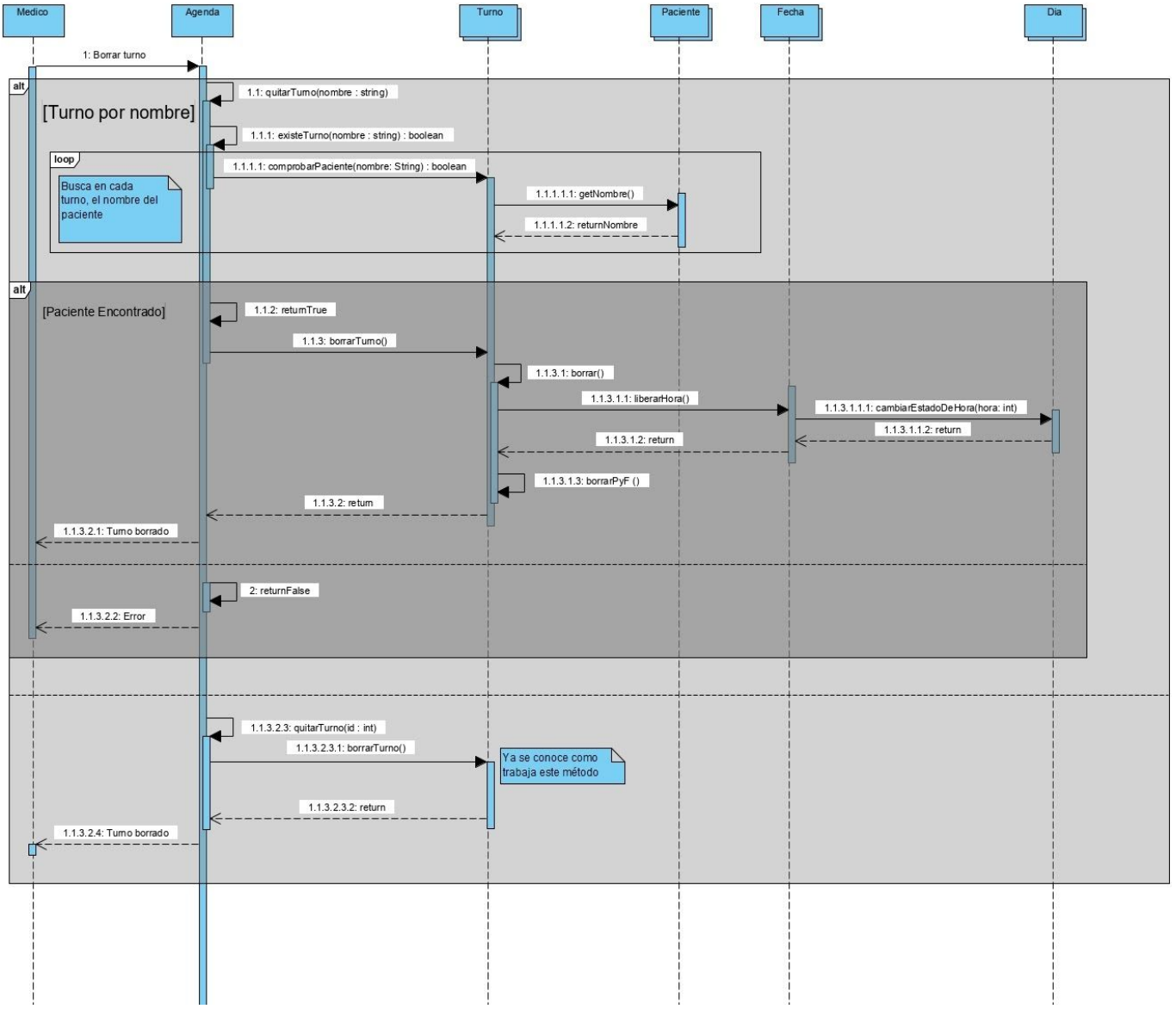
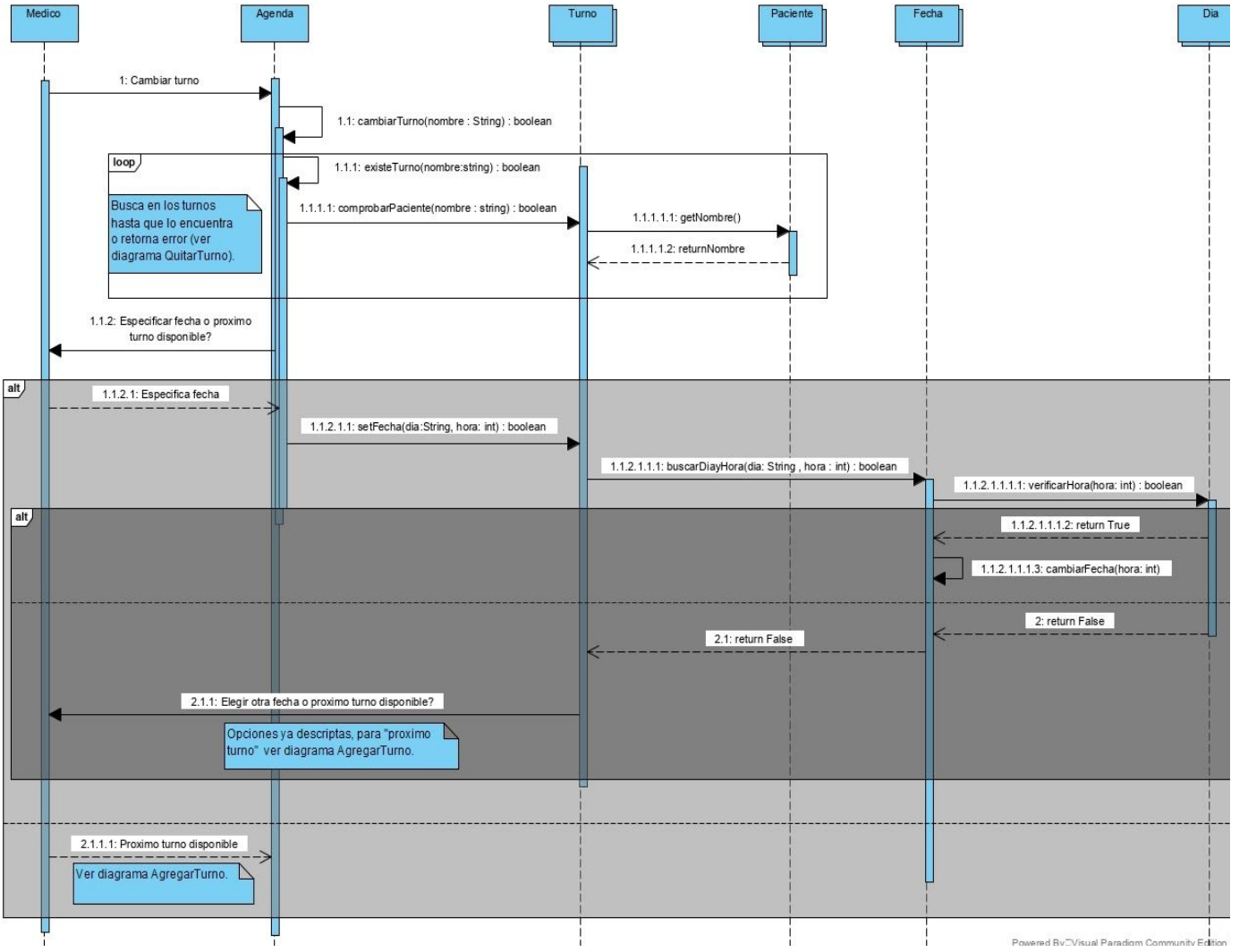


Diagrama de Secuencia:

A continuación se presentan algunos diagramas de secuencias que muestran algunas funcionalidades del programa.




Medico
Agenda
Turno
Paciente
Fecha
Dia
1: Borrar turno
alt
[Turno por nombre]
1.1: quitarTurno(nombre : string)
1.1.1: existeTurno(nombre : string) : boolean
loop
Busca en cada turno, el nombre del paciente
1.1.1.1: comprobarPaciente(nombre: String) : boolean
1.1.1.1.1: getNombre()
1.1.1.1.2: returnNombre
alt
[Paciente Encontrado]
1.1.2: returnTrue
1.1.3: borrarTurno()
1.1.3.1: borrar()
1.1.3.1.1: liberarHora()
1.1.3.1.1.1: cambiarEstadoDeHora(hora: int)
1.1.3.1.1.2: return
1.1.3.1.2: return
1.1.3.1.3: borrarPyF ()
1.1.3.2: return
1.1.3.2.1: Turno borrado
2: returnFalse
1.1.3.2.2: Error
1.1.3.2.3: quitarTurno(id : int)
1.1.3.2.3.1: borrarTurno()
Ya se conoce como trabaja este método
1.1.3.2.3.2: return
1.1.3.2.4: Turno borrado


Medico
Agenda
Turno
Paciente
Fecha
Dia
1: Cambiar turno
1.1: cambiarTurno(nombre : String) : boolean
loop
1.1.1: existeTurno(nombre:string) : boolean
Busca en los turnos hasta que lo encuentra o retorna error (ver diagrama QuitarTurno).
1.1.1.1: comprobarPaciente(nombre : string) : boolean
1.1.1.1.1: getNombre()
1.1.1.1.2: returnNombre
1.1.2: Especificar fecha o proximo turno disponible?
alt
1.1.2.1: Especifica fecha
1.1.2.1.1: setFecha(dia:String, hora: int) : boolean
1.1.2.1.1.1: buscarDiayHora(dia: String , hora : int) : boolean
1.1.2.1.1.1.1: verificarHora(hora: int) : boolean
alt
1.1.2.1.1.1.2: return True
1.1.2.1.1.1.3: cambiarFecha(hora: int)
2: return False
2.1: return False
2.1.1: Elegir otra fecha o proximo turno disponible?
Opciones ya descriptas, para "proximo turno" ver diagrama AgregarTurno.
2.1.1.1: Proximo turno disponible
Ver diagrama AgregarTurno.
Powered By Visual Paradigm Community Edition

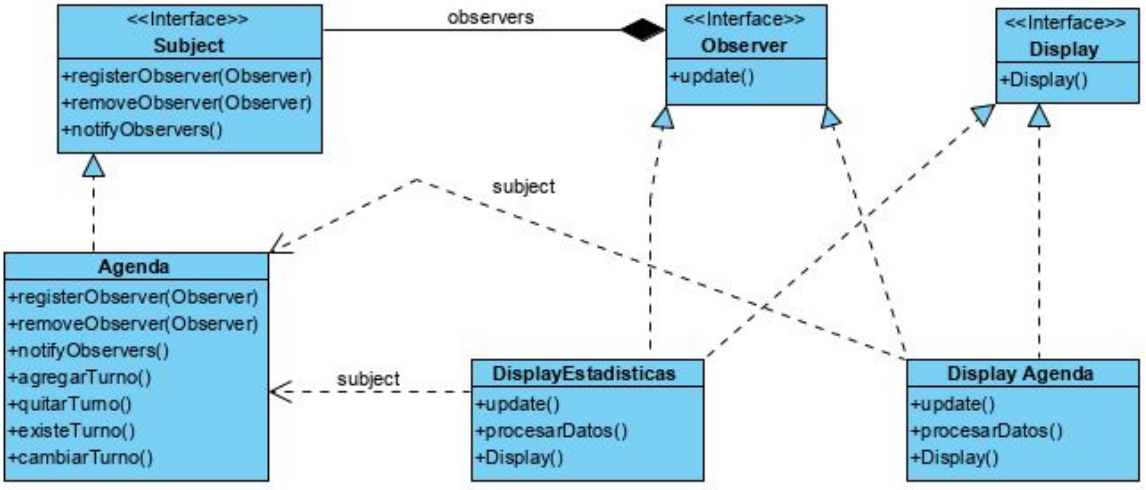
PATRONES DE DISEÑO

Observer

Se implementará el patrón de diseño Observer para poder actualizar la vista de la agenda y también para mostrar una vista de estadísticas de turnos. De esta manera se evita estar preguntando constantemente el estado de los turnos a la clase Agenda. También evitamos actualizar las estadísticas constantemente en cada cambio de estado de un turno, de manera que solo actualizaremos las estadísticas cuando el cambio de estado de un turno sea pertinente a la estadística.

Tendremos dos observadores que se suscribirán al sujeto observable, éstas serán las vistas en la interfaz gráfica de nuestro proyecto. Los observadores se actualizarán cuando el sujeto observable les indique que hubo algún cambio en el mismo. Entonces por ejemplo, cuando el usuario agrega un turno se notificará a los observadores este cambio y, de ser necesario, se actualizará los gráficos en la vista de la Agenda. También se actualizarán las estadísticas de turnos totales agendados.

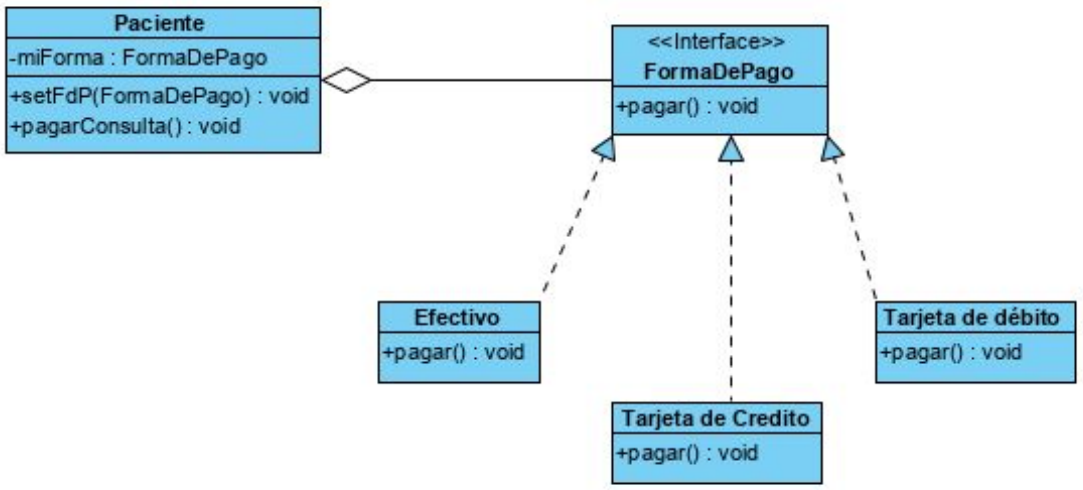
A continuación se adjunta la figura de lo nombrado anteriormente.



Strategy

El patrón Strategy es un patrón de diseño para el desarrollo de software. Se clasifica como patrón de comportamiento porque determina cómo se debe realizar el intercambio de mensajes entre diferentes objetos para resolver una tarea. El mismo permite mantener un conjunto de algoritmos de entre los cuales el software puede elegir aquel que le convenga o se solicite e intercambiarlo dinámicamente según sus necesidades en tiempo de ejecución.

En nuestro proyecto este patrón es utilizado para elegir la forma de pago que se provee a los pacientes. De esta manera un usuario podría cambiar en tiempo de ejecución el método a pagar según lo solicitado por cada cliente. A continuación un diagrama de clases simplificado donde se muestra el patrón aplicado.



PRUEBAS UNITARIAS

Las siguientes pruebas unitarias serán denotadas por un código de 3 caracteres (<AAN>) para poder identificarlas con mayor facilidad posteriormente. En cada prueba existirán 4 apartados; **Componentes** del código evaluados en la prueba, **pre-requisitos** necesarios antes de ejecutarla, datos de **entrada** necesarios y **salida** esperada de la prueba.

PU1: Agregar un turno.

- **Clases involucradas:**
  Agenda
- **Pre-requisitos:** -
- **Entrada:**
  Agenda.agregarTurno()
- **Salida esperada:**
  El método pedirá los datos del paciente. Una vez introducido los datos del paciente, arrojará "Turno creado con éxito para el día <Dia del turno> del <Mes del turno> a las <Hora del turno>"

PU2: Quitar un turno.

- **Clases involucradas:**
  Agenda
- **Pre-requisitos:**
  Tiene que haberse creado un turno.
- **Entrada:**
  Agenda.borrarTurno(<DNI del paciente>)
- **Salida esperada**:
  El método arrojará “Turno quitado con éxito”.

PU3: Cambiar turno.

- **Clases involucradas:**
  Agenda
- **Pre-requisitos:**
  Tiene que haberse creado un turno.
- **Entrada:**
  Agenda.cambiarTurno(<DNI del paciente>)
- **Salida esperada:**
  El método arrojará “Turno cambiado con éxito para el día <Dia del turno> del <Mes del turno> a las <Hora del turno>”

PU4: Utilizar medio de pago efectivo.

- **Clases involucradas:**
  Agenda
- **Pre-requisitos:**
  Tiene que haberse creado un turno
- **Entrada:**
  Agenda.pagar(<nombreDelPaciente>, 0)
- **Salida esperada:**
  El método arrojará “Pago efectuado con efectivo”.

PU5: Utilizar medio de pago tarjeta debito.

- **Clases involucradas:**
  Agenda
- **Pre-requisitos:**
  Tiene que haberse creado un turno
- **Entrada:**
  Agenda.pagar(<nombreDelPaciente>, 1)

- **Salida esperada:**
  El método arrojará “Pago efectuado con Débito”.

PU6: Utilizar medio de pago tarjeta credito.

- **Clases involucradas:**
  Agenda, Turno, Paciente, TarjetaDeCredito
- **Pre-requisitos:**
  Tiene que haberse creado un turno.
- **Entrada:**
  Agenda.pagar(<nombreDelPaciente>, 2)
- **Salida esperada:**
  El método arrojará “Pago efectuado con Crédito”.

PU7: Agregar un turno cuando el horario ya está ocupado.

- **Clases involucradas:**
  Agenda, Turno, Paciente, Fecha, Dia
- **Pre requisitos:**
  Que exista un turno previamente.
- **Entrada:**
  Agenda.agregarTurno(<Dia del turno existente>,<Mes del turno existente>, <Hora del turno existente>)

- **Salida Esperada:**
  Una excepción la cual arrojará “No se agregó el turno porque ya existe uno en el horario solicitado.”

PU8: Eliminar un turno inexistente.

- **Clases involucradas:**
  Agenda, Turno, Paciente
- **Pre requisitos: -**

- **Entrada:**
  Agenda.eliminarTurno(<DNI del Paciente>)

- **Salida Esperada:**
  Una excepción la cual arrojará “Error. No existe el turno. Paciente no encontrado.”

PU9: Cambiar un turno inexistente.

- **Clases involucradas:**
  Agenda, Turno
- **Pre requisitos:** -

- **Entrada:**
  Agenda.cambiarTurno(<DNI del paciente>)

- **Salida Esperada:**
  Una excepción la cual arrojará “Error. No existe el turno. Paciente no encontrado.”

PU10: Introducir datos de fecha inválidos.

- **Clases involucradas:**
  Fecha
- **Pre requisitos:**
  Crear una instancia de Fecha
- **Entrada:**
  Fecha.setCampos(<Dia no valido>, <Mes no valido>, <Hora no valida>)
- **Salida Esperada:**
  Una excepción la cual arrojará “Error. Fecha no válida.”

PU11: Introducir datos de paciente inválidos.

- **Clases involucradas:**
  Paciente
- **Pre requisitos:**
  Crear una instancia Paciente.
- **Entrada:**
  Paciente.setDatos(<DNI no valido>, <nombre>, <telefono>);
- **Salida Esperada:**
  Una excepción la cual arrojará “Error. El DNI introducido no es válido."

MATRIZ DE TRAZABILIDAD

A continuación se encuentra la matriz de trazabilidad actualizada con las pruebas unitarias. Se adjuntará la misma también en formato excel.

Requerimientos/ Casos de uso y de prueba	Req_Fu_ 1	Req_Fu_ 2	Req_Fu_ 3	Req_Fu_ 4	Req_Fu_ 5	Req_Fu_ 6	Req_Fu_ 7	Req_Fu_ 8	Req_Fu_ 9	Req_Fu_ 10	Req_Fu_ 11	Req_Fu_ 12	Req_Fu_ 13	Req_NF_ 1	Req_NF_ 2	Req_NF_ 3	Req_NF_ 4	Req_NF_ 5	Req_NF_ 6
Crear turno			X			X		X	X										X
Anular turno				X	X			X	X										X
Ver turno	X						X		X										X
Cambiar turno					X	X		X	X										X
Ingresar al Sistema									X		X								X
Ver Estadisticas									X			X	X						X
Forma de Pagos									X	X									
Casoprueba_1	X								X										
Casoprueba_2		X	X						X										
Casoprueba_3			X			X			X			X							
Casoprueba_4			X	X			X		X			X							
Casoprueba_5			X		X				X										
Casoprueba_6			X			X			X										
Casoprueba_7		X					X		X										
Casoprueba_8								X	X										
Casoprueba_9	X		X	X	X	X	X	X	X	X	X	X	X				X	X	X
Casoprueba_10			X							X									
Casoprueba_11											X								
Casoprueba_12									X			X							
Casoprueba_13									X				X						
CasopruebaNf_1									X					X					
CasopruebaNf_2															X				
CasopruebaNf_3																X			
CasopruebaNf_4									X								X		
CasopruebaNf_5									X									X	
CasopruebaNf_6									X										X
CasopruebaAlt_ 1			X						X										
CasopruebaAlt_ 2			X						X										
CasopruebaAlt_ 3				X					X										
PU1			X																
PU2				X															
PU3					X														
PU4										X									
PU5										X									
PU6										X									
PU7			X			X													
PU8				X															
PU9					X														
PU10								X											
PU11								X											