TP Final: Testing Clínica

Taller de Programación I

Universidad Nacional de Mar del Plata
Facultad de Ingeniería

Alumnos

- D'ALU DE BONI, Lisandro
- CAZORLA MARTINEZ, Nicolás
- LONGARETTO, Lorenzo
- MALETTA, Augusto

Presentación del Informe

Alumnos - (Grupo 1) :

- CAZORLA MARTINEZ, Nicolas
- D'ALU, Lisandro
- LONGARETTO, Lorenzo
- MALETTA, Augusto

Asignatura: Taller de Programación 1

Repositorio Github:
https://github.com/LorenzoLongaretto/TallerProg1/tree/main/clinicaTestearTPFinal

Introducción

En el presente informe se detallarán los resultados obtenidos tras aplicar técnicas de testing en un sistema que simula el funcionamiento de una clínica.
El sistema de la clínica corresponde al desarrollado como trabajo final para la asignatura "Programación III", durante el primer cuatrimestre de 2021

En el trabajo se cubrirán los siguientes temas:

- **Pruebas de caja negra:** Basadas en verificar si el código cumple los contratos especificados, sin observar el código fuente que resuelve el problema
- **Pruebas de caja blanca:** Se analiza el código fuente, prestando atención a los posibles caminos que puede tomar el código por las estructuras de decisión.
- **Prueba de persistencia:** Se observará si los datos de la clínica se almacenan y se recuperan correctamente. En el caso de este trabajo se utiliza persistencia XML
- **Prueba de Interfaz Gráfica:** También llamado Test de GUI, consiste en revisar la parte gráfica del proyecto, específicamente en este trabajo se analiza el módulo de facturación de la clínica.
- **Prueba de Integración:** Diseñadas para probar la interacción entre los distintos componentes de un sistema.

Caja Negra

Como primera parte se testean todos los métodos de la clínica. En este caso se tomaron los dos escenarios posibles, uno con la clínica con datos y otro sin datos.

En ambos escenarios se pudo observar la falla en el ingreso de un paciente repetido, permitiendo el ingreso del mismo. Esto no coincide con la documentación.
Por otra parte también se testean algunas de las clases más importantes, que serían las clases de Paciente, Médico, Factura, Habitación Compartida, Sala de Espera Privada y los Factory de Médicos y Pacientes. En ninguna de estas clases se encontraron fallas, ni errores.

Dentro de las clases Paciente y Médico, se utilizó el Factory de ambos como constructor ya que al ser clases abstractas su constructor no se puede instanciar. Las demás clases no se testean ya que tenían un comportamiento similar a las testeadas, por ejemplo entre habitaciones y distintos tipos de médico y paciente, y se priorizo las más importantes.Las pruebas están en el paquete de testing "testCajaNegra".

A modo de resumen se muestra a continuación los escenarios y batería de pruebas para un método de cada una de las clases testeadas:

Clase: Clinica
Método: buscaPaciente

Escenarios

Escenario	Descripción
1	La colección de pacientes se encuentra vacía, sin pacientes
2	La colección de pacientes se encuentra con pacientes. Ej: Nro:1 Nombre: Lisandro DAlu

Tabla de particiones

Condición de entrada	Clases VÁLIDAS	Clases INVÁLIDAS
numeroPaciente	nro >0 (1)	nro<=0 (2)
Estado colección de pacientes	Existe el paciente (3)	No existe el paciente(4)

Batería de pruebas

Clase (Correcta o Incorrecta)	Valores de entrada	Salida esperada	Salida obtenida	Clases cubiertas
Correcta	Nro = 1 Escenario 2	Devuelve paciente nro 1 Nombre: "Lisandro DAlu"	Devuelve paciente nro 1 Nombre: "Lisandro DAlu" El metodo funciona correctamente	1,3
Correcta	Nro = 2 Escenario 2	Devuelve null	Devuelve null, el metodo funciona correctamente	1,4
Correcta	Nro =1 Escenario 1	Devuelve null	Devuelve null, el metodo funciona correctamente	1,4
Incorrecta	Nro = -1	Devuelve null	Funciona correctamente	2,4

Clase: Paciente
Método: Agregar consulta de un médico

Escenario	Descripción
1	La colección de médicos contiene médicos
2	No hay médicos en la colección

Tabla de particiones

Condición de entrada	Clases VÁLIDAS	Clases INVÁLIDAS
Paciente	objeto no nulo (1)	nulo (2)
Médico	no nulo (3)	nro de matrícula vacía o nula (4)

Batería de pruebas

Clase (Correcta o Incorrecta)	Valores de entrada	Salida esperada	Salida obtenida	Clases cubiertas
Correcta	Paciente válido con datos completos Médico existente, con matrícula válida Escenario 1	Se agrega una nueva consulta al paciente. El médico también la almacena para generar un reporte	La salida esperada, el método funciona correctamente	1,3
Incorrecta	Paciente es nulo Médico existe Escenario 1	No se agrega la consulta	Un mensaje de error y no se agrega la consulta	2,3
Incorrecta	El médico no existe Escenario 2	No se agrega la consulta	La consulta no se carga,y no se informa al usuario	1,4

Clase: Médico
Método: muestraReporte

Escenarios

Escenario	Descripción
1	La colección de reportes se encuentra vacía
2	La colección de reportes se encuentra con datos Ej: Reportes entre 18-06-2020 y 25-06-2020, con un total = 21230

Tabla de particiones

Condición de entrada	Clases VÁLIDAS	Clases INVÁLIDAS
fecha1,fecha2	fecha1 < fecha2(1)	fecha1 >= fecha2(2)
Estado colección de reporte	Existen reportes en esas fechas(3)	No existen reportes (4)

Batería de pruebas

Clase (Correcta o Incorrecta)	Valores de entrada	Salida esperada	Salida obtenida	Clases cubiertas
Correcta	fecha1 = 15-07=2021 fecha2 = 19-08-2021 Escenario 1	FechaInvalidaException	La salida esperada, el método funciona correctamente No hay reportes	1,4
Correcta	fecha1 = 15-06=2020 fecha2 = 30-06-2020 Escenario 2	Devuelve los reportes del médico Total = 21230	Salida Esperada, Funciona Correctamente	1,,3
Incorrecta	fecha1 = 30 -06=2020 fecha2 = 20-06-2020 Escenario 2	FechaInvalidaException	Salida Esperada, Funciona Correctamente fecha2>fecha1	2,3

Clase: Factura
Método: Constructor

No hay distintos escenarios posibles.

Tabla de particiones

Condición de entrada	Clases VÁLIDAS	Clases INVÁLIDAS
Paciente	Paciente!=null (1)	Paciente == null (2)

Batería de pruebas

Clase (Correcta o Incorrecta)	Valores de entrada	Salida esperada	Salida obtenida	Clases cubiertas
Correcta	Paciente!=null Nombre: Lisandro DAlu	Se crea correctamente y le agrega el paciente a la factura correspondiente	La salida esperada, el método funciona correctamente Se crea la factura correctamente	1
Incorrecta	Paciente == null	PacienteInvalidoExcepti on	Salida Esperada, Funciona Correctamente Se trató de crear con paciente null	2

Clase: SalaEsperaPrivada
Método: ingresaPaciente
Escenarios

Escenario	Descripción
1	La sala se encuentra vacía
2	La sala se encuentra con un paciente. Ej: Paciente con rango etario = "Adulto"

Tabla de particiones

Condición de entrada	Clases VÁLIDAS	Clases INVÁLIDAS
Paciente	Rango Etario de mayor prioridad (1)	Rango Etario de menor prioridad (2)
Estado de la sala	Existe un paciente (3)	No existe un paciente (4)

Batería de pruebas

Clase (Correcta o Incorrecta)	Valores de entrada	Salida esperada	Salida obtenida	Clases cubiertas
Correcta	Paciente con rango Etario = "Joven" Escenario 2	Se ingresa al joven en la sala privada ya que le gana al adulto Segun la documentacion pierde siempre	La salida esperada, el método funciona correctamente	1,3
Correcta	Paciente con rango Etario = "Joven" Escenario 1	Se ingresa ya que la sala está vacía	Salida Esperada, funciona correctamente	1,4
Incorrecta	Paciente con rango = "Adulto"	NoIngresaSalaPrivadaException	Salida Esperada, Funciona Correctamente El paciente se va al Patio	2,3

Clase: HabCompartida
Método: calculaArancel

No hay distintos escenarios posibles.

Tabla de particiones

Condición de entrada	Clases VÁLIDAS	Clases INVÁLIDAS
cantDias	cantDias > 0 (1)	cantDias <= 0 (2)

Batería de Pruebas

Clase (Correcta o Incorrecta)	Valores de entrada	Salida esperada	Salida obtenida	Clases cubiertas
Correcta	cantDias = 1	Devuelve el arancel correctamente Arancel = costodeAsigancion + CostoInicial	La salida esperada, el método funciona correctamente	1
Incorrecta	cantDias = 0	DiasInvalidosException	La salida esperada, se ingresó una cantidad menor o igual a 0.	2

Caja Blanca

Las pruebas de caja blanca se centran en los detalles procedimentales del software, por lo que su diseño está fuertemente ligado al código fuente. El testeador escoge distintos valores de entrada para examinar cada uno de los posibles flujos de ejecución del programa y cerciorarse de que se devuelven los valores de salida adecuados.
Por lo tanto a diferencia de caja negra, estas pruebas se centran en analizar el código fuente, intentando que todas las líneas de código se ejecuten.

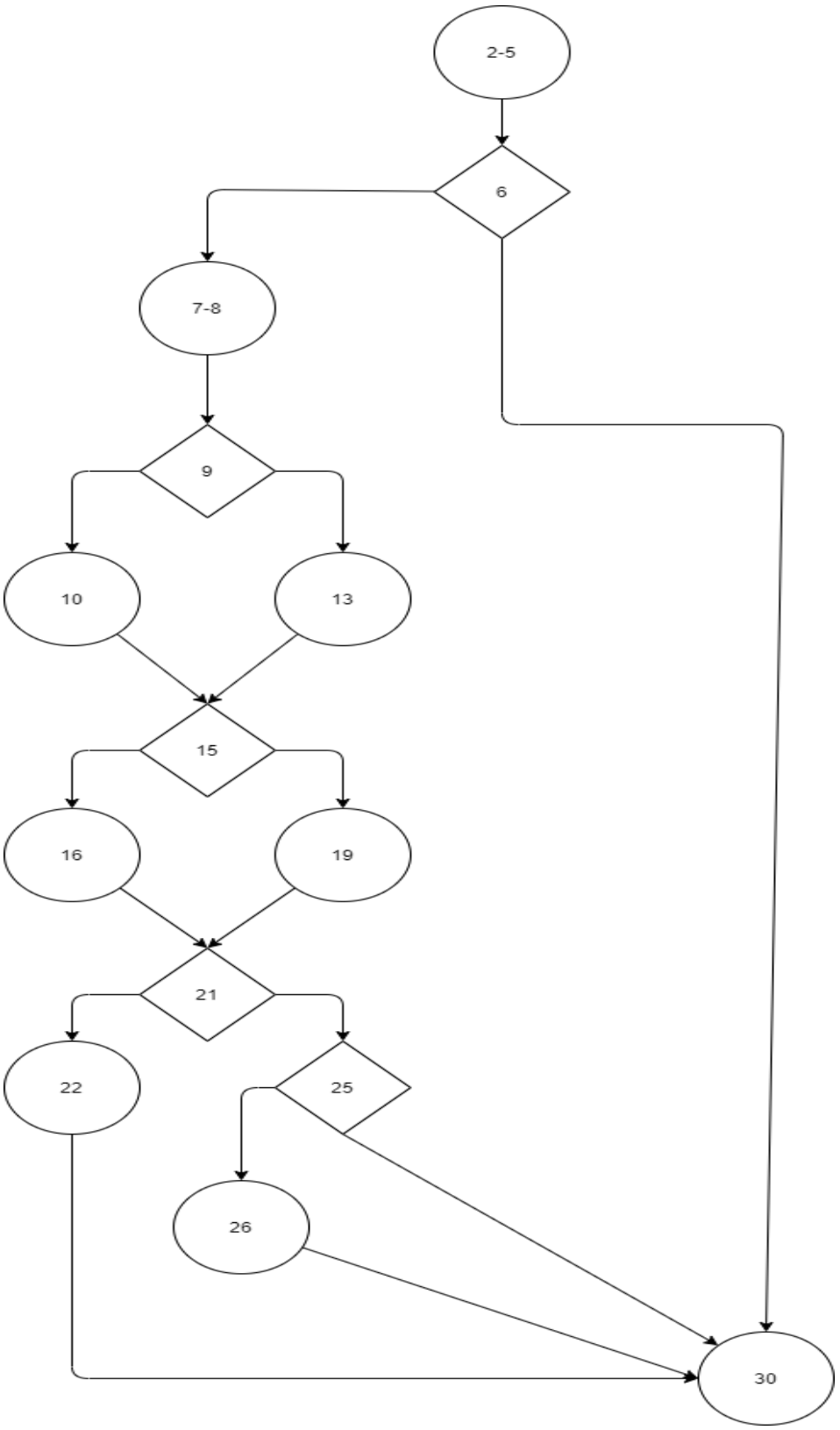
En el caso particular de este trabajo,vamos a analizar un método que calcula importes adicionales de una factura.

A partir del pseudocódigo dado, codificamos el método, obteniendo el siguiente código Java:

```java
public double calculoImporteAdicionales(int numeroDeFactura, GregorianCalendar fechaDeSolicitud, ArrayList<Double> listaDeInsumos) {
double respuesta = 0;
double importeparcial=0,importetotal=0;
long diasDiferencia=0;
Factura factura= ExistenFactura(numeroDeFactura);
if (factura!=null) {
    //factura.ImprimeFacturaConsola(); Se hace en agregaFactura()
    diasDiferencia= ((fechaDeSolicitud.getTimeInMillis()-factura.getFecha().getTimeInMillis())/(1000 * 60 * 60 * 24));
    if(diasDiferencia<10) {
        importeparcial=factura.getCostoTotalFactura()-(factura.sumaprestacionesimpares()*0.8);
    }
    else {
        importeparcial=factura.getCostoTotalFactura()*0.7;
    }
    if(factura.getPaciente().getRangoEtario().equals("Mayor")) {
        importetotal=importeparcial*1.5;
    }
    else {
        importetotal=importeparcial*0.9;
    }
    if(NumeroAzar.getNumero() ==factura.getFecha().toZonedDateTime().getDayOfMonth()) {
        respuesta=importetotal;
    }
    else {
        if(listaDeInsumos!=null && listaDeInsumos.size()>0) {
            respuesta=importetotal+sumaArray(listaDeInsumos);
        }
    }
}
return respuesta;
```

Grafo Ciclomático

Antes de desarrollar las pruebas para lograr la cobertura mostrada en la imagen, comenzamos a analizar el método realizando un grafo ciclomático, el cual adjuntamos:



Un vez realizado el grafo, calculamos la complejidad contando los nodos condición:
Complejidad = Nodos Condición + 1 =
Complejidad = 5 + 1 = 6

Entonces hay 6 caminos como cota máxima.

A partir de esto, comenzamos a buscar los caminos utilizando el método simplificado, esto significa que comenzamos buscando el camino más corto y luego nos extendemos a partir de ese

Caminos

1) (2-5)-(6)-(30)

2) (2-5)-(6)-(7-8)-(9)-(10)-(15)-(16)-(21)-(22)-(30)

3) (2-5)-(6)-(7-8)-(9)-(13)-(15)-(19)-(21)-(25)-(26)-(30)

4) (2-5)-(6)-(7-8)-(9)-(13)-(15)-(19)-(21)-(25)-(30)

Tras plantear estos caminos, se recorren todos los nodos del grafo.
Es importante resaltar que tenemos una cobertura del 100% del método, esto se puede observar en las líneas verdes a la izquierda en la imágen del código.
Debido a la extensión de las pruebas decidimos no colocarlas en el informe, pero invitamos al lector a verlas en nuestro repositorio Github. Las pruebas están en la clase "testCajaBlanca" en el paquete de testing: Pruebas de Caja Blanca

Casos de prueba caja blanca

Escenario 1: Lista de insumos vacía o nula.
Escenario 2: Lista de insumos con al menos un insumo.

Observación: Al momento de establecer las salidas, no es posible dar un valor exacto de retorno, ya que depende de los valores de A, B, C, D y de los costos de la factura específica.

Camino	Escenario	Caso	Salida esperada
1	1 o 2 es indistinto	No existe nro factura	Devuelve 0, no se puede hacer los cálculos sin una factura válida
2	1 o 2 es indistinto	Existe nro factura Días de diferencia<10 El paciente es "Mayor" Valor aleatorio=fecha	Importes adicionales aumentan por ser mayor, pero disminuyen por ser menor a 10 días la diferencia de fechas. No recibe cargos adicionales por insumos, porque salió beneficiado con el número aleatorio
3	2	Existe nro factura Días de diferencia >=10 El paciente no es "Mayor" Valor aleatorio!=fecha Hay por lo menos un valor válido en la lista de insumos	Recibe descuento por no ser mayor y por ser mayor a 10 la diferencia de días. Aumenta el costo por los insumos en la lista, no coincide el número aleatorio
4	1	Existe nro factura Días de diferencia >=10 El paciente no es "Mayor" Valor aleatorio!=fecha La lista de insumos está vacía o es null	Recibe descuento por no ser mayor y por ser mayor a 10 la diferencia de días. No recibe cargos adicionales por insumos, ya que no hay ninguno.

Test de Persistencia

Para el test de persistencia se eligió el módulo de los pacientes. El método de persistencia usado en esta clínica fue XML.
Durante las pruebas se testean los siguientes escenarios:

- La creación correcta del archivo.
- La escritura en el archivo, tanto con el arreglo de pacientes vacíos como con pacientes cargados.
- Despersistir con un archivo incorrecto que no existe.
- Despersistir con el archivo correcto.

En todos estos escenarios no se encontraron fallas ni errores. Por lo que se concluyó que la persistencia del módulo de pacientes está bien implementada.

Test de GUI

Implementamos pruebas para la ventana de la clínica, para esto elegimos el módulo de facturación que incluye la carga de consultas e internaciones.

El testeo de interfaces gráficas lo hicimos utilizando JUnit 4 y la clase Robot perteneciente al paquete AWT de Java.
El método de testeo automatizado usado consiste en simular la interacción del usuario, utilizando la clase robot para clickear y completar los componentes de la ventana (cuadros de texto, botones, listas, etc).

Como objetivo a testear, nos enfocamos en verificar que los ingresos por teclado estén validados, y que todo funcione como es esperado.

Tras la primer iteración de testeo, nos encontramos los siguientes problemas en el código:

- No se lanza un error ni se valida que la fecha de inicio para la consulta de facturas sea anterior a la de final.
- Según el contrato el botón para generar una internación se desactiva cuando se ingresa un valor no válido, sin embargo esto no ocurre para números negativos.
- Si los campos de la consulta de factura están vacíos, el sistema deja de funcionar debido a que ambos campos son nulos.

Más allá de estos errores, el resto de la ventana responde a lo pedido según el contrato, validando las entradas y lanzando ventanas pop-up ante errores, solicitando el ingreso del dato erróneo nuevamente.

En el vídeo entregado junto con el informe mostraremos todas las pruebas de la ventana, visualizando los movimientos por la ventana del Robot que programamos.

Test de Integración

Las pruebas de integración lo que prueban es que todos los elementos unitarios que componen el software, funcionan juntos correctamente probándolos en grupo. Se centra principalmente en probar la comunicación entre los componentes y sus comunicaciones

Una herramienta muy útil para las pruebas de integración son los mocks, que son objetos de "mentira" que imitan el comportamiento de componentes aún no implementados.

En el caso de este trabajo, aplicamos integración en múltiples casos, esto se debe a que muchos componentes de la clínica interactúan entre sí.

También nos vimos forzados a usar un mock para las pruebas unitarias del método de caja blanca de cálculo de importes adicionales, esto fue así ya que una parte del método requiere el uso de números aleatorios.
Para solucionar este problema, utilizamos una clase mock llamada "NumeroAzar" que permite generar números al azar usando el método Math.random(), pero también permite "forzar" el siguiente número aleatorio solamente durante la fase de testeo, para así lograr probar todos los caminos de prueba.

Por otro lado, podemos concluir que nuestras pruebas de integración son orientadas a objetos, ya que probamos individualmente cada uno de los componentes a nivel de unidad, para poder verificar luego su correcto funcionamiento en conjunto

Conclusión

Como se vió en el desarrollo del presente informe, se utilizaron distintos métodos de testing a lo largo del mismo de acuerdo al código que se iba a probar.

En las pruebas de caja negra se priorizaron los métodos que consideramos de mayor importancia para el software ya que el gran tamaño del proyecto nos obliga a enfocarnos solo en las clases más esenciales para el correcto funcionamiento del programa.
En la elaboración de este tipo de testing utilizamos el Javadoc, para en base a las precondiciones y postcondiciones verificar el correcto funcionamiento del código sin necesidad de ver el código del mismo. La mayoría de los métodos probados utilizando caja negra funcionan correctamente, salvo uno (ingresaPaciente) que permite el ingreso de pacientes repetidos, el cual no cumple con los contratos pre-establecidos lo cual sugiere mediocre robustez en el código original, ya que no se cumplió algo que era pre-condición.

En el testing de caja blanca se verificó el correcto funcionamiento del método calculaImportesAdicionales, lo primero que hicimos fue graficar el grafo ciclomático del código, luego calculamos la complejidad del mismo dándonos como resultado que tiene una cota máxima de 6 caminos, para luego concluir que tiene 4 caminos básicos. Luego planteamos los escenarios y confeccionamos un cuadro, dando como resultado que en todos los casos existen salidas válidas.

Luego, pudimos verificar que la ventana es bastante robusta al momento de cargarle consultas e internaciones a un paciente (facturación) solo se encontraron errores menores en esta sección

Respecto al módulo de consulta de facturas, podemos decir que tras las pruebas de GUI concluimos que no se validan los errores más comunes, por ejemplo dejar campos en texto en blanco o colocar una fecha de inicio posterior a la fecha final.

En cuanto al test de persistencia, se optó por el módulo de los pacientes, en todos los escenarios planteados el funcionamiento de la persistencia fue correcta.