UNIVERSIDAD NACIONAL de MAR DEL PLATA

FACULTAD DE INGENIERÍA

Trabajo Final:

Taller de Programación 1

Grupo 8 - Integrantes:

Santin, Francisco
Biscay, Federico
Rios, Lucas
Zubiarrain, Joaquin

Cátedra: Taller de Programación I

Propósito

El siguiente informe presenta los resultados más relevantes del testing aplicado al sistema de software **“Subí que te llevo”** proporcionado por la cátedra. Se proveyó el archivo ‘.jar’ del programa junto con su documentación, por lo que se llevaron a cabo pruebas de caja negra.

Estrategia de Prueba

Se diseñaron casos de prueba basados en la documentación, aplicando particiones de equivalencia, baterías de pruebas y escenarios.

Estructura del programa

La documentación del software permite ver la siguiente estructura.

Subí que te llevo: (raíz)
(paquetes)
- controlador
- excepciones
- modeloDatos
- modeloNegocio
- persistencia
- util
- vista

2. Criterio de Selección de Pruebas para el informe

Con el fin de mantener este informe conciso, se ha decidido mostrar en el mismo las pruebas sobre los métodos que contienen la lógica crítica, el resto de las tablas estan en el siguiente link: Tablas TP Taller

3. Diseño de Casos de Prueba

3.1 Modelo de Datos
Viaje:

Tabla de Particiones - getValor			
Dato de entrada	Descripción de la Clases de equivalencia	Aplica ? (cumple el contrato?)	Identificador de clase de equivalencia
---	---	---	---

Bateria de Pruebas					
Número de prueba	Escenario	Datos de entrada (Variaciones en Pedido)	Valor	Salida Esperada	Clases de equiv. que abarca
1	1	---	---	170	---
2	2	---	---	215	---
3	3	---	---	220	---
4	4	---	---	220	---
5	5	---	---	145	---

Escenarios		
Escenario	Descripción	Estado Interno relevante
1	Zona standard, sin mascota ni baul	pedido1, chofer1, vehiculo1, valorBase: 100
2	Zona sin asfaltar, sin mascota ni baul	pedido2, chofer1, vehiculo1, valorBase: 100
3	Zona peligrosa, sin mascota ni baul	pedido3, chofer1, vehiculo1, valorBase: 100
4	Zona standard, con mascota, sin baul	pedido4, chofer1, vehiculo1, valorBase: 100
5	Zona standard, sin mascota, con baul	pedido5, chofer1, vehiculo1, valorBase: 100

ChoferPermanente:

Tabla de Particiones - getSueldoBruto()			
Dato de entrada / Estado	Descripción de la Clase de Equivalencia	Aplica ? (cumple contrato)	ID
-	-	-	-

Bateria de Pruebas				
N° Prueba	**Escenario**	**Datos de entrada**	**Valor esperado (Cálculo)**	**ID Partición**
1	1	Chofer con antiguedad < 20	1390	1
2	2	Chofer con antiguedad igual a 20	2070	1
3	3	Chofer con antiguedad mayor a 20	2070	1
4	5	Chofer sin hijos	1250	1

Escenarios		
Escenario	Descripción	Estado Interno relevante
Escenario 1 (Base)	Chofer Permanente con antiguedad menor a 20	nombre= Juan Perez dni=111222 antiguedad = 5 años hijos = 2 sueldoBasico = 1000.0
Escenario 2 (Limite)	Chofer con antiguedad igual a 20	antiguedad = 20 años hijos = 0
Escenario 3 (Veterano)	Chofer con antigüedad superior al tope (>20).	antiguedad = 21 años hijos = 1 (o definido en test)
Escenario 4 (Sin Hijos)	Chofer con antigüedad estándar pero sin hijos.	antiguedad = 5 años hijos = 0

Auto:

Tabla de Particiones - getPuntajePedido(Pedido pedido)			
Dato de entrada	Descripción de la Clases de equivalencia	Aplica ?	Identificador
Pedido	Requiere uso de baúl (true)	Sí	1
Pedido	No requiere uso de baúl (false)	Sí	2
Pedido	Cantidad pasajeros <= Capacidad Auto	Sí	3
Pedido	Cantidad pasajeros > Capacidad Auto	No	4
Pedido	Pedido requiere mascota & Auto NO acepta	No	5

Pedido	Pedido requiere mascota & Auto SÍ acepta	Sí	6

Bateria de Pruebas				
Número de prueba	Escenario	Datos de entrada (Pedido)	Salida Esperada	Clases de equiv.
4 (ConBaul)	A	Pasajeros: 4, Baúl: true, Mascota: true	160 (Integer)	1 , 3 , 6
5 (SinBaul)	A	Pasajeros: 4, Baúl: false, Mascota: true	120 (Integer)	2 , 3 , 6
6 (SuperaCap)	A	Pasajeros: 5 (vs Capacidad 4)	null	4
7 (RechazaMascota)	B	Pasajeros: 2, Mascota: true	null	5

Escenarios		
Escenario	Descripción	Estado Interno relevante
Escenario A	Auto Acepta Mascotas	Auto: patente="1", plazas=4, mascota=true.
Escenario B	Auto Rechaza Mascotas	Auto: patente="1", plazas=4, mascota=false.

Empresa:

Tabla de Particiones - Archivo: EmpresaTestCrearViaje			
crearViaje(Pedido,Chofer, Vehiculo)			
Dato de entrada	**Descripción de la Clases de equivalencia**	**Aplica ?(cumple el contrato?)**	**Identificador de clase de equivalencia**
Pedido	El pedido no se encuentra registrado en la colección de pedidos de la empresa.	Sí	1
Chofer	El chofer no se encuentra en la lista de choferes desocupados.	Sí	2
Vehiculo	El vehiculo no se encuentra en la lista de vehículos desocupados.	Sí	3

Vehiculo	El vehiculo no cumple las condiciones del pedido (ej. plazas, baúl, mascota).	Sí	4
Pedido	El cliente asociado al pedido ya tiene un viaje en curso.	Sí	5
Pedido	El parámetro pedido es null.	No	6
Chofer	El parámetro chofer es null.	No	6
Vehiculo	El parámetro vehiculo es null.	No	6

3.2 Modelo de Negocios

Tabla de Particiones - Archivo: EmpresaTestCrearViaje			
crearViaje(Pedido,Chofer, Vehiculo)			
Dato de entrada	**Descripción de la Clases de equivalencia**	**Aplica ?(cumple el contrato?)**	**Identificador de clase de equivalencia**
Pedido	El pedido no se encuentra registrado en la colección de pedidos de la empresa.	Sí	1
Chofer	El chofer no se encuentra en la lista de choferes desocupados.	Sí	2
Vehiculo	El vehiculo no se encuentra en la lista de vehículos desocupados.	Sí	3
Vehiculo	El vehiculo no cumple las condiciones del pedido (ej. plazas, baúl, mascota).	Sí	4
Pedido	El cliente asociado al pedido ya tiene un viaje en curso.	Sí	5
Pedido	El parámetro pedido es null.	No	6
Chofer	El parámetro chofer es null.	No	6

Bateria de Pruebas - crearViaje(..)					
Número de prueba	**Escenario**	**Datos de entrada**	**Valor**	**Salida Esperada**	**Clases de equiv. que abarca**
1	Escenario 1	(pedido1, chofer1, vehiculo1)	Todos los s son válidos y están en el estado correcto.	El viaje se crea exitosamente. Se elimina el pedido de la lista de pendientes. El cliente1 ahora tiene un viaje iniciado. El chofer1 y vehiculo1 ya no están desocupados.	1
2	Escenario 1	(pedidoInexistente, chofer1, vehiculo1)	pedidoInexistente es un Pedido válido pero no registrado en la empresa.	Lanza PedidoInexistenteException.	2
3	Escenario 1	(pedido1, chofer2, vehiculo1)	chofer2 está registrado pero no disponible (ocupado en viaje1).	Lanza ChoferNoDisponibleException.	3
4	Escenario 1	(pedido1, chofer1, vehiculo2)	vehiculo2 está registrado pero no disponible (ocupado en viaje1).	Lanza VehiculoNoDisponibleException.	4
5	Escenario 1	(pedido1, chofer1, vehiculo3)	vehiculo3 (Moto) está disponible pero no es válido para pedido1 (que requiere baúl y 3 pasajeros).	Lanza VehiculoNoValidoException.	5
6	Escenario 1	(pedido2_cliente2, chofer1, vehiculo1)	Se crea un pedido2 para cliente2, quien ya tiene un viaje (viaje1) en curso.	Lanza ClienteConViajePendienteException.	6

Escenarios - crearViaje(...)		
Escenario	**Descripción**	**Estado Interno relevante**
Escenario 1	**Empresa con estado mixto: Contiene entidades en diferentes estados (disponibles, ocupadas, inválidas) para permitir la prueba de todos los casos de éxito y de error.**	- Cliente 1: Registrado, con pedido1 pendiente. - Cliente 2: Registrado, con viaje1 ya iniciado. - Chofer 1: Registrado y desocupado. - Chofer 2: Registrado pero ocupado (en viaje1). - Vehículo 1 (Auto): Registrado, desocupado y válido para pedido1. - Vehículo 2 (Combi): Registrado pero ocupado (en viaje1). - Vehículo 3 (Moto): Registrado, desocupado pero no válido para pedido

Tabla de Particiones - Archivo: EmpresaTestPagarYFinalizarViaje			
pagarYFinalizarViaje(int calificacion)			
Dato de entrada	Descripción de la Clases de equivalencia	Aplica ?(cumple el contrato?)	Identificador de clase de equivalencia
int calificacion	Calificación válida (0 a 5)	Sí	1
int calificacion	Calificación inválida (< 0)	No	2
int calificacion	Calificación inválida (> 5)	No	3

Bateria de Pruebas - pagarYFinalizarViaje()					
Número de prueba	**Escenario**	**Datos de entrada**	**Valor**	**Salida Esperada**	**Clases de equiv. que abarca**
Número de prueba	Escenario	Datos de entrada	Valor	Salida Esperada	Clases de equiv. que abarca

1	Escenario 1	calificacion	3	Lanza ClienteSinViajePendienteException. El mensaje debe coincidir con util.Mensajes.CLIENTE_SIN_VIAJE_PENDIENTE.	1, 4, 2007
2	Escenario 2	calificacion	5	No lanza excepción. 1. viajeIniciado.getCalificacion() retorna. 2. viajeIniciado.isFinalizado() retorna true. 3. empresa.getViajesIniciados() está vacío 4. empresa.getViajesTerminados() contiene el viajeIniciado.	1, 4, 2006

Escenarios -pagarYFinalizarViaje(int calificacion)		
Escenario	**Descripción**	**Estado Interno relevante**
Escenario	Descripción	Estado Interno relevante
Escenario 1	Empresa con Cliente logueado, sin viaje pendiente.	empresa.getUsuarioLogeado() es un Cliente. empresa.getViajesIniciados() no contiene a ese Cliente

Escenario 2	Empresa con Cliente logueado y un Viaje iniciado.	empresa.getUsuarioLogeado() es el Cliente C1. empresa.getViajesIniciados() contiene (C1, V1). empresa.getViajesTerminados() está vacío

3.4 Controlador

Tabla de Particiones - Nuevo Pedido			
Dato de entrada	Descripción de la Clases de equivalencia	Aplica ? (cumple el contrato?)	identificador de clase de equivalencia
Pedido (de la Vista)	Pedido es nulo o no existe en Empresa.pedidos.	si	1
Chofer (de la Vista)	Chofer es nulo o no existe en Empresa.choferesDisponibles.	si	2
Vehiculo (de la Vista)	Vehiculo es nulo o no existe en Empresa.vehiculosDisponibles.	si	3
Vehiculo, Pedido	Vehiculo no cumple los requisitos del Pedido (ej. plazas, mascota).	si	4
Cliente (del Pedido)	Cliente (dueño del Pedido) ya tiene un Viaje en curso.	si	5
Pedido, Chofer, Vehiculo, Cliente	Todos los datos son válidos, compatibles, y el cliente está libre.	si	6

Bateria de Pruebas				
Número de prueba	Datos de entrada	Valor (Entradas de la Vista + Estado Previo)	Salida esperada	Clases de equivalencia que abarca
1	Pedido, Chofer, Vehiculo (Vista)	P1, C1, V1 (Válidos, compatibles y disponibles) Estado=EscenarioConUnChofer (Cliente está libre)	Empresa.viajesIniciados.size() == 1 actualizar vista se invoca. No se muestra mensaje	6

2	Pedido (Vista)	Pedido=pedidoFalso (No está en Empresa.pedidos) (Chofer y Vehiculo válidos)	mensaje(PEDIDO_INEXISTENTE) actualizar vista NUNCA se invoca. viajesIniciados.size() == 0	1
3	Chofer (Vista)	Chofer=chofer (pero choferesDisponibles está vacío) (Pedido y Vehiculo válidos)	mensaje(CHOFER_NO_DISPONIBLE) actualizar vista NUNCA se invoca. viajesIniciados.size() == 0	2
4	Vehiculo (Vista)	Vehiculo=vehiculoFalso (No está en Empresa.vehiculosDisponibles) (Pedido y Chofer válidos)	mensaje(VEHICULO_NO_DISPONIBLE) actualizar vista NUNCA se invoca. viajesIniciados.size() == 0	3
5	Vehiculo (Vista), Pedido (Vista)	Vehiculo=vehiculoNoCumple (ej. 1 plaza) Pedido=pedido (ej. 2 plazas) (Chofer válido)	mensaje(VEHICULO_NO_VALIDO) actualizar vista NUNCA se invoca. viajesIniciados.size() == 0	4
6	Pedido (Vista)	Pedido=pedido2 Estado=Cliente de pedido2 ya tiene viaje1 iniciado	mensaje(CLIENTE_CON_VIAJE_PENDIENTE) actualizar vista NUNCA se invoca. viajesIniciados.size() == 1 (solo el viaje1)	5

Tabla de Particiones - Calificar y pagar			
Dato de entrada	Descripción de la Clases de equivalencia	Aplica ? (cumple el contrato?)	identificador de clase de equivalencia
-	-	-	-

Bateria de Pruebas				
Número de prueba	Datos de entrada	Valor (Entradas de la Vista + Estado Previo)	Salida esperada	Clases de equivalencia

				que abarca
1	cliente (logueado) calificacion (vista) Empresa	cliente="clienteViaje" calificacion=5 Estado=EscenarioCompleto (Cliente tiene viaje)	viajesIniciados.size() == 0 viajesTerminados.get(0).calificacion == 5 actualizar vista se invoca. mensaje NO se invoca.	1
2	cliente (logueado) Empresa	cliente="clienteLibre" Estado=EscenarioCompleto (Cliente sin viaje)	mensaje(CLIENTE_SIN_VIAJE_PENDIENTE) actualizar vista NO se invoca.	2

3.6 Persistencia

Tabla de Particiones - Archivo: TestPersistenciaBIN			
escribir()			
Dato de entrada	**Clase de equivalencia**	**Aplica**	**C#**
-	-	-	-

escribir()				
Nº	**Escenario**	**Datos de entrada**	**Salida esperada**	**CE**
1	E1	output abierto + objeto válido	Escritura exitosa	1
2	E2	escribir null	IOException o NPE según implementación	2
3	E3	escribir sin abrir output	IOException	3
4	E4	escribir objeto NO serializable	IOException	4

escribir()		
Escenario	**Descripción**	**Estado Interno relevante**
E1	Escritura válida con output abierto	output ≠ null, objeto válido
E2	Intento de escribir null	output ≠ null
E3	Intento de escribir sin abrir output	output = null
E4	Escribir un objeto no serializable	output ≠ null

UtilPersistencia:

Tabla de Particiones - Archivo: TestUtilPersistencia			
EmpresaDtoFromEmpresa()			
Dato de entrada	**Descripción de la Clases de equivalencia**	**Aplica ?(cumple el contrato?)**	**Identificador de clase de equivalencia**
-	-	-	-

EmpresaDtoFromEmpresa()				
Número de prueba	**Escenario**	**Datos de entrada**	**ValorSalida Esperada**	**Clases de equiv. que abarca**
1	E1	Empresa con datos	DTO con mismas estructuras	1,5
2	E2	Empresa con colecciones vacías	DTO vacío	2
3	E3	Alguna colección null	DTO con campos null	3
4	E4	usuarioLog == null	dto.usuarioLog == null	4
5	E5	usuarioLog != null	dto.usuarioLog == copia	5

EmpresaDtoFromEmpresa()		
Escenario	**Descripción**	**Estado Interno relevante**
E1	Empresa con todas las estructuras cargadas	todas las colecciones != null y con elementos
E2	Empresa con colecciones vacías	todas las colecciones inicializadas pero vacías
E3	Empresa con colecciones null	una o varias colecciones = null
E4	Empresa sin usuario logueado	empresa.usuarioLogeado = null
E5	Empresa con usuario logueado definido	empresa.usuarioLogeado != null

EmpresaDTO:

Tabla de Particiones - Archivo: testPersistenciaEmpresaDTO			
Constructor EmpresaDTO()			
Dato / Estado esperado	**Clase de Equivalencia**	**Aplica**	**C#**
-	-	-	-

Constructor EmpresaDTO()				
Nº	**Escenario**	**Datos de entrada**	**Salida esperada**	**CE**

1	E1	new EmpresaDTO()	Todas las colecciones vacías y no-null; usuarioLogeado = null	1, 2

Constructor EmpresaDTO()		
Escenario	**Descripción**	**Estado Interno relevante**
E1	Instanciar EmpresaDTO y verificar valores iniciales	colecciones vacías, no null; usuarioLogeado = null

4. Resultados de la Ejecución

A continuación se detallarán los errores encontrados por paquete y sus respectivos módulos internos. Se consideraron probadas las excepciones durante el testeo del resto de métodos que las arrojan.

Paquete controlador:

Tiene un único módulo llamado “controlador”. Los errores encontrados fueron:

`public void nuevoChofer()`

1) Dado un chofer repetido, no se invoca el mensaje CHOFER_YA_REGISTRADO

`public void nuevoPedido()`

1) Dado un pedido ingresado por una vista mock, el mismo es almacenado en el modelo, pero al recuperarlo la distancia recorrida en kilómetros no coincide con la ingresada.
2) Se intenta cargar un viaje para un cliente logueado con viajes pendientes. La vista debería mostrar un mensaje de error CLIENTE_CON_VIAJE_PENDIENTE, pero no se invoca.
3) Se intenta hacer lo mismo que en el caso 2) pero con un cliente con pedidos pendientes. No se invoca el mensaje CLIENTE_CON_PEDIDO_PENDIENTE

public void nuevoViaje()

1) Se prueba el iniciar un viaje para un chofer que no está disponible. No se invoca el mensaje CHOFER_NO_DISPONIBLE.
2) Se prueba iniciar un viaje para un chofer que ya tiene un viaje pendiente. No se invoca el mensaje CHOFER_CON_VIAJE_PENDIENTE

Paquete Modelo de Datos:

Chofer permanente

public double getSueldoBruto()

1) Se crea un chofer con mas de 20 años de experiencia, se calcula el sueldo con la fórmula y se invoca a getSueldoBruto, los sueldos no coinciden

Viaje

public double getValor()

1) Se crea un viaje con un chofer permanente, un auto con baul y una zona estandar, el valor base del viaje estaba seteado en 100, cuando se calculó el valor del viaje no coincidia con lo que tendria que dar (145)

2) Se crea un viaje con un chofer permanente, un auto y una zona peligrosa, el valor base del viaje estaba seteado en 100, cuando se calculó el valor del viaje no coincidia con lo que tendria que dar (220)

3) Se crea un viaje con un chofer permanente, un auto y una zona estandar y con mascota, el valor base del viaje estaba seteado en 100, cuando se calculó el valor del viaje no coincidia con lo que tendria que dar (220)

4) Se crea un viaje con un chofer permanente, un auto y una zona estandar y, el valor base del viaje estaba seteado en 100, cuando se calculó el valor del viaje no coincidia con lo que tendria que dar (215)

Auto

public double getValor()

1) Se crea un auto y un pedido, se calcula el puntaje del auto para ese pedido, debería dar 120 y da otra cosa

Paquete Modelo de Negocios:

Tiene una única clase Empresa.

public void agregarPedido(Pedido pedido)

```
throws SinVehiculoParaPedidoException,
ClienteNoExisteException,
ClienteConViajePendienteException,
ClienteConPedidoPendienteException
```

1) Se agrega un pedido para un cliente con viajes pendientes y el sistema no lanza la excepción ClienteConViajePendienteException.

public void agregarChofer(Chofer chofer)

```
throws ChoferRepetidoException
```

1) Se agrega un chofer con el mismo documento y el sistema no lanza la excepción ChoferRepetidoException

```
public double calificacionDeChofer(Chofer
chofer)
                                    throws
SinViajesException
```

1) Se califica un chofer sin viajes y el sistema no lanza la excepción SinViajesException

```
public void agregarVehiculo(Vehiculo vehiculo)
                        throws
VehiculoRepetidoException
```

1) Al agregar un vehículo repetido, se lanza la excepción VehiculoRepetidoException, pero el atributo de la excepción que referencia al vehículo es distinto del agregado.

```
public void crearViaje(Pedido pedido,
 Chofer chofer,
 Vehiculo vehiculo)
                 throws PedidoInexistenteException,
ChoferNoDisponibleException,
VehiculoNoDisponibleException,
VehiculoNoValidoException,
ClienteConViajePendienteException
```

1) Se crea un viaje para un cliente con un viaje pendiente y no se lanza la excepción ClienteConViajePendienteException

Paquete Persistencia:

PersistenciaBIN

public void abrirInput(String nombre)
throws IOException

1) Se abre un archivo null, deberia tirar excepción y no la tira

public void cerrarOutput()
throws IOException

2) Se cierra un archivo sin haberlo abierto, deberia tirar excepcion y no la tira

public void cerrarInput()
throws IOException

1) Se cierra un archivo sin haberlo abierto, deberia tirar excepcion y no la tira

UtilPersistencia

public Usuario getUsuarioLogeado()

1) Se crea una empresaDTO a partir de una empresa con ciertos atributos cuando se recupera la clase empresa desde la clase empresaDTO, y se obtiene el usuario logueado, los usuarios no coinciden

Paquete Vista:

Panel Cliente

testFormularioPedidoDeshabilitadoCuandoHayUnViajePendiente()

- Este caso de prueba de la vista no se puede testear por una excepcion que se lanza a la mitad de la ejecución: “El pedido no figura en la lista”

testPedidoCompletoYViajeCalificadoConExito()

- Este caso de prueba de la vista no se puede testear por una excepcion que se lanza a la mitad de la ejecución: “El pedido no figura en la lista”

Panel Administrador

testVisualizacionInfoChofer()

- Este caso de prueba de la vista el sueldo del chofer no se muestra donde deberia

testHabilitacionBotonChoferPermanente()

- Este caso de prueba de la vista el boton no se habilita con todos los datos correctos

testCreacionNuevoViajeHabilitado()

- Este caso de prueba de la vista el boton no se habilita al seleccionar los 3 elementos pertinentes

testRegistroChoferDuplicado()

- Este caso de prueba de la vista no se muestra el mensaje de chofer ya registrado

Conclusión

La realización de este trabajo práctico nos permitió darnos cuenta la verdadera importancia y dificultad del testing para lograr un producto de calidad. Hasta ahora veníamos tan solo aprendiendo a programar, y esto nos dió una visión más amplia de cómo se desarrolla el software verdadero en la industria. Resulta de gran utilidad saber que tan solo con la documentación del código se puede planear el testing desde fases tempranas de desarrollo, sin tener todo el código implementado.

En resumen, esta experiencia nos ayudó a consolidar nuestras habilidades para diseñar pruebas, anticipar fallos y colaborar mejor con el desarrollo. Nos llevamos la certeza de que una estrategia de testing bien estructurada es lo que marca la diferencia entre un software que simplemente funciona y uno que es confiable para el usuario.

Los errores detectados con pruebas de caja negra, y pendientes de corrección son:

Módulo	Errores
Controlador	7
Modelo de datos	6
Modelo de negocios	7
Persistencia	4
Vista	6

Sumando un total de 30 errores.