Universidad Nacional de Mar del Plata.
Facultad de Ingeniería.
Departamento de Ingeniería informática.
Asignatura: Taller de Programación I

Trabajo Práctico Especial

Sistema de Cervecería

Grupo 5 - Subgrupo 2

Integrantes:

- Luna, Lautaro
- Sosa, Santiago

Repositorio: https://github.com/lunalauti/TPTallerDeProgramacion

Fecha de entrega: 16/11/2022

Índice

Introducción

La intención de este informe es expresar los resultados obtenidos tras llevar a cabo un proceso de testing en un sistema elaborado para administrar una cervecería. El sistema utilizado en nuestro caso fue elaborado por el subgrupo opuesto.

Testing

Entre las técnicas aplicadas para el testing podemos distinguir: pruebas de Caja Negra, pruebas de Caja Blanca, test de Persistencia, test de Integración, test de GUI y test de Excepciones. Dichas técnicas serán desarrolladas a continuación con mayor detenimiento.

Caja Negra

En esta prueba los casos de prueba se basan solo en el comportamiento de entrada/salida ya que no se utiliza información del código implementado, y se encargan de verificar si el código cumple los contratos especificados. Para nuestros casos de prueba tomamos dos escenarios posibles, uno con la cervecería vacía (sin datos) y otro con la cervecería cargada (con datos).

Métodos a testear

Sucedió con varios métodos que ibamos a testear que la gran cantidad de precondiciones impuestas provocaba que no se lancen excepciones (lo que no permite crear clases inválidas). Esto causaba que muchos parámetros de entrada puedan tomar solamente valores válidos (ninguno fuera de rango o incorrecto)

Debido a esto, se seleccionaron (para incluir en el informe sus tablas de particiones y baterías de pruebas) métodos que no contaban con estas dificultades para ejecutar los tests. Además, los consideramos importantes para el flujo común del programa:

Clase: Cervecería
Método: asignarMesa

Excepciones que se lanzan:

- MozoNoDisponibleException,
- MozoInexistenteException,
- MesaInexistenteException
- MesaNoDisponibleException

Nro de escenario	Descripción
1	Cerveceria con una mesa. **mesa3** (nroMesa=3, sin asignar). No hay mozos en la cervecería.
2	Cervecería con al menos 2 mozos. **Mozo1** (nya= "Juan Lopez" , activo) y **mozo2** (nya="Jose Perez" , ausente). Además, hay 2 mesas. **mesa1** (NroMesa=1 , sin asignar) y **mesa4** (NroMesa=4 , asignada)

Tabla de Particiones		
Condición de entrada	**Clases Válidas**	**Clases Inválidas**
mozo	el mozo existe y esta activo (1)	el mozo no existe en la cerveceria (2)
		El mozo no está activo (3)
mesa	la mesa existe y no esta asignada (4)	La mesa no existe en la cervecería (5)
		La mesa ya fue asignada (6)

Batería de Pruebas			
Tipos de clase (correcta o incorrecta)	**Valores de entrada**	**Salida esperada**	**Clases de prueba cubiertas**
Incorrecta	Mozo1 (nya= "Juan Lopez" , activo) Mesa3 (NroMesa = 3, sin asignar) **Escenario 1**	MozoInexistenteException	2
Incorrecta	mozo2 (nya="Jose Perez" , ausente) mesa1 (NroMesa=1 , sin asignar) **Escenario 2**	MozoNoDisponibleException	3,4
Incorrecta	Mozo1 (nya= "Juan Lopez" , activo) Mesa3 (NroMesa = 3,sin asignar) **Escenario 2**	MesaInexistenteException	1,5
Incorrecta	Mozo1 (nya= "Juan Lopez" , activo) Mesa4 (NroMesa = 4, asignada) **Escenario 2**	MesaNoDisponibleException	1,6
Correcta	Mozo1 (nya= "Juan Lopez" , activo) Mesa1 (NroMesa = 1, sin asignar) **Escenario 2**	Se asigna la mesa1 al mozo1 correctamente	1,4

Clase: Cerveceria
método: EliminarPromocion(PromoProducto)

Excepciones que se lanzan:

- PromocionInexistenteException

Batería de Pruebas			
Tipos de clase (correcta o incorrecta)	**Valores de entrada**	**Salida esperada**	**Clases de prueba cubiertas**
Incorrecta	promo5 **Escenario 1**	PromocionInexistenteException	1
Incorrecta	promo1 **Escenario 2**	PromocionInexistenteException	1
Correcta	promo3 **Escenario 2**	Se elimina la promoción correctamente	2

Clase: Cervecería
método: agregarProducto

Excepciones que se lanzan:

- ProductoRepetidoException

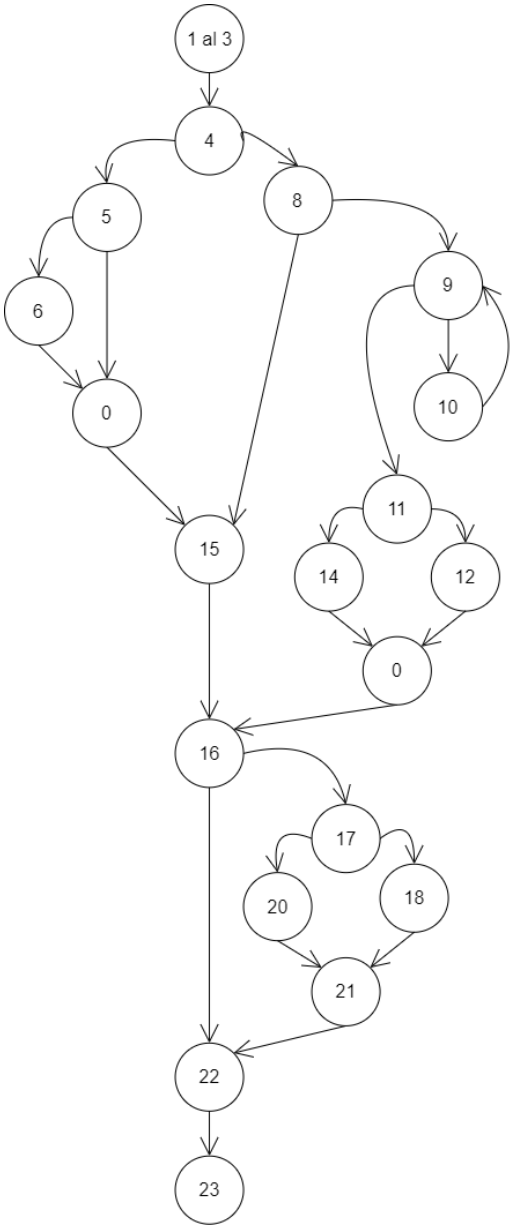
Escenarios	
Nro de escenario	**Descripción**
1	Cerveceria vacía.
2	Cervecería cargada con productos. Existe producto1 [hamburguesa]. producto4 [milanesa], no existe.

Tabla de Particiones		
Condición de entrada	**Clases Válidas**	**Clases Inválidas**
producto	no existe en la cervecería (1)	existe en la cervecería (2)

Batería de Pruebas			
Tipos de clase (correcta o incorrecta)	**Valores de entrada**	**Salida esperada**	**Clases de prueba cubiertas**
Correcta	producto3 **Escenario 1**	Se agrega el producto3 correctamente	1
Correcta	producto4 **Escenario 2**	Se agrega el producto4 correctamente	1
Inorrecta	producto1 **Escenario 2**	ProductoRepetidoException	2

Los tests realizados se completaron de forma exitosa (test incluidos en el paquete test del repositorio). Sin embargo, cabe destacar que para el correcto funcionamiento se debió eliminar el aserto que verifica la cantidad de comensales en el constructor de la clase Mesa. Al ser un problema ajeno a los tests se comentaron estas líneas y se pudieron cargar los escenarios y ejecutar los tests correctamente.

Además de los tests mencionados anteriormente, se llevó a cabo la eliminación de todos los objetos que contiene la cervecería (productos, mesas, mozos, operarios, promociones) (que también se llevó a cabo sin encontrar errores).



Caja Blanca

En esta prueba los casos de prueba se basan en información sobre cómo el software ha sido diseñado o codificado, el objetivo de ésta técnica es diseñar casos de prueba para que se ejecuten, al menos una vez, todas las sentencias del programa y todas las condiciones tanto por verdadero como por falso, además de comprobar de que se devuelven los valores de salida adecuados.

Para aplicar esta técnica a este sistema en particular elegimos un método que consideramos interesante analizar el método **loguear** de la clase Cervecería, ya que es uno de los métodos donde más nodos condicionales encontramos y por ende, un método que tendrá una mayor cantidad de caminos.

Grafo ciclomático

A partir del **método loguear** de la clase Cervecería se elaboró este grafo de control que representa los nodos por los que puede transitar su ejecución. Para esta técnica tomamos como criterio de cobertura a la **Cobertura de Decisión,** la cual busca lograr la cobertura de un porcentaje específico de todas las decisiones.

Una forma de determinar el número de caminos independientes dentro de un fragmento de código y determinar el límite superior número de pruebas que se deben realizar para asegurar que se ejecuta cada sentencia al menos una vez es calculando la **Complejidad Ciclomática:**

V(G) = arcos - nodos + 2 = número de nodos condición + 1= **8**

```
1   public void loguear(String username, String password, String tipo) {
2           int i = 0;
3           String mensaje = "INCORRECTO";
4           if (tipo.equalsIgnoreCase("ADMIN")) {
5               if (this.getAdmin().getUsername().equals(username) && this.getAdmin().getPassword().equals(password))
6                   mensaje = "ADMIN";
7           }else
8               if (tipo.equalsIgnoreCase("OPERARIO")) {
9                   while (i < this.operarios.size() && !this.operarios.get(i).getUsername().equals(username))
10                      i++;
11                  if (i < this.operarios.size() && this.operarios.get(i).getPassword().equals(password) && this.operarios.get(i).isActivo())
12                      mensaje = "OPERARIO";
13                  else if (i < this.operarios.size() && this.operarios.get(i).getPassword().equals(password) && !this.operarios.get(i).isActivo())
14                      mensaje = "INACTIVO";
15              }
16          if (mensaje != "INCORRECTO" || mensaje != "INACTIVO") {
17              if (mensaje == "OPERARIO")
18                  setOperarioLogueado(this.operarios.get(i));
19              else
20                  setOperarioLogueado(this.getAdmin());
21          }
22          this.setChanged();
23          this.notifyObservers(mensaje);
24      }
```

N°	Camino (Método General)
1	(1-3)-4-8-9-10-9-11-12-16-17-18-21-22-23
2	(1-3)-4-8-9-11-12-16-17-18-21-22-23
3	(1-3)-4-8-9-10-9-11-14-16-17-18-21-22-23
4	(1-3)-4-8-9-10-9-11-14-16-17-20-21-22-23
5	(1-3)-4-8-9-10-9-11-14-16-22-23
6	(1-3)-4-8-15-16-22-23
7	(1-3)-4-5-6-15-16-22-23
8	(1-3)-4-5-15-16-22-23

De los 8 caminos definidos, podemos agruparlos en 5 caminos para ejecutar el test de cobertura de manera más sencilla, quedando así los siguientes caminos:

N°	Caminos
1	(1-3) - 4 - 8 - 9 - 10 - 9 - 11 - 12 - 16 - 17 - 18 - 21 - 22 - 23
2	(1-3) - 4 - 8 - 9 - 11 - 14 - 16 - 22 - 23
3	(1-3) - 4 - 8 - 15 - 16 - 22 - 23
4	(1-3) - 4 - 5 - 6 - 15 - 16 - 17 - 20 - 21 - 22 - 23
5	(1-3) - 4 - 5 -15- 16 - 22 - 23

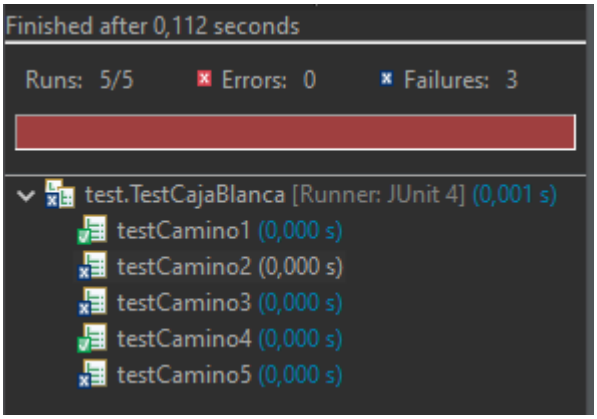
Al implementar estos 5 caminos vemos que se recorren todos los nodos del grafo de control, esto lo comprobamos al ejecutar el test de cobertura y observar que no quedaron líneas de código sin ejecutar.

```
571
572     public void loguear(String username, String password, String tipo) { // ----
573         int i = 0;
574         String mensaje = "INCORRECTO";
575
576         if (tipo.equalsIgnoreCase("ADMIN")) {
577             if (this.getAdmin().getUsername().equals(username) && this.getAdmin().getPassword().equals(password))
578                 mensaje = "ADMIN";
579
580         } else if (tipo.equalsIgnoreCase("OPERARIO")) {
581
582             while (i < this.operarios.size() && !this.operarios.get(i).getUsername().equals(username))
583                 i++;
584             if (i < this.operarios.size() && this.operarios.get(i).getPassword().equals(password)
585                     && this.operarios.get(i).isActivo())
586                 mensaje = "OPERARIO";
587             else if (i < this.operarios.size() && this.operarios.get(i).getPassword().equals(password)
588                     && !this.operarios.get(i).isActivo())
589                 mensaje = "INACTIVO";
590
591         }
592
593         if (mensaje != "INCORRECTO" || mensaje != "INACTIVO") {
594             if (mensaje == "OPERARIO")
595                 setOperarioLogueado(this.operarios.get(i));
596             else {
597                 setOperarioLogueado(this.getAdmin());
598             }
599         }
```

Casos de prueba caja blanca

Escenario 1: Cervecería vacía (solo con ADMIN).
Escenario 2: Cervecería con al menos dos operarios (username=”user1” y username=”user2” y ADMIN)

Camino	Escenario	Caso	Salida esperada
1	2	username=”user2” password=”User456” tipo=”OPERARIO”	Operario logueado = “user2”
2	2	username=”user1” password=”User1234” tipo=”OPERARIO” “user1” está inactivo en la cervecería	Operario logueado= null
3	Cualquiera	tipo != {OPERARIO,ADMIN}	Operario logueado= null
4	Cualquiera	username=”ADMIN” password=”ADMIN1234” tipo=”ADMIN”	Operario logueado= “ADMIN”
5	Cualquiera	username!= ”ADMIN”	Operario logueado= null



Durante la ejecución de las pruebas de caja blanca, podemos observar que el resultado esperado no fue el mismo que el obtenido en los caminos 2, 3 y 5, mostrando el siguiente error:

AssertionFailedError:expected:<null>
but was:<ADMIN (ADMIN), activo>

Determinando así que se detectó un error en la implementación del método al momento de intentar loguear a un operario inactivo, al ingresar un tipo distinto a “OPERARIO” o “ADMIN” y al ingresar un nombre de usuario incorrecto para el Admin.

Test de Persistencia

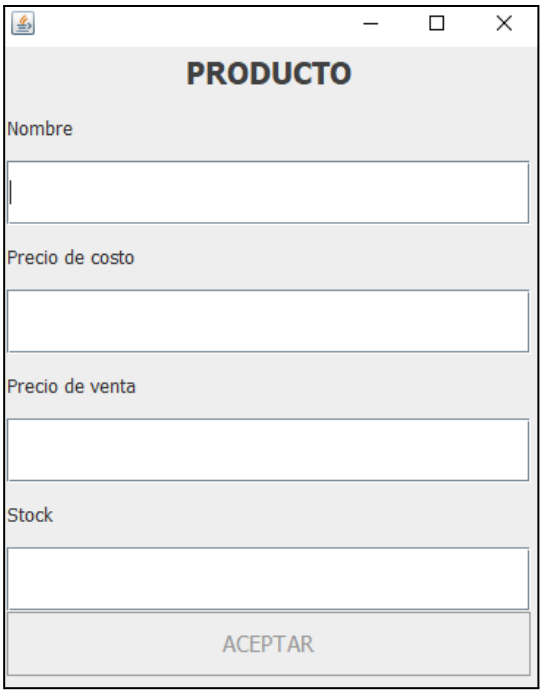
Para el test de persistencia se eligió el módulo de los productos ya que se encuentra entre los más importantes a persistir y a la hora de utilizar un escenario con datos es sencilla su carga. En el sistema se utilizó un archivo binario “Cerveceria.bin” para persistir los datos. Como posibles escenarios se tomó uno con la lista de productos vacía, y uno con la lista de productos cargada.

Durante las pruebas testeamos los siguientes casos:

- La creación correcta del archivo.
- La escritura en el archivo, para ambos escenarios
- Despersistir tanto cuando no existe el archivo como cuando existe.

No se encontraron fallas en ninguna de las pruebas ejecutadas, por lo tanto, asumimos que la persistencia del sistema está implementada de manera correcta.

Test de GUI



Para las pruebas de funcionalidad de las interfaces gráficas utilizamos la técnica de métodos automatizados, la cual consiste en simular la interacción del usuario, generar casos de prueba y comparar los resultados de la ejecución de dichos casos en la aplicación real con los esperados. Esta simulación de interacción la hicimos utilizando la clase Robot que provee Java para clickear y completar los componentes de la interfaz gráfica.

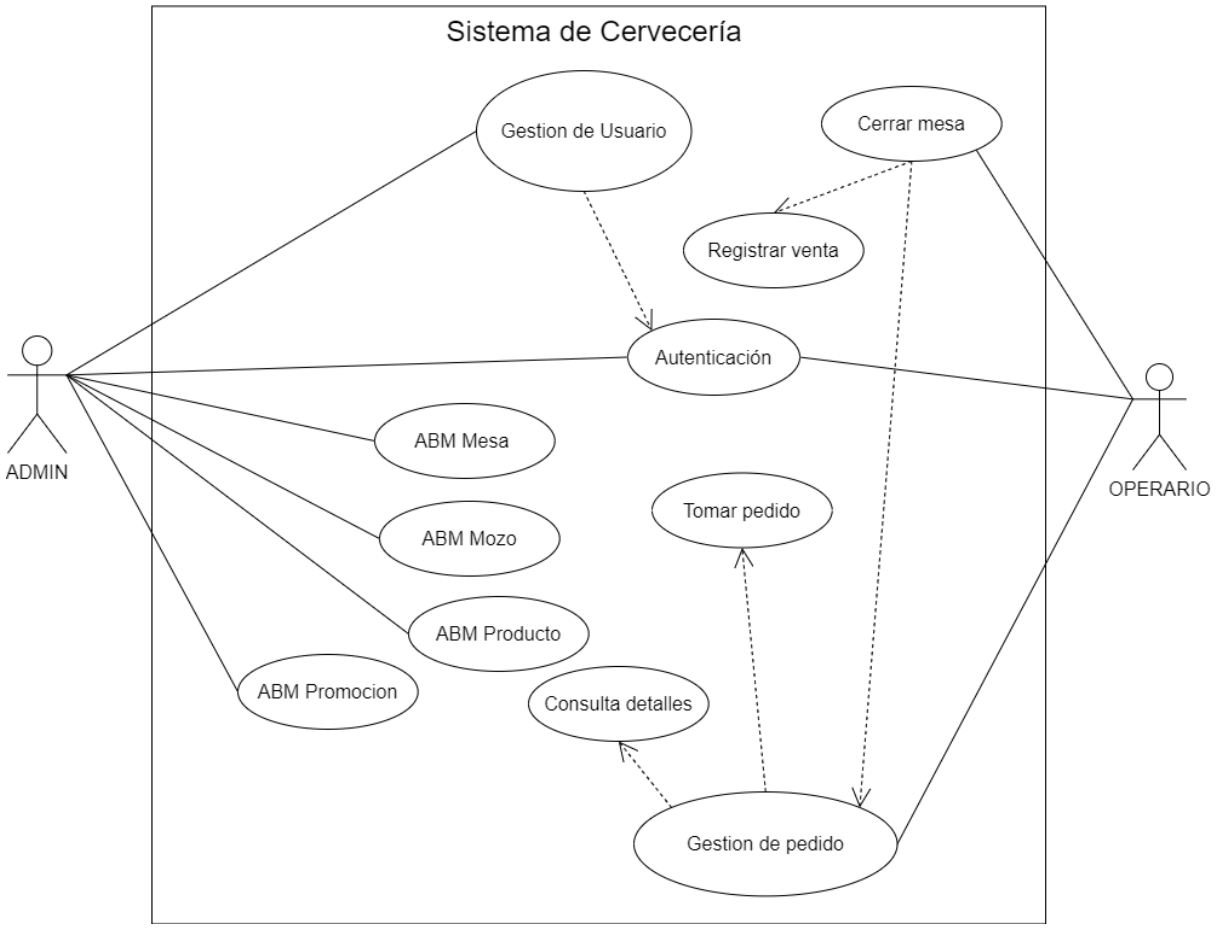
Para esta prueba decidimos utilizar la ventana VProducto, la cual agrega nuevos productos al sistema, ya que tenía varios JTextFields para completar y tenía implementado el bloqueo de los botones.

Entre las pruebas aplicadas distinguimos a verificar que el botón se habilite cuando corresponde, el correcto ingreso de datos al sistema y que se muestren los mensajes esperados.

No se encontraron fallas en ninguna de las pruebas ejecutadas, por lo tanto, asumimos que la ventana de productos está implementada de manera correcta.

Test de Integración

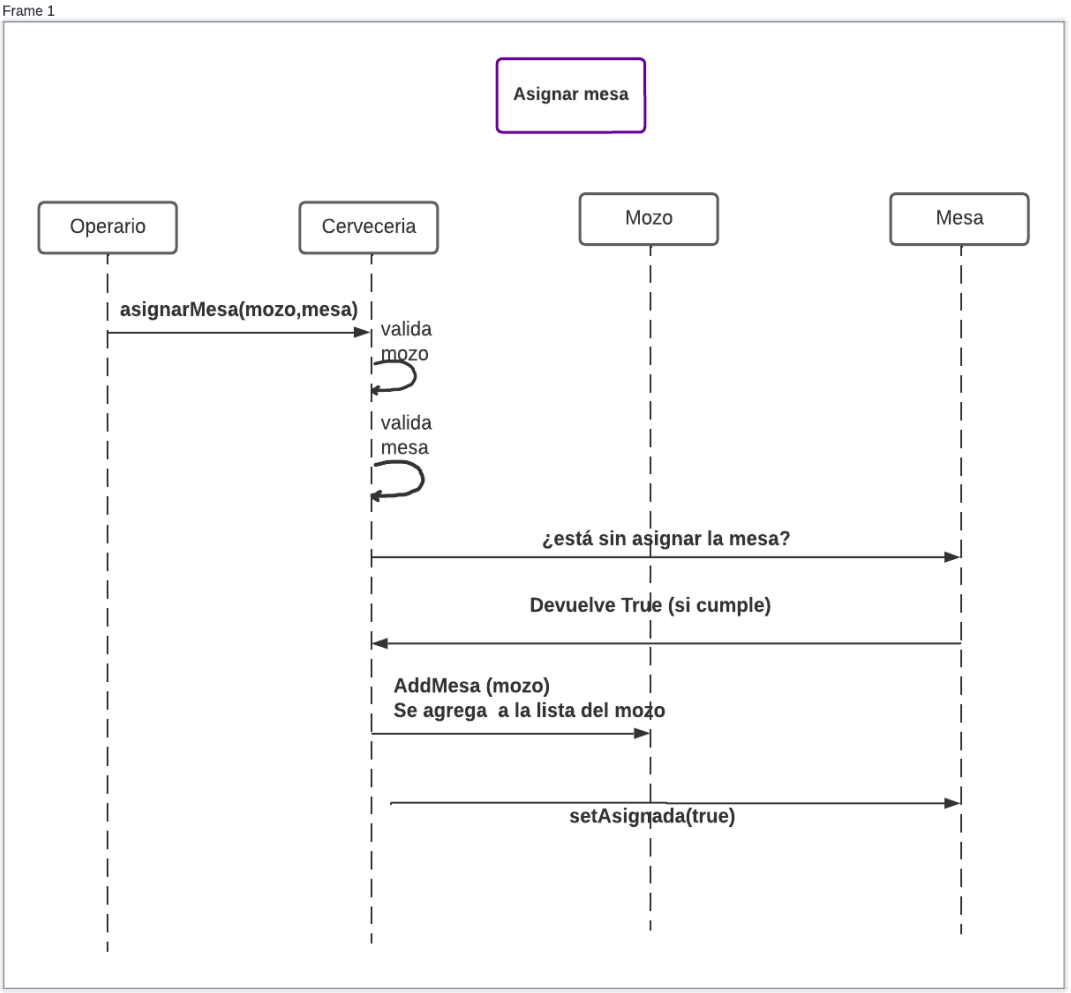
Diagrama de Casos de uso



Para determinar los casos de prueba en el test de integración, se realizó el diagrama de secuencias para los métodos asignarMesa() y tomarComanda() ambos de la clase Operario.

Diagramas de secuencias

Caso de uso: AsignarMesa



Escenario	Descripción
1	Cervecería vacía.
2	Cervecería con un mozo1 (activo), un mozo2 (ausente). Además, mesa1 (sin asignar). mesa2 (ya asignada).

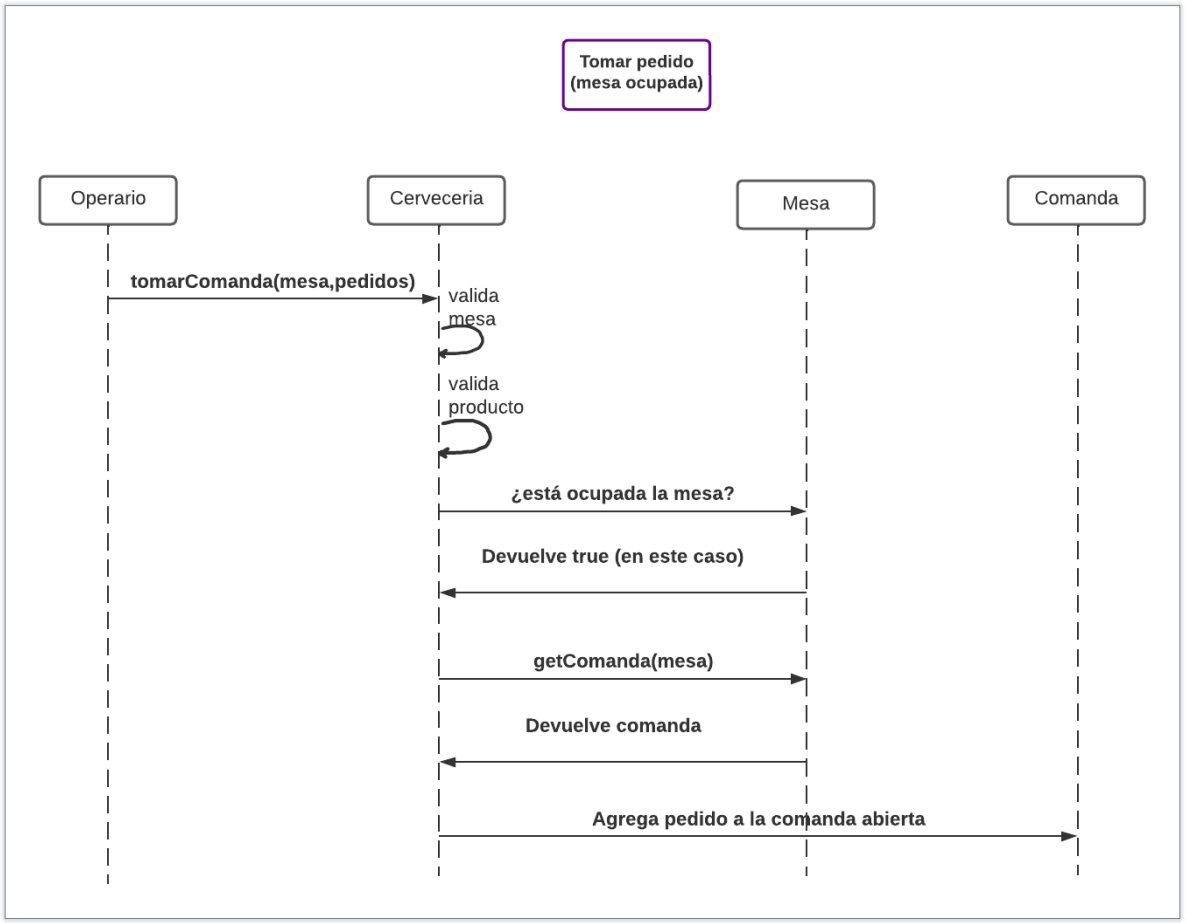
Caso	Entrada	Salida esperada
1	mozo1 mesa1 **escenario1**	MesaInexistenteException
2	mozo1 mesa2 escenario2	MesaNoDisponibleException
3	mozo4 mesa1 escenario2	MozoInexistenteException
4	mozo2 mesa1 escenario2	MozoNoDisponibleException
5	mozo1 mesa1 escenario2	Se asigna la mesa

Caso de uso: tomarComanda

Escenario	Descripción
1	Cervecería vacía.
2	Cervecería con producto1, mesa1 (ocupada)

Caso	Entrada	Salida esperada
1	mesa2 producto1 **escenario1**	MesaInexistenteException
2	mesa1 producto4 **escenario2**	ProductoInexistenteException
3	mesa1 producto1 **escenario2**	Se tomó el pedido correctamente

Frame 1


Tomar pedido
(mesa ocupada)
Operario
Cerveceria
Mesa
Comanda
tomarComanda(mesa,pedidos)
valida
mesa
valida
producto
¿está ocupada la mesa?
Devuelve true (en este caso)
getComanda(mesa)
Devuelve comanda
Agrega pedido a la comanda abierta

Conclusión

Durante el transcurso de la etapa de testeo, se tuvieron en cuenta las distintas técnicas vistas en la materia.

Al tratarse de un sistema tan grande, en el momento de las pruebas de caja negra se priorizó el testeo de los métodos que consideramos más relevantes para su correcto funcionamiento. Durante el transcurso de esta técnica de testing fue de suma importancia el Javadoc provisto por nuestros compañeros para poder verificar el comportamiento del sistema sin tener que ejecutarlo. Los tests realizados se completaron de forma exitosa y no se detectó ninguna falla en los métodos.

Gracias al testing de caja blanca pudimos identificar una falla en el método loguear de la clase Cervecería, el cual setea al administrador como logueado si el usuario está inactivo y si la contraseña del ADMIN no es correcta. Para detectar esta falla primero se elaboró el grafo ciclomático del método, sobre el cual se calculó la complejidad dando como resultado 8, es decir, que la cota máxima de caminos es 8. Sin embargo, pudimos cubrir todas las condiciones utilizando 5 caminos, comprobandolo con el test de cobertura.

Podemos decir que el módulo de agregar productos al sistema mediante interfaces gráficas funciona correctamente luego de llevar a cabo pruebas de GUI en las que se evaluó la validación de datos de entrada y el funcionamiento de los botones y campos de texto. Cabe destacar que para poder realizar este tipo de test se debió modificar la ventana, asignando los nombres a sus componentes para poder acceder a ellos posteriormente, sin embargo, estas modificaciones no afectaron al funcionamiento natural de la misma.

En cuanto al test de persistencia, se optó por probar el módulo de los productos, el cual resultó correcto en todos los escenarios planteados.

En conclusión, al llevar a cabo este proyecto pudimos apreciar la importancia que tiene la etapa de testing en el desarrollo de software para lograr cierto nivel de calidad. Además, pudimos observar el valor que tiene una documentación clara y completa, no solo para realizar pruebas sobre el sistema, sino también para entender con mayor facilidad su funcionamiento. Para finalizar, nos gustaría destacar la relevancia que tiene el trabajo en conjunto a la hora de desarrollar software, ya que al fin y al cabo este lleva a obtener un sistema con mayor integridad.