

**Taller de Programación I**

**Informe Trabajo Práctico**

**Profesores:**

* **Leonel Guccione**
* **Adolfo Tomas Spinelli**
* **Guillermo Lazzurri**

**Integrantes:**

* **Criado, Facundo**
* **Nieto, Iván Ezequiel**
* **Saladino, Juan Cruz**
* **San Pedro, Gianfranco**

**Repositorio GitHub:** [**https://github.com/ffacun2/TPTallerProgra1**](https://github.com/ffacun2/TPTallerProgra1)

**Introducción**

Se define al testing como un proceso para verificar y validar la funcionalidad de un programa o una aplicación de software con el objetivo de garantizar que el producto contenga la menor cantidad de defectos posibles. Consiste en la verificación dinámica del comportamiento de un programa sobre un conjunto finito de casos de prueba, apropiadamente seleccionados a partir del dominio de ejecución que usualmente es infinito, en relación con el comportamiento esperado.

El Testing no es una actividad que se piensa al final del desarrollo del software, sino que va paralelo a este. El programa se prueba ejecutando solo unos pocos casos de prueba dado que por lo general es física, económica y técnicamente imposible ejecutarlo para todos los valores de entrada posibles.

En este informe, se expondrán diferentes pruebas funcionales como “Pruebas unitarias de Caja Negra”, “Test de Persistencia”, “Test de Interfaces Graficas de Usuario” y “Test de Integración”, aplicadas a un sistema de gestión de viajes.

**Desarrollo**

**Test Unitario de Caja Negra**

Las pruebas de caja negra, también llamadas de comportamiento o Funcionales se basan en lo que se espera de un módulo, fundamentándose en la especificación de requerimientos y el contrato.

Para la realización de estas pruebas y poder realizarlas de forma automática se utilizó la librería “JUnit4” la cual permitió desarrollar conjuntos de pruebas de forma sencilla y sistemática.

Cabe destacar, que no se incluyó el testeo de los métodos getters, setters, toString, equals y hashcode, por no aportar datos relevantes respecto del método, ya que en su mayoría, fueron generados por el entorno de desarrollo.

Para este trabajo práctico se testean las clases:

* Paquete modelo.datos
  + Administrador
  + Auto
  + ChoferPermanente
  + ChoferTemporal
  + Cliente
  + Combi
  + Moto
  + Pedido
  + Viaje
* Paquete modelo.empresa
  + Empresa

*En el siguiente enlace se tiene una plantilla con los métodos de dichas clases y sus respectivas tablas de peticiones y batería de pruebas.*

***Enlace:*** [Modelo de Datos - Tabla de Particiones y Bateria de Prueba.xlsx](https://docs.google.com/spreadsheets/d/1Cf_HmT9a4-9ioSCL98LfO4Nq8k70Ph9o/edit?gid=907083773#gid=907083773)

*Las clases abstractas Chofer, Vehiculo y Usuario no se testearon individualmente debido a que si lo hicimos con las que las heredan.*

***Clase Auto***

***public Auto (string patente, int cantidadPlazas, boolean mascota)***

**Escenarios**

1. El auto permite mascota y capacidad para 3 pasajeros
2. El auto no permite mascota y máximo pasajeros 3

**Tabla de Particiones**

|  |  |  |  |
| --- | --- | --- | --- |
| **Dato de Entrada** | **Descripción de la Clase de equivalencia** | **¿Aplica?(cumple contrato)** | **Identificador de clase de equivalencia** |
| **Patente** | != NULL | Si | 1 |
| == NULL | No | 2 |
| **Cant. Plazas** | 0 < && < 5 | Si | 3 |
| < 0  > 5 | No | 4.1  4.2 |
| **Mascota** | True or False | Si | 5 |

**Batería de Prueba**

|  |  |  |  |
| --- | --- | --- | --- |
| **N° de Prueba** | **Dato de entrada** | **Salida esperada** | **clase equivalencia** |
| **1** | patente = “ABC 123”  cantPlazas = 3  mascota = false | auto creado correctamente | 1,3,5 |
| **2** | patente = “ABC 123”  cantPlazas = 1  mascota = true | auto creado correctamente | 1,3,5 |

***public Integer getPuntajePedido ( Pedido pedido)***

**Escenarios**

1. El auto no permite mascota y máximo pasajeros 3
2. El auto permite mascota y máximo pasajeros 2

**Tabla de Particiones**

|  |  |  |  |
| --- | --- | --- | --- |
| **Dato de entrada** | **Descripción de la clase de equivalencia** | **¿Aplica?(cumple contrato)** | **Identificador de la clase de equivalencia** |
| **Pedido** | == null | No | 1 |
| != null | Si | 2 |

**Batería de Pruebas**

|  |  |  |  |
| --- | --- | --- | --- |
| ***N° de Prueba*** | ***Datos de entrada*** | ***Salida esperada*** | ***Clase de equivalencia*** |
| Escenario 1 | | | |
| ***1*** | pedido != null  baul = true  mascota = false  cantPax = 3 | 40\*cantPax = 120 | 2 |
| ***2*** | pedido != null  baul = false  mascota = true  cantPax = 2 | null | 2 |
| Escenario 2 | | | |
| ***3*** | pedido != null  baul = false  mascota = true  cant = 2 | 30\*cantPax = 60 | 2 |

***Clase ChoferPermanente***

***public ChoferPermanente(String dni, String nombre,int anioIngreso,int cantidadHijos)***

***tabla de particiones***

|  |  |  |  |
| --- | --- | --- | --- |
| **Dato de entrada** | **Descripción de la clase de equivalencia** | **Aplica?(cumple contrato)** | **Identificador de la clase de equivalencia** |
| **DNI** | == null | No | 1 |
| != null | Si | 2 |
| **Nombre** | == null | No | 3 |
| != null | Si | 4 |
| **anioIngreso** | >1900  <3000 | Si | 5.1  5.2 |
| <1900  >3000 | No | 6.1  6.2 |
| **cantidadHijos** | >= 0 | Si | 7 |
| < 0 | No | 8 |

***Batería de pruebas***

|  |  |  |  |
| --- | --- | --- | --- |
| ***N° de Prueba*** | ***Datos de entrada*** | ***Salida Esperada*** | ***Clase de equivalencia*** |
| ***1*** | dni = “1234”  nombre = “jorge gomez”  anioIngreso = 2000  cantHijos = 0 | Crea un chofer Válido | 1,3,5,7 |
| ***2*** | dni = “1”  nombre = “a”  anioIngreso = 1990  cantHijos = 3 | Crea un chofer válido | 1,3,5,7 |

***public static double getSueldoBruto()***

***Tabla de particiones***

|  |  |  |  |
| --- | --- | --- | --- |
| ***Dato de entrada*** | ***Descripción de la clase de equivalencia*** | **¿Aplica?** | ***Identificador de clase de equivalencia*** |
| ***-*** |  |  |  |

***Batería de pruebas***

|  |  |  |  |
| --- | --- | --- | --- |
| ***N° de Prueba*** | ***Datos de entrada*** | ***Salida Esperada*** | ***Clase de equivalencia*** |
| **1** | anioIngreso = 2000 (antiguedad 24 años),  cantidadHijos = 2,  sueldoBasico = 1000 | Sueldo bruto = 1000 + (1000 \* 1) + (1000 \* 0.14) = 1000 + 1000 + 140 = 2140 |  |
| **2** | anioIngreso = 2015 (antiguedad 9 años),  cantidadHijos = 1,  sueldoBasico = 1000 | Sueldo bruto = 1000 + (1000 \* 0.45) + (1000 \* 0.7) = 1000 + 450 + 70 = 1520 |  |

***Clase ChoferTemporario:***

***public ChoferTemporario(String dni, String nombre)***

***Tabla de particiones***

|  |  |  |  |
| --- | --- | --- | --- |
| ***Dato de entrada*** | ***Descripción de la clase de equivalencia*** | **¿Aplica?** | ***Identificador de clase de equivalencia*** |
| **dni** | dni != null  dni con al menos un caracter | SI | 1 |
| dni == null  dni sin ningun caracter | NO |  |
| **nombre** | nombre != null  nombre con al menos un caracter | SI | 2 |
| nombre == null  nombre sin ningun caracter | NO |  |

***Batería de pruebas***

|  |  |  |  |
| --- | --- | --- | --- |
| ***N° de Prueba*** | ***Datos de entrada*** | ***Salida Esperada*** | ***Clase de equivalencia*** |
| ***1*** | dni = 123  nombre = mi nombre | crea un chofer temporario valido | 1, 2 |
| ***2*** |  |  |  |

***public static double getSueldoBruto()***

***Tabla de particiones***

|  |  |  |  |
| --- | --- | --- | --- |
| ***Dato de entrada*** | ***Descripción de la clase de equivalencia*** | **¿Aplica?** | ***Identificador de clase de equivalencia*** |
| ***-*** |  |  |  |

***Batería de pruebas***

|  |  |  |  |
| --- | --- | --- | --- |
| ***N° de Prueba*** | ***Datos de entrada*** | ***Salida Esperada*** | ***Clase de equivalencia*** |
| **1** | sueldoBasico = 1000 | Sueldo bruto = sueldo basico |  |

***3***

***Clase Cliente:***

***public Cliente(String nombreUsuario, String pass, String nombreReal)***

***Tabla de particiones***

|  |  |  |  |
| --- | --- | --- | --- |
| ***Dato de entrada*** | ***Descripción de la clase de equivalencia*** | **¿Aplica?** | ***Identificador de clase de equivalencia*** |
| **nombreUsuario** | nombreUsuario != null  nombreUsuario con al menos un caracter | SI | 1 |
| nombreUsuario == null  nombreUsuario sin ningun caracter | NO |  |
| **pass** | pass != null  pass con al menos un caracter | SI | 2 |
| pass == null  pass sin ningun caracter | NO |  |
| **nombreReal** | nombreReal != null  nombreReal con al menos un caracter | SI | 3 |
| nombreReal == null  nombreReal sin ningun caracter | NO |  |

***Batería de pruebas***

|  |  |  |  |
| --- | --- | --- | --- |
| ***N° de Prueba*** | ***Datos de entrada*** | ***Salida Esperada*** | ***Clase de equivalencia*** |
| ***1*** | nombreUsuario = usuario1  pass = password  nombreReal = mi nombre | crea un cliente valido | 1, 2, 3 |
|  |  |  |  |

***Clase Combi:***

**Escenarios:**

1. Combi con capacidad de llevar mascotas
2. Combi sin capacidad para llevar mascotas

***Escenario 1:***

***public Combi(String patente, int cantidadPlazas, boolean mascota)***

***Tabla de particiones***

|  |  |  |  |
| --- | --- | --- | --- |
| ***Dato de entrada*** | ***Descripción de la clase de equivalencia*** | **¿Aplica?** | ***Identificador de clase de equivalencia*** |
| **patente** | patente != null | SI | 1 |
| patente == null | NO |  |
| **cantidadPlazas** | 4 < cantidadPlazas < 11 | SI | 2 |
| cantidadPlazas < 4  cantidadPlazas > 11 | NO |  |
| **mascota** | mascota == true  mascota == false | SI | 3 |
|  | NO |  |

***Batería de pruebas***

|  |  |  |  |
| --- | --- | --- | --- |
| ***N° de Prueba*** | ***Datos de entrada*** | ***Salida Esperada*** | ***Clase de equivalencia*** |
| ***1*** | patente = asd 123  cantidadPlazas = 6  mascota = true | crea una combi valida | 1, 2, 3 |
|  |  |  |  |

***public Integer getPuntajePedido(Pedido pedido)***

***Tabla de particiones***

|  |  |  |  |
| --- | --- | --- | --- |
| ***Dato de entrada*** | ***Descripción de la clase de equivalencia*** | **¿Aplica?** | ***Identificador de clase de equivalencia*** |
| **pedido** | pedido != null | SI | 1 |
| pedido == null | NO |  |

***Batería de pruebas***

|  |  |  |  |
| --- | --- | --- | --- |
| ***N° de Prueba*** | ***Datos de entrada*** | ***Salida Esperada*** | ***Clase de equivalencia*** |
| ***1*** | pedido = new Pedido(new Cliente("usuario1","passwor, "minombre"),4,true,true,10,Constantes.ZONA\_PELIGROSA) | 10 \* 4 + 100 | 1 |
| ***2*** | pedido = new Pedido(new Cliente("usuario1","password","mi nombre"),4,false,false,10,Constantes.ZONA\_PELIGROSA) | 10 \* 4 | 1 |

***Escenario 2:***

***public Integer getPuntajePedido(Pedido pedido)***

***Tabla de particiones***

|  |  |  |  |
| --- | --- | --- | --- |
| ***Dato de entrada*** | ***Descripción de la clase de equivalencia*** | **¿Aplica?** | ***Identificador de clase de equivalencia*** |
| **pedido** | pedido != null | SI | 1 |
| pedido == null | NO |  |

***Batería de pruebas***

|  |  |  |  |
| --- | --- | --- | --- |
| ***N° de Prueba*** | ***Datos de entrada*** | ***Salida Esperada*** | ***Clase de equivalencia*** |
| ***1*** | pedido = new Pedido(new Cliente("usuario1","passwor, "minombre"),4,true,true,10,Constantes.ZONA\_PELIGROSA) | null | 1 |
|  |  |  |  |

***Clase Moto:***

**Escenarios:**

1. Se crea una moto valida y se realizan tres pedidos diferentes.

***public Moto(String patente)***

***Tabla de particiones***

|  |  |  |  |
| --- | --- | --- | --- |
| ***Dato de entrada*** | ***Descripción de la clase de equivalencia*** | **¿Aplica?** | ***Identificador de clase de equivalencia*** |
| **patente** | patente != null | SI | 1 |
| patente == null | NO |  |

***Batería de pruebas***

|  |  |  |  |
| --- | --- | --- | --- |
| ***N° de Prueba*** | ***Datos de entrada*** | ***Salida Esperada*** | ***Clase de equivalencia*** |
| ***1*** | patente = asd 123 | crea una moto valida | 1 |
|  |  |  |  |

***public Integer getPuntajePedido(Pedido pedido)***

***Tabla de particiones***

|  |  |  |  |
| --- | --- | --- | --- |
| ***Dato de entrada*** | ***Descripción de la clase de equivalencia*** | **¿Aplica?** | ***Identificador de clase de equivalencia*** |
| **pedido** | pedido != null | SI | 1 |
| pedido == null | NO |  |

***Batería de pruebas***

|  |  |  |  |
| --- | --- | --- | --- |
| ***N° de Prueba*** | ***Datos de entrada*** | ***Salida Esperada*** | ***Clase de equivalencia*** |
| ***1*** | pedido = new Pedido(new Cliente("usuario1","password","mi nombre"),1,false,false,10,Constantes.ZONA\_SIN\_ASFALTAR) | 1000 | 1 |
| ***2*** | pedido = new Pedido(new Cliente("usuario1","password","mi nombre"),2,false,false,10,Constantes.ZONA\_SIN\_ASFALTAR) | null | 1 |
| ***3*** | pedido = new Pedido(new Cliente("usuario1","password","mi nombre"),1,true,false,10,Constantes.ZONA\_SIN\_ASFALTAR) | null | 1 |
| ***4*** | pedido = new Pedido(new Cliente("usuario1","password","mi nombre"),1,false,true,10,Constantes.ZONA\_SIN\_ASFALTAR) | null | 1 |

***Clase Pedido:***

**Escenarios:**

1. Se crea un pedido y un clienta para poder realizar el pedido

cliente = new Cliente("usuario1","password","Mi nombre")

***public Pedido(Cliente cliente, int cantidadPasajeros, boolean mascota, boolean baul, int km, String zona)***

***Tabla de particiones***

|  |  |  |  |
| --- | --- | --- | --- |
| ***Dato de entrada*** | ***Descripción de la clase de equivalencia*** | **¿Aplica?** | ***Identificador de clase de equivalencia*** |
| **cliente** | cliente != null | SI | 1 |
| cliente == null | NO |  |
| **cantidadPasajeros** | >1 | SI | 2 |
| <1 | NO |  |
| **mascota** | mascota = true  mascota = false | SI | 3 |
| **baul** | baul = true  baul = false | SI | 4 |
| **km** | >= 0 | SI | 5 |
| < 0 | NO |  |
| **zona** | zona = Constantes.ZONA\_PELIGROSA  zona=Constantes.ZONA\_SIN\_ASFALTAR  zona=Constantes.ZONA\_STANDARD | SI | 6 |
| zona distinta a las anteriores | NO |  |

***Batería de pruebas***

|  |  |  |  |
| --- | --- | --- | --- |
| ***N° de Prueba*** | ***Datos de entrada*** | ***Salida Esperada*** | ***Clase de equivalencia*** |
| ***1*** | cliente = new Cliente("usuario1","password","Mi nombre"),  cantidadPasajeros = 2,  mascota = true,  baul = false,  km = 10,  zona = Constantes.ZONA\_SIN\_ASFALTAR | crea un pedido valido | 1, 2, 3, 4, 5, 6 |
|  |  |  |  |

***public Cliente getCliente()***

***Batería de pruebas***

|  |  |  |  |
| --- | --- | --- | --- |
| ***N° de Prueba*** | ***Datos de entrada*** | ***Salida Esperada*** | ***Clase de equivalencia*** |
| ***1*** |  | cliente |  |
|  |  |  |  |

***public boolean isMascota()***

***Batería de pruebas***

|  |  |  |  |
| --- | --- | --- | --- |
| ***N° de Prueba*** | ***Datos de entrada*** | ***Salida Esperada*** | ***Clase de equivalencia*** |
| ***1*** |  | true |  |
|  |  |  |  |

***public boolean isBaul()***

***Batería de pruebas***

|  |  |  |  |
| --- | --- | --- | --- |
| ***N° de Prueba*** | ***Datos de entrada*** | ***Salida Esperada*** | ***Clase de equivalencia*** |
| ***1*** |  | false |  |
|  |  |  |  |

***public String getZona()***

***Batería de pruebas***

|  |  |  |  |
| --- | --- | --- | --- |
| ***N° de Prueba*** | ***Datos de entrada*** | ***Salida Esperada*** | ***Clase de equivalencia*** |
| ***1*** |  | Constantes.ZONA\_SIN\_ASFALTAR |  |
|  |  |  |  |

***public int getCantidadPasajeros()***

***Batería de pruebas***

|  |  |  |  |
| --- | --- | --- | --- |
| ***N° de Prueba*** | ***Datos de entrada*** | ***Salida Esperada*** | ***Clase de equivalencia*** |
| ***1*** |  | 2 |  |
|  |  |  |  |

***Clase Viaje:***

**Escenarios:**

1. Se crea un cliente, un pedido, un chofer, un vehiculo y un viaje

cliente = new Cliente("usuario1","password","mi nombre");

pedido = new Pedido((Cliente)cliente,3, false, false, 10,Constantes.ZONA\_STANDARD);

chofer = new ChoferPermanente("123","mi nombre",2000,2);

vehiculo = new Moto("asd 123");

viaje = new Viaje(pedido,chofer, vehiculo);

1. Se crea un cliente, un pedido, un chofer, un vehiculo y un viaje

cliente = new Cliente("usuario1","password","mi nombre");

pedido = new Pedido((Cliente)cliente,6, true, false, 10,Constantes.ZONA\_PELIGROSA);

chofer = new ChoferPermanente("123","mi nombre",2000,2);

vehiculo = new Combi("asd 123", 10, true);

viaje = new Viaje(pedido,chofer, vehiculo);

1. Se crea un cliente, un pedido, un chofer, un vehiculo y un viaje

cliente = new Cliente("usuario1","password","mi nombre");

pedido = new Pedido((Cliente)cliente,3, true, true, 10,Constantes.ZONA\_SIN\_ASFALTAR);

chofer = new ChoferPermanente("123","mi nombre",2000,2);

vehiculo = new Auto("asd 123", 4, true);

viaje = new Viaje(pedido,chofer, vehiculo);

***Escenario 1:***

***public Viaje(Pedido pedido, Chofer chofer, Vehiculo vehiculo)***

***Tabla de particiones***

|  |  |  |  |
| --- | --- | --- | --- |
| ***Dato de entrada*** | ***Descripción de la clase de equivalencia*** | **¿Aplica?** | ***Identificador de clase de equivalencia*** |
| **pedido** | pedido != null | SI | 1 |
| pedido == null | NO |  |
| **chofer** | chofer != null | SI | 2 |
| chofer == null | NO |  |
| **vehiculo** | vehiculo != null | SI | 3 |
| vehiculo == null | NO |  |

***Batería de pruebas***

|  |  |  |  |
| --- | --- | --- | --- |
| ***N° de Prueba*** | ***Datos de entrada*** | ***Salida Esperada*** | ***Clase de equivalencia*** |
| ***1*** | pedido = new Pedido((Cliente)cliente,3, false, false, 10,Constantes.ZONA\_STANDARD)  chofer = new ChoferPermanente("123","mi nombre",2000,2)  vehiculo = new Moto("asd 123") | crea un viaje valido, calificación en 0 y atributo finalizado en false | 1, 2, 3 |
|  |  |  |  |

***public Pedido getPedido()***

***Batería de pruebas***

|  |  |  |  |
| --- | --- | --- | --- |
| ***N° de Prueba*** | ***Datos de entrada*** | ***Salida Esperada*** | ***Clase de equivalencia*** |
| ***1*** |  | pedido generado anteriormente |  |
|  |  |  |  |

***public Chofer getChofer()***

***Batería de pruebas***

|  |  |  |  |
| --- | --- | --- | --- |
| ***N° de Prueba*** | ***Datos de entrada*** | ***Salida Esperada*** | ***Clase de equivalencia*** |
| ***1*** |  | chofer generado anteriormente |  |
|  |  |  |  |

***public Vehiculo getVehiculo()***

***Batería de pruebas***

|  |  |  |  |
| --- | --- | --- | --- |
| ***N° de Prueba*** | ***Datos de entrada*** | ***Salida Esperada*** | ***Clase de equivalencia*** |
| ***1*** |  | vehiculo generado anteriormente |  |
|  |  |  |  |

***public static void setValorBase(double valorBase)***

***Batería de pruebas***

|  |  |  |  |
| --- | --- | --- | --- |
| ***N° de Prueba*** | ***Datos de entrada*** | ***Salida Esperada*** | ***Clase de equivalencia*** |
| ***1*** | valorBase = 1000 | 1000 |  |
|  |  |  |  |

***public void finalizarViaje(int calificacion)***

***Tabla de particiones***

|  |  |  |  |
| --- | --- | --- | --- |
| ***Dato de entrada*** | ***Descripción de la clase de equivalencia*** | **¿Aplica?** | ***Identificador de clase de equivalencia*** |
| **calificacion** | calificación >= 0  calificacion <= 5 | SI | 1 |
| calificacion <0  calificacion >5 | NO |  |

***Batería de pruebas***

|  |  |  |  |
| --- | --- | --- | --- |
| ***N° de Prueba*** | ***Datos de entrada*** | ***Salida Esperada*** | ***Clase de equivalencia*** |
| ***1*** | calificacion = 5 | atributo calificación seteado en 5 y atributo finalizado = true | 1 |

***public double getValor()***

***Batería de pruebas***

|  |  |  |  |
| --- | --- | --- | --- |
| ***N° de Prueba*** | ***Datos de entrada*** | ***Salida Esperada*** | ***Clase de equivalencia*** |
| ***1*** |  | valorBase + valorBase\*(cant\_pasajeros+cant\_km)\*0.1 |  |
|  |  |  |  |

**Escenario 2:**

***public double getValor()***

***Batería de pruebas***

|  |  |  |  |
| --- | --- | --- | --- |
| ***N° de Prueba*** | ***Datos de entrada*** | ***Salida Esperada*** | ***Clase de equivalencia*** |
| ***1*** |  | valorBase + valorBase\*(cant\_pasajeros \*0.1+cant\_km\*0.2) + base\*(cant\_pasajeros\*0.1 + cant\_km\*0.2) |  |
|  |  |  |  |

**Escenario 3:**

***Batería de pruebas***

|  |  |  |  |
| --- | --- | --- | --- |
| ***N° de Prueba*** | ***Datos de entrada*** | ***Salida Esperada*** | ***Clase de equivalencia*** |
| ***1*** |  | valorBase + valorBase\*(cant\_pasajeros \*0.2+cant\_km\*0.15) + valorBase\*(cant\_pasajeros\*0.1 + cant\_km\*0.2) + valorBase \* (cant\_pasajeros\*0.1 + cant\_km\*0.05) |  |
|  |  |  |  |

**3.2 Package Modelo.Negocio**

Como podremos observar todos los test unitarios en esta etapa se realizaron dentro de la clase Empresa

3.2.1. Metodo double getTotalSalarios()

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Metodo double getTotalSalarios()** | | | | |
| **Tabla de Particiones** | | | | |
| **Condición** | **Clases Correctas** | **Clase** | **Clases Erróneas** | **Clase** |
| Cantidad de choferes registrados | Hay al menos un chofer registrado | **1** | No hay choferes registrados | **2** |
| Salarios de los choferes | Los choferes tienen salarios válidos (> 0) | **3** | Los choferes tienen salarios negativos  (teóricamente incorrecto,  si ocurre en la implementación) | **4** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Batería de prueba** | | | | | |
|  | **Entradas** | **Salida Esperada** | **Clases Cubiertas** | **Valores Límites** | **Salidas** |
| **Clases correctas** | Empresa con 2 choferes registrados:  salario1 = 1000, salario2 = 1500 | 2500.0 | 1, 3 | Choferes registrados |  |
| Empresa con 1 chofer registrado:  salario = 2000 | 2000.0 | 1, 3 | Un chofer registrado |  |
| **Clases Erróneas** | Empresa sin choferes registrados | 0.0 | 2 | Ningún chofer registrado |  |

**testGetTotalSalarios\_MultiplesChoferes()** Clases cubiertas: 1, 3

Este test se realizó en el Escenario Vacío, luego en el mismo se agregaron solo 2 choferes, uno de cada tipo, para poder calcular su salario individual y comparar la suma de estos con el resultado del test.

El cual arrojó una discrepancia, de la suma esperada de 5680 se obtuvo 4884.8

**testGetTotalSalarios\_UnChofer()** Clases cubiertas: 1, 3

Este test fue planteado en el Escenario Vacío para poder tener un mayor control, dentro del mismo agregamos un Chofer Permanente establece el sueldoBasico en 2000, el anioIngreso en 2010 y la CantidadHijos en 1.

Por lo tanto esperabamos que el resultado de **getTotalSalarios** fuese 3450, el salario de este único chofer. Sin embargo, el resultado obtenido fue de 3044.40 lo cual nos indicaba o que el método estaba realizando algo extraño con el monto o que uno de los métodos de los cuales depende no estaba haciendo lo que debía ( **testSueldoBruto()** de ChoferPermanente )

3.2.2. Método ArrayList<Vehiculo> vehiculosOrdenadosPorPedido(Pedido pedido)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Método ArrayList<Vehiculo> vehiculosOrdenadosPorPedido(Pedido pedido)** | | | | |
| **Tabla de Particiones** | | | | |
| **Condición** | **Clases Correctas** | **Clase** | **Clases Erróneas** | **Clase** |
| pedido | pedido != null | **1** | pedido == null |  |
| Vehículos disponibles | Vehículos habilitados para el pedido | **2** | Ningún vehículo habilitado | **3** |
| Vehículos con puntajes distintos | Vehículos tienen puntajes distintos | **4** | Todos los vehículos tienen  el mismo puntaje | **5** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Batería de prueba** | | | | | |
|  | **Entradas** | **Salida Esperada** | **Clases Cubiertas** | **Valores Límites** | **Salidas** |
| **Clases correctas** | Pedido válido con vehículos habilitados | ArrayList de vehículos ordenado por puntaje descendente | 1, 2, 4 | Vehículos habilitados |  |
| **Clases Erróneas** | pedido sin vehículos habilitados | ArrayList vacío | 1, 3 | Ningún vehículo  habilitado |  |
| pedido con vehículos con el mismo puntaje | ArrayList de vehículos  (sin orden específico) | 1, 2, 5 | Todos los vehículos  tienen el mismo puntaje |  |

**testVehiculosOrdenadosPorPedido\_ClaseCorrecta()** Clases cubiertas: 1, 2, 4

Este test se realizó en el Escenario 2 que cuenta con 3 vehículos de cada tipo con distintos atributos y ningún viaje o pedido pendiente, se creó un pedido de 2 pasajeros, sin baúl ni mascotas para el cual se esperaba 6 vehículos disponibles (solo las motos estarán imposibilitadas de tomarlo).

Se plantearon 3 asertos:

* Que el arrayList no podía ser null
* La cantidad de vehículos habilitados que debía contener el arrayList debia ser 6
* Los vehículos deben estar ordenados de forma descendente de acuerdo al puntaje de cada vehículo en relación al pedido

Para el 3er aserto calculamos de antemano el puntaje que debían tener los vehículos para comparar con el arrayList resultante y saber si estaban correctamente ordenados. De esta forma descubrimos que si bien el arrayList que devolvió el método contenía los 6 vehículos, estos no estaban ordenados correctamente debido a que los puntajes no coinciden con los esperados. Indicando que el fallo no era provocado por **vehiculosOrdenadosPorPedido** sino probablemente por **getPuntajePedido()** del modelo de datos.

**testVehiculosOrdenadosPorPedido\_SinVehiculosHabilitados()** Clases cubiertas: 1, 3

Este test se realizó también en el Escenario 2, como el anterior, pero en este caso creamos un pedido de 10 pasajeros que no puede ser satisfecho por ningún vehículo del escenario ya que la Combi con más plazas es de 9.

Se esperaba un arrayList vacío sin vehículos habilitados pero el que obtuvimos no estaba vacío.

3.2.3. Método void agregarPedido(Pedido pedido)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Método void agregarPedido(Pedido pedido)** | | | | |
| **Tabla de Particiones** | | | | |
| **Condición** | **Clases Correctas** | **Clase** | **Clases Erróneas** | **Clase** |
| pedido | pedido != null | **1** | pedido == null |  |
| Cliente registrado | cliente del pedido está registrado | **2** | cliente del pedido no está registrado | **3** |
| Cliente sin viaje pendiente | cliente no tiene viaje pendiente | **4** | cliente tiene un viaje pendiente | **5** |
| Cliente sin pedido pendiente | cliente no tiene pedido pendiente | **6** | cliente tiene un pedido pendiente | **7** |
| Vehículo disponible | vehículo disponible para el pedido | **8** | no hay vehículo disponible  para las características  solicitadas en el pedido | **9** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Batería de prueba** | | | | | |
|  | **Entradas** | **Salida Esperada** | **Clases Cubiertas** | **Valores Límites** | **Salidas** |
| **Clases correctas** | Pedido válido con cliente registrado  y sin viajes o pedidos pendientes y hay vehículo disponible | Pedido agregado correctamente | 1, 2, 4, 6, 8 | Todos los parámetros  válidos |  |
| **Clases Erróneas** | pedido con cliente no registrado | ClienteNoExisteException | 1, 3 | Cliente no registrado |  |
| pedido con cliente que tiene viaje pendiente | ClienteConViajePendienteException | 1, 2, 5 | Cliente con viaje  pendiente |  |
| pedido con cliente que tiene pedido pendiente | ClienteConPedidoPendienteException | 1, 2, 4, 7 | Cliente con pedido  pendiente |  |
| pedido con características que no pueden ser satisfechas por los vehículos | SinVehiculoParaPedidoException | 1, 2, 4, 6, 9 | No hay vehículos  disponibles |  |

**testAgregarPedido\_SinVehiculoDisponible()** Clases cubiertas: 1, 2, 4, 6, 9

Este test se llevó a cabo en el Escenario 2, se creó un pedido de 10 pasajeros esperando obtener la excepción SinVehiculoDisponible ya que la Combi de más plazas es de 9. Sin embargo no se lanzó la excepción, probablemente debido a que el método **vehiculosOrdenadosPorPedido** tambien esta fallando, generando un efecto cascada. Se le ofrecen a los pedidos vehículos libres que no satisfacen las condiciones de los viajes y por ende no se lanza la excepción.

**testAgregarPedido\_ClienteConPedidoPendiente()** Clases cubiertas: 1, 2, 4, 7

Se realizó el test en el Escenario 3, se invocó al “user4” y se verificó con un assertTrue que figura con un pedido pendiente como plantea el escenario. Luego se creó un nuevo pedido con dicho cliente y se invocó **agregarPedido** con ese mismo pedido.

Se lanzó la excepción esperaba ClienteConPedidoPendienteException pero el mensaje obtenido no corresponde a dicha excepción:: “Cliente con VIAJE pendiente”

**testAgregarPedido\_ClienteConViajePendiente()** Clases cubiertas: 1, 2, 4, 7

Se realizó el test en el Escenario 4, se invocó al “user4” y se verificó con un assertTrue que figura en un viaje iniciado como plantea el escenario. Luego se creó un nuevo pedido con dicho cliente y se invocó **agregarPedido** con ese mismo pedido.

Se esperaba ClienteConViajePendienteException al agregar un segundo pedido con un cliente que tiene un viaje pendiente pero no se lanzó ninguna excepción.

3.2.4. Método void crearViaje(Pedido pedido, Chofer chofer, Vehiculo vehiculo)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Método void crearViaje(Pedido pedido, Chofer chofer, Vehiculo vehiculo)** | | | | |
| **Tabla de Particiones** | | | | |
| **Condición** | **Clases Correctas** | **Clase** | **Clases Erróneas** | **Clase** |
| pedido | pedido != null | **1** | pedido == null |  |
| Pedido registrado | El pedido está registrado en el sistema | **2** | El pedido no está registrado  en el sistema | **3** |
| chofer | chofer != null | **4** | chofer == null |  |
| Chofer disponible | El chofer está disponible | **5** | El chofer no está disponible | **6** |
| vehículo | vehículo != null | **7** | vehículo == null |  |
| Vehículo disponible | El vehículo está disponible | **8** | El vehículo no está disponible | **9** |
| Vehículo válido para el pedido | El vehículo puede satisfacer el pedido | **10** | El vehículo no es válido para el pedido | **11** |
| Cliente sin viaje pendiente | Cliente no tiene viaje pendiente | **12** | Cliente tiene un viaje pendiente | **13** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Batería de prueba** | | | | | |
|  | **Entradas** | **Salida Esperada** | **Clases Cubiertas** | **Valores Límites** | **Salidas** |
| **Clases correctas** | pedido = Pedido válido,  chofer = Chofer disponible,  vehiculo = Vehiculo disponible y válido | Viaje creado correctamente | 1, 2, 4, 5, 7, 8, 10, 12 | Todos los parámetros  son válidos |  |
| **Clases Erróneas** | pedido = Pedido no registrado,  chofer = Chofer válido,  vehículo = Vehículo válido | PedidoInexistenteException | 1, 3, 4, 7 | Pedido no registrado |  |
| pedido = Pedido válido,  chofer = Chofer no disponible,  vehículo = Vehículo válido | ChoferNoDisponibleException | 1, 2, 4, 6, 7 | Chofer no disponible |  |
| pedido = Pedido válido,  chofer = Chofer disponible,  vehiculo = Vehiculo no disponible | VehiculoNoDisponibleException | 1, 2, 4, 5, 7, 9 | Vehículo no disponible |  |
| pedido = Pedido válido,  chofer = Chofer disponible,  vehiculo = Vehículo no válido para el pedido | VehiculoNoValidoException | 1, 2, 4, 5, 7, 8, 11 | Vehículo no válido  para el pedido |  |
| pedido = Pedido válido,  chofer = Chofer disponible,  vehiculo = Vehiculo válido,  cliente con viaje pendiente | ClienteConViajePendienteException | 1, 2, 4, 5, 7, 8, 10, 13 | Cliente con viaje  pendiente |  |

**testCrearViaje\_ChoferNoDisponible()** Clases cubiertas: 1, 2, 4, 5

Se planteó el test en el Escenario 3 el cual cuenta con los mismos elementos base que el Escenario 2 pero con la diferencia de que hay varios pedidos pendientes.

Primero se vació el arrayList de choferes desocupados de la empresa para asegurarnos de que no hubiese ningún chofer disponible y luego procedimos a implementar crearViaje con un pedido, vehículo y chofer del escenario.

Se lanzó la excepción esperada pero en vez de contener el mensaje “El chofer no está disponible”, la excepción contenía “El pedido no figura en la lista”

**testCrearViaje\_ClienteConViajePendiente()** Clases cubiertas: 1, 2, 4, 5, 7, 8, 10, 13

Este test se realizó en el Escenario 4, similar al Escenario 3 pero en vez de viajes pendientes disponemos de viajes en curso. Creamos un nuevo pedido con “user4” que se encuentra en un viaje en curso e invocamos **crearViaje** con dicho pedido y un chofer y vehículos disponibles.

Se esperaba el lanzamiento de ClienteConViajePendienteException al crear un viaje con un cliente que ya tiene un viaje en curso pero no se lanzó excepción alguna.

3.2.5. Método double calificación Chofer(Chofer chofer)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Método double calificacionDeChofer(Chofer chofer)** | | | | |
| **Tabla de Particiones** | | | | |
| **Condición** | **Clases Correctas** | **Clase** | **Clases Erróneas** | **Clase** |
| chofer | chofer != null | **1** | chofer == null |  |
| Chofer con viajes | El chofer ha realizado al menos un viaje | **2** | El chofer no ha realizado ningún viaje | **3** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Batería de prueba** | | | | | |
|  | **Entradas** | **Salida Esperada** | **Clases Cubiertas** | **Valores Límites** | **Salidas** |
| **Clases correctas** | chofer = Chofer con 3 viajes, calificaciones = [5, 4, 3] | 4.0 | 1, 2 | Calificaciones promedio  válidas |  |
| chofer = Chofer con 1 viaje, calificación = 5 | 5.0 | 1, 2 | Calificación única válida |  |
| **Clases Erróneas** | chofer = Chofer con 0 viajes | SinViajesException | 1, 3 | Ningún viaje realizado |  |

**testCalificacionDeChofer\_ChoferSinViajes()** Clases cubiertas: 1, 3

Se realizó el test en el Escenario 5 donde se han finalizado 9 viajes y se solicitó la calificación del chofer con dni "77777777" que no ha realizado ningún viaje.

Se esperaba un SinViajesException al intentar obtener la calificación de un chofer sin viajes pero no se lanzó ninguna excepción.

**testCalificacionDeChofer\_ChoferConTresViajes()** Clases cubiertas: 1, 2

Este test también se realizó en el Escenario 4 pero esta vez se solicitó la calificación promedio del chofer con dni "11111111" que tiene 3 viajes en su historial con calificaciones de 5, 4 y 3.

Se esperaba un promedio de 4 pero el resultado fue 1.66

**Test de Persistencia**

Consiste en validar que los datos que se almacenan en una base de datos o algún otro almacenamiento (archivo binario en este caso) puedan ser recuperados correctamente. Asegura que las operaciones de lectura y escritura funcionen como corresponda. Para el trabajo práctico se testeó las siguientes clases:

* Paquete persistencia
  + EmpresaDTO
  + UtilPersistencia
  + PersistenciaBin

enlace al documento Excel:  [Persistencia - Tabla de Particiones y Bateria de Prueba](https://docs.google.com/spreadsheets/d/1fQOBbSmh02QYtEpFUoD3G9APSrNex68xc9I58m0DZrE/edit?gid=1250545647#gid=1250545647)

**Test de Integración**

Tiene como objetivo verificar que diferentes módulos o componentes funcionen correctamente cuando se integran entre sí. Este test lo realizamos al controlador utilizando el framework Mockito que nos permitio imitar el comportamiento de la ventana. Gracias a esto logramos testear la comunicación de la ventana con los métodos de empresa. Para el trabajo práctico se testeo la siguiente clase:

* Paquete controlador:
  + Controlador

Para los casos que se lancen excepciones, el mensaje es mostrado por un objeto que implementa IOptionPane, por lo que se procedió a crear una clase que implemente dicha interfaz para poder capturar el mensaje y poder compararlo con el mensaje correcto.

Utilizamos dos escenario distintos:

* Escenario 1: Controlador con escenario empresa vacia, lista choferes vacio, lista choferes disponible vacio, lista vehiculos vacio, lista vehiculos disponibles vacio, lista clientes vacio, lista pedidos vacio, lista viajes iniciados vacio, lista viajes terminados vacio
* Escenario 2: Escenario con empresa con elementos en las listas

\* lista chofer con dos choferes:

\* 1.Permanente

\* 2.Temporario (viaje iniciado)

\* lista chofer disponible con el chofer Permanente, el temporario tiene viaje iniciado

\* lista vehiculo con :

\* 1. auto 4 lugares admite mascota

\* 2.combi 6 lugares no permite mascota

\* 3.moto (moto viaje iniciado)

\* lista vehiculo disponible inicialmente con los 2 vehiculos

\* lista clientes con 3 clientes

\* 1-con pedido pendiente

\* 2-sin pedido ni viaje

\* 3-con viaje iniciado

\* lista pedidos con 1 pedido hecho por 1 de los clientes

\* lista viajes iniciados con el viaje de uno de los clientes

\* lista viajes terminado vacia

**Test de GUI**

Se enfoca en verificar que los elementos visuales y las interacciones de la interfaz funcionen correctamente (ventanas, botones, formularios, etc). Para este test se utilizó la clase Robot que es parte de Java AWT que nos permitio simular la entrada del usuario, como el teclado y el mouse, de forma programática.

[Enlace del reporte de los test](TP/target/reports/surefire.html)

**Conclusión**

Gracias al contenido aprendido durante la cursada de la materia hemos logrado identificar áreas críticas en la aplicación, resaltando tanto los módulos que funcionan correctamente como aquellos que requieren ajustes.

El trabajo nos permitió familiarizarnos más con la parte de testing de un código de una forma más práctica, porque aunque ya habíamos visto muchos conceptos teóricos a la hora de llevarlos a la práctica nos encontramos con algunas dificultades. Tratamos de llevar cierta organización para implementar las diferentes técnicas de testing (estáticas y dinámicas) y a la hora de ir realizando los diferentes testeos de cada uno de los métodos.

Este trabajo nos llevó tiempo y fue crucial la coordinación de los integrantes del grupo, los que nos proporcionó experiencia en trabajos grupales. La dificultad fue moderada, ya que el pensar todos los posibles escenarios y casos de prueba para poder testear el funcionamiento correcto de cada parte del código nos resultó algo bastante complejo.