

Trabajo Práctico Grupal 2024

“Subí que te llevo”

Larsen, Jeremías Tomás

Mayor, Dario Agustin

Nigro, Maite

Verón, Francisco

Materia: Taller de Programación I.

Profesores: Guccione, Leonel; Lazzurri, Guillermo;
Spinelli, Adolfo; Bonifazi, Paula.

Fecha de entrega: 13/11/2024

● **Objetivos:**

El objetivo de este proyecto es testear el proyecto “Subí que te llevo” con los métodos vistos en la materia para asegurar la correcta funcionalidad del sistema. Los tipos de pruebas implementadas son:

- Caja Negra.
- Persistencia.
- GUI.
- Integración.

● **Introducción:**

Para la realización de este trabajo, se utilizó un programa desarrollado en la materia Programación III, este proyecto consiste en la gestión de una empresa de transporte de pasajeros, la cual tiene por objetivo administrar clientes, choferes, vehículos y viajes.

A partir de este proyecto, se buscó realizar los test anteriormente mencionados y así lograr detectar errores, fallas e inconsistencias que pueda presentar el código. De esta manera, pudimos aplicar y llevar a cabo todos los conceptos aprendidos en la materia Taller de Programación I.

● **Test Caja Negra:**

Se efectuaron pruebas unitarias utilizando la herramienta JUnit de JAVA, basándonos en el Javadoc del programa. En estas pruebas se evaluaron distintos casos de prueba para cada uno de los módulos del sistema que fueron solicitados en la consigna del trabajo. A continuación, se presentan las tablas de particiones y la batería de pruebas diseñadas para cada método que se consideró funcionalmente relevante.

Tabla particiones y Batería de Pruebas

Dentro de las pruebas unitarias, se lograron encontrar los siguientes errores al momento de ejecutar el código JUnit:

TestEmpresa:

- Escenario 1 :
 - Cuando se intenta hacer login con usuario admin y la password es incorrecta, lanza excepción UsuarioNoExisteException en lugar de lanzar excepción de PasswordErroneaException.
- Escenario 2 :
 - Permite agregar pedido cuando no hay vehículo que lo satisface. Se esperaba que arroje la excepción SinVehiculoParaPedidoException.

Asumimos que se da porque getPuntajePedido de alguno de los vehículos no funciona correctamente

- El parámetro vehículo que es pasado en la excepción VehiculoRepetido es incorrecto (es null).
- Escenario 3 :
 - El mensaje arrojado de la excepción ClienteConPedidoPendiente es incorrecto.
 - Si se pide la calificación promedio de un chofer que no posee viajes, no arroja excepción.
- Escenario 4 :
 - Permite que un usuario con viaje en proceso, pueda crear otro viaje.
 - El mensaje arrojado de la excepción ChoferNoDisponibleException es incorrecto.
 - Permite que un cliente con viaje en proceso, pueda cargar otro pedido. Se esperaba que arroje la excepción ClienteConViajePendienteException.
- Escenario 5:
 - Se considera como invalido la combi1 para el pedido1 y como consecuencia, al querer crear el viaje junto con el chofer1, lanza la excepcion VehiculoNoValidoException.


Resto de métodos:

- TestViaje:
 - El cálculo de la function getValor para todas las zonas no funciona correctamente.
- TestCombi:
 - La function getPuntajePedido se calcula incorrectamente.
- TestChoferPermanente:
 - El cálculo del sueldo bruto del chofer (getSueldoBruto) se calcula incorrectamente cuando tiene más de 20 años de antigüedad.
- TestAuto:
 - La function getPuntajePedido se calcula incorrectamente.

● **Integración:**

Para probar la integración, primero realizamos un diagrama de actividad donde están representados los casos de uso del sistema. Una vez listo derivamos los casos de prueba para poder realizar una integración descendente en profundidad.

En el siguiente link se puede visualizar el diagrama de actividad:

 [Diagrama de actividad](#)

Al realizar los tests no encontramos utilidad al hecho de aplicar Mocks ya que los métodos de los componentes que interfieren en las pruebas son de tipo void y era difícil imitar su

comportamiento correctamente sin conocer cómo están constituidos, por lo tanto optamos por seguir cada flujo utilizando sus componentes originales.

Finalmente una vez realizadas las pruebas no todos los resultados fueron exitosos, hubo 4 pruebas que fallaron:

1. Sucede cuando intentamos agregarle un pedido a un cliente que está realizando un viaje. Se esperaba la excepción `ClienteConViajePendienteException` pero no fue lanzada.
2. Sucede cuando intentamos crearle un viaje a un cliente que ya está realizando uno. Se esperaba la excepción `ClienteConViajePendienteException` pero no fue lanzada.
3. Sucede cuando intentamos crear un viaje con un chofer no disponible. El mensaje de la excepción no es el correcto. Se esperaba el mensaje "El chofer no está disponible" y en su lugar recibimos: "El pedido no figura en la lista".
4. Sucede cuando intentamos agregarle un pedido a un cliente que ya tiene uno pendiente. El mensaje de la excepción no es el correcto. Se esperaba el mensaje: "Cliente con pedido pendiente" y en su lugar recibimos: "Cliente con viaje pendiente".

Analizando los primeros dos resultados podemos concluir que los métodos `agregarPedido()` y `crearViaje()` no están realizando un manejo adecuado de las condiciones que deberían desencadenar un error cuando un cliente ya tiene un viaje en curso. Los últimos dos resultados nos indican que esos flujos funcionan correctamente, pero a la hora de indicarnos cuál es el error fallan la tarea.

Por último cabe destacar que el resto de los flujos no presentaron problema alguno y los módulos al final de los mismos no vieron afectado su comportamiento por los resultados de módulos en los niveles superiores.

● **Persistencia:**

Para probar la persistencia, tomamos en cuenta 2 escenarios, el primero cuando el archivo no existe y el segundo cuando el archivo existe. Para realizar los tests en primer lugar, decidimos separar los datos de `EmpresaDTO` para aislar cada aspecto individual, lo que facilita la detección de posibles errores específicos. Luego los estructuramos de forma tal que la lectura y la escritura se prueben de forma simultánea, utilizando cada operación como punto de comparación para la otra, lo que nos permite verificar que los datos almacenados y recuperados coincidan.

Por otro lado, decidimos no realizar tests específicos para los métodos de lectura, escritura, apertura y cierre de archivos, ya que consideramos que estos métodos deben funcionar como una capa de abstracción sobre las funcionalidades ya testeadas de Java.

Finalmente, al ejecutar los tests, obtuvimos resultados exitosos en todas las pruebas realizadas, lo que confirma que el sistema de persistencia de datos opera de forma precisa y confiable, asegurando que `EmpresaDTO` puede ser almacenado y recuperado correctamente.

• **Test GUI:**

El proceso de testing de la GUI incluye varios casos de prueba para cubrir diferentes aspectos de la interfaz de usuario. Estos casos de prueba están diseñados para verificar el comportamiento de la aplicación bajo varios escenarios.

El principal objetivo de estas pruebas, es asegurar que el estado de la interfaz de usuario sea consistente con el modelo de datos. Además, verificamos que se muestren los mensajes de error apropiados para acciones o entradas inválidas y que las transiciones entre los diferentes paneles se realice de forma adecuada

Se decidió testear cada panel en dos partes: Una parte de habilitación y deshabilitación de botones y campos de texto, y la otra parte de comportamiento ante el click de botones. Al probar la habilitación y deshabilitación de botones, se considera rellenar los campos primero y luego eliminar el contenido para ver si los botones se deshabilitan correctamente nuevamente.

La importancia de la clase Robot

La clase Robot fue fundamental para el testeo de GUI, ya que permite automatizar eventos de usuario como clicks en el mouse y entradas de teclado. Esto es especialmente importante porque hay eventos que solo se activan cuando se interactúa físicamente con la interfaz, como escribir en un campo de texto o hacer click en un boto. La clase Robot permite simular estas acciones, asegurando que los test puedan replicar fielmente la interacción del usuario con la aplicación

La clase Robot proporciona métodos para mover el mouse, hacer click, presionar y soltar teclas, entre otros. Esto permite realizar pruebas más completas y realistas, asegurando que la aplicación responda correctamente a las acciones del usuario

Clases utilitarias

- GuiTestUtils: Esta clase proporciona métodos utilitarios para operaciones comunes de la GUI, como conseguir componentes por su nombre, establecer texto y realizar acciones como imitar la escritura por teclado o el click de un mouse. Ayuda a reducir la duplicación de código y mejorar el mantenimiento
- FalsoOptionPane: Esta clase se utiliza para simular el comportamiento de JOptionPane en las pruebas. Permite verificar que se muestran los mensajes correctos al usuario sin necesariamente mostrar los cuadros de diálogos reales. Esto es útil para probar el manejo de errores y la retroalimentación del usuario

Resultados Obtenidos

1. Durante el login de un administrador, en caso de que la contraseña se ingrese incorrectamente, el mensaje que surge en la ventana emergente es incorrecto. Se deduce que este problema surge ya que, como vimos en la sección de caja negra, se espera que se arroje la excepción PasswordErroneaException pero se arroja la excepción UsuarioNoExisteException
2. Siguiendo el caso anterior, al no detectar el nombre de usuario "admin" como un usuario valido y ya existente, se generan conflictos ya que permite que un usuario se registre con el nombre de usuario admin y no surge una ventana emergente con el mensaje de usuario ocupado
3. En el panel administrador

- a. Se espera que al intentar registrar un chofer con un DNI ya ocupado, se muestre una ventana comunicando el error, pero esto no sucede, y en cambio el chofer se agrega a la lista de choferes disponibles pero no a la lista de choferes totales
- b. Al dar de alta a un nuevo chofer, los JTextField del formulario no se resetean, quedan con el contenido del chofer recién agregado
- c. Al dar de alta un nuevo vehículo, los JTextField y la checkbox del formulario no se resetean, quedan con el contenido del vehiculo recién agregado
- d. El botón de nuevo vehículo no se deshabilita cuando se quiere crear un auto con cero plazas.

● Testeo de excepciones

- **ChoferNoDisponibleException:** Durante los test de caja negra (Escenario 4) e integración se comprobó que esta excepción tiene un mensaje incorrecto. Se esperaba que contenga el mensaje Mensajes. CHOFER_NO_DISPONIBLE ("El chofer no está disponible"), pero se encontró "El Pedido no figura en la lista".
- **ChoferRepetidoException:** No presenta errores
- **ClienteConPedidoPendienteException:** Esta excepción presento fallas en caja negra y test de Integración. La excepción se lanzó en el momento esperado, pero el contenido de su mensaje era incorrecto. Se esperaba que contenga el mensaje Mensajes.CLIENTE_CON_PEDIDO_PENDIENTE ("Cliente con pedido pendiente"), pero se encontró "Cliente con viaje pendiente"
- **ClienteConViajePendienteException:** Esta excepción se puede arrojar en el método agregarPedido y en crearViaje.
 - En el método agregarPedido, no se arroja cuando se espera y permite que un cliente con un viaje pendiente haga un pedido.
 - En el método crearViaje, ocurre lo mismo que en agregarPedido. No se detecta el error, por lo que no se lanza la excepción y permite que un cliente tenga dos viajes iniciados
- **ClienteNoExisteException:** No presenta errores a excepción de la situación mencionada en PasswordErroneaException
- **ClienteSinViajePendienteException:** No presenta errores
- **PasswordErroneaException:** Esta excepción se espera que se arroje en el método login. En caso de que un cliente ingrese contraseña incorrecta, la excepción funciona correctamente. A la hora de que un administrador intente hacer login, en caso de que este ingrese mal la contraseña, no se arroja la excepción como esperado, sino que se arroja ClienteNoExisteException
- **PedidoInexistenteException:** No presenta errores
- **SinVehiculoParaPedidoException:** Esta excepción se espera que se arroje en el método agregarPedido, cuando ningún vehículo del sistema es válido para el pedido. El error encontrado con esta excepción, se asume como consecuencia de que el método getPuntajePedido de los autos y combis no funciona correctamente y por este motivo no está detectando vehículos válidos como tal.
- **SinViajesException:** Esta excepción nunca se lanza. El único escenario en el que se puede arrojar esta excepción es en el método calificacionDeChofer, cuando el chofer no tiene viajes. Pero este escenario fue testeado y nunca se arrojó.

- **UsuarioNoExisteException:** No presenta errores
- **UsuarioYaExisteException:** No presenta errores
- **VehiculoNoDisponibleException:** No presenta errores
- **VehiculoNoValidoException:** Se lanza en una situación que debería ejecutarse sin fallas (Escenario 5, metodo crearViaje). Esto se considera consecuencia del incorrecto funcionamiento del metodo getPuntajePedido()
- **VehiculoRepetidoException:** Durante los test de caja negra (Escenario 2) se comprobó que esta excepción tiene el parámetro vehiculoExistente incorrecto, ya que este siempre es null.

● Conclusión

Trabajar en el testing de "Subí que te llevo" nos permitió abordar diferentes tipos de pruebas para un sistema de gestión de transporte de pasajeros. Cada prueba trajo sus propios desafíos y aprendizajes.

Nos encontramos con problemas al manejar excepciones y condiciones particulares, como evitar que un cliente o chofer ocupado cree nuevos viajes o pedidos. En las pruebas de integración y de interfaz, surgieron dificultades que requirieron estudiar a fondo el código y las interacciones entre los módulos.

Cada tipo de test aportó algo valioso: las pruebas de unidad identificaron errores específicos en funciones individuales; las pruebas de GUI, con la clase Robot, simularon acciones de usuarios y revelaron problemas de interacción; las pruebas de persistencia confirmaron que el sistema puede almacenar y recuperar datos de manera fiable.

Esta experiencia nos mostró la importancia del testing para asegurar que el software funcione correctamente y sea fácil de usar. Aprendimos a estructurar pruebas que cubran las necesidades reales de la aplicación y detecten problemas antes de que afecten a los usuarios, preparándonos mejor para futuros desarrollos y destacando la importancia de la calidad en el software.