TP FINAL Taller de Programación I

Integrantes Grupo 8:

- Jose María Gutierrez Paredes
- Tomás Civiero
- Franco Vallone

Introducción

En este trabajo se busca aplicar los conocimientos vistos a lo largo de la materia llamada “Taller de Programación” para encontrar errores dentro del sistema. El cual permite a los clientes realizar un pedido para poder viajar, permite registrar a choferes y vehículos los cuales el administrador asignará a cada pedido para poder realizar el viaje y además, le permitirá al cliente pagar, calificar al chofer y finalizar el viaje.

Se aplicarán tanto técnicas de caja negra como de caja blanca, manuales y automatizadas.

Las pruebas de caja negra consisten en encontrar errores sin leer el código del proyecto, si no que, a partir, del contrato y la SRS se busca que con las entradas ingresadas el sistema responda con las salidas esperadas. Si no es el caso, entonces el sistema tendrá un error ya que no estaría respetando lo que se debe construir. También se realizaron las pruebas de integración en donde se verifica que, luego de haber testeado de manera unitaria cada método, funcione correctamente la interacción entre los módulos.

Las pruebas de caja de blanca se realizaron luego, las cuales consisten en probar todas las líneas del código que no han sido alcanzadas por las pruebas de caja negra. Para esto se arma para cada método el diagrama de flujo y así poder elegir los caminos necesarios determinando así qué parámetros y estados de los objetos permiten alcanzarlas.

Tanto caja negra como caja blanca fueron pruebas manuales. Pero no es el caso del testeo de GUI, ya que fue automatizado, empleando una librería de java que permite interactuar con la interfaz gráfica y así poder verificar el correcto funcionamiento de botones, campos de texto, listas, cambio de ventanas, etc.

Las librerías usadas para realizar los testeos fueron: JUnit 4 para los tests en eclipse, mockito para los mocks en las pruebas de integración y el controlador, y finalmente robot para el testeo de la GUI.

Caja negra

Pruebas unitarias

Clase: Empresa

Método: **public void agregarChofer(Chofer chofer) throws ChoferRepetidoException**

Precondiciones: El parámetro chofer es distinto de null.

Función: Agrega un chofer al HashMap de choferes si no está repetido.

tabla de particiones en clases de equivalencia			
Dato de entrada	Descripción	¿cumple el contrato ?	identificador
chofer	instancia de chofer	si	1
	null	no	2

escenarios	
num escenario	descripción
1	hashmap choferes vacío
2	con chofer(dni = "1")

batería de pruebas					
num prueba	Escenario	Datos de entrada	Valor	Salida Esperada	clases de equiv.
1	1	Chofer	{dni = "1", nombre="a"}	agrega chofer al hashmap de choferes	1
2	2	Chofer	{dni = "1", nombre="a"}	ChoferRepetidoException	1
3	2	Chofer	{dni = "2", nombre="a"}	agrega chofer al hashmap de choferes	1

Errores encontrados: Agrega al chofer aunque ya haya uno registrado con el mismo dni.

Clase: Empresa

Método: **public void agregarCliente(String usuario,String pass,String nombreReal) throws UsuarioYaExisteException**

Precondiciones: Los parámetros usuario, pass y nombreReal son diferentes de null y tienen al menos un carácter.

Función: Agrega un Cliente al HashMap de clientes si no está repetido.

tabla de particiones en clases de equivalencia			
Dato de entrada	Descripción	¿cumple el contrato ?	identificador
usuario	al menos un carácter	si	1
	null	no	2
	vacio	no	3
pass	al menos un carácter	si	4
	null	no	5
	vacio	no	6
nombreReal	al menos un carácter	si	7
	null	no	8
	vacio	no	9

escenarios	
num escenario	descripción

1	hashmap clientes vacío
2	con cliente(nombreUsuario = "a")

batería de pruebas					
num prueba	Escenario	Datos de entrada	Valor	Salida Esperada	clases de equiv.
1	1	usuario	"a"	agrega al cliente al hashmap de clientes	1,4,7
		pass	"b"		
		nombreReal	"c"		
2	1	usuario	"admin"	UsuarioYaExisteException	1,4,7
		pass	"b"		
		nombreReal	"c"		
3	2	usuario	"a"	UsuarioYaExisteException	1,4,7
		pass	"b"		
		nombreReal	"c"		
4	2	usuario	"b"	agrega al cliente al hashmap de clientes	1,4,7
		pass	"b"		
		nombreReal	"c"		
5	2	usuario	"admin"	UsuarioYaExisteException	1,4,7
		pass	"b"		
		nombreReal	"c"		

Errores encontrados: Registra a un usuario con el nombre de usuario de “admin” cuando en realidad este le pertenece al administrador del sistema.

Clase: Empresa

Método: **public void agregarPedido(Pedido pedido) throws SinVehiculoParaPedidoException, ClienteNoExisteException,ClienteConViajePendienteException,ClienteConPedidoPendienteException**

Precondiciones: El parámetro pedido es diferente de null.

Función: Se agrega al HashMap pedidos.

tabla de particiones en clases de equivalencia			
Dato de entrada	Descripción	¿cumple el contrato ?	identificador
pedido	instancia de pedido	si	1
	null	no	2

escenarios	
num escenario	descripción
1	con auto("aaa111",3,true), moto("mmm111",1), con cliente(nombre_usuario="pepe123",nombre_usuario="alfa"), con choferTemporario("123"),choferTemporario("777") con pedido("alfa",1,false,false,5,"ZONA_PELIGROSA")
2	con auto("aaa111",3,true), moto("mmm111",1), con cliente(nombre_usuario="pepe123",nombre_usuario="alfa"), con choferTemporario("123"),choferTemporario("777"), con pedido1("alfa",1,false,false,5,"ZONA_PELIGROSA"), con viaje(pedido1,choferTemporario,moto)

batería de pruebas					
num prueba	Escenario	Datos de entrada	Valor	Salida Esperada	clases de equiv.

1	1	pedido	{cliente("pepe123"),3,true,true,10,"ZONA_STANDARD"}	agrega al pedido al hashmap de pedidos	1
2	1	pedido	{cliente("pepe123"),7,true,true,10,"ZONA_STANDARD"}	SinVehiculoParaPedidoException	1
3	1	pedido	{cliente("zzz"),7,true,true,10,"ZONA_STANDARD"}	ClienteNoExisteException	1
4	1	pedido	{cliente("alfa"),3,true,true,10,"ZONA_STANDARD"}	ClienteConPedidoPendienteException	1
5	2	pedido	{cliente("pepe123"),3,true,true,10,"ZONA_STANDARD"}	agrega al pedido al hashmap de pedidos	1
6	2	pedido	{cliente("pepe123"),1,false,false,10,"ZONA_STANDARD"}	SinVehiculoParaPedidoException	1
7	2	pedido	{cliente("alfa"),3,true,true,10,"ZONA_STANDARD"}	ClienteConViajePendienteException	1

Errores encontrados: Agrega un pedido cuando el cliente que lo solicitó está en viaje.

Clase: Empresa

Método: **public void agregarVehiculo(Vehiculo vehiculo) throws VehiculoRepetidoException**

Precondiciones: El atributo vehículo es distinto de null.

Función: Agrega un vehículo al HashMap vehículos, si no está repetido.

tabla de particiones en clases de equivalencia			
Dato de entrada	Descripción	¿cumple el contrato ?	identificador
vehiculo	instancia de vehículo	si	1
	null	no	2

escenarios	
num escenario	descripción

1	hashmap vehículos vacío
2	con vehículo(patente = "a")

batería de pruebas					
num prueba	Escenario	Datos de entrada	Valor	Salida Esperada	clases de equiv.
1	1	vehiculo	{"a",4,true}	se agrega al hashmap de vehículos	1
2	2	vehiculo	{"a",4,true}	VehiculoRepetidoException	2
3	2	vehiculo	{"b",4,true}	se agrega al hashmap de vehículos	3

Clase: Empresa

Método: **public ArrayList<Vehiculo> vehiculosOrdenadosPorPedido(Pedido pedido)**

Precondiciones: El parámetro pedido es diferente de null..

Función: Devuelve un ArrayList de objetos de tipo Vehículo que contiene los vehiculas habilitados para el pedido en cuestión ordenados de forma descendente de acuerdo al puntaje de cada vehículo en relación al pedido.

tabla de particiones en clases de equivalencia			
Dato de entrada	Descripción	¿cumple el contrato ?	identificador
pedido	instancia de pedido	si	1
	null	no	2

escenarios	
num escenario	descripción
1	sin vehiculos
2	con moto("mmm111",1)

3	con moto("mmm111",1), auto("aaa111",3,true)
4	con moto("mmm111",1), auto("aaa111",3,true), combi("ccc111",8,false)

batería de pruebas					
num prueba	Escenario	Datos de entrada	Valor	Salida Esperada	clases de equiv.
1	1	pedido	pedido(1,false,false,5,"ZONA_SIN_ASFALTAR")	arrayList vacio	1
2	2	pedido	pedido(1,false,false,5,"ZONA_SIN_ASFALTAR")	arrayList solo moto	1
3	2	pedido	pedido(3,true,false,5,"ZONA_SIN_ASFALTAR")	arrayList vacio	1
4	3	pedido	pedido(1,false,false,5,"ZONA_SIN_ASFALTAR")	arrayList con moto y auto en ese orden	1
5	3	pedido	pedido(3,true,false,5,"ZONA_SIN_ASFALTAR")	arrayList solo auto	1
6	3	pedido	pedido(10,true,false,5,"ZONA_SIN_ASFALTAR")	arrayList vacio	1
7	4	pedido	pedido(1,false,false,5,"ZONA_SIN_ASFALTAR")	arrayList con moto,auto,combi en ese orden	1
8	4	pedido	pedido(3,true,false,5,"ZONA_SIN_ASFALTAR")	arrayList solo auto	1
9	4	pedido	pedido(3,false,false,5,"ZONA_SIN_ASFALTAR")	arrayList con auto y combi en ese orden	1
10	4	pedido	pedido(7,false,false,5,"ZONA_SIN_ASFALTAR")	arrayList solo combi	1
11	4	pedido	pedido(10,true,false,5,"ZONA_SIN_ASFALTAR")	arrayList vacio	1

Clase: Empresa

Método: **public boolean validarPedido(Pedido pedido)**

Precondiciones: El parámetro pedido es diferente de null.

Función: Indica si un pedido tiene al menos un vehículo registrado con las características necesarias para satisfacer el pedido.

tabla de particiones en clases de equivalencia			
Dato de entrada	Descripción	¿cumple el contrato ?	identificador
pedido	instancia de pedido	si	1
	null	no	2

escenarios	
num escenario	descripción
1	sin vehículos registrados
2	con moto(patente="mmm111")
3	con auto(patente="aaa111",3,false)
4	con auto(patente="aaa111",3,true) //pet friendly
5	con combi(patente="ccc111",8,false)
6	con combi(patente="ccc111",8,true) //pet friendly

batería de pruebas					
num prueba	Escenario	Datos de entrada	Valor	Salida Esperada	clases de equiv.
1	1	pedido	{cliente,1,false,false,10,"ZONA_STANDARD"}	FALSE	1
2	1	pedido	{cliente,2,false,false,10,"ZONA_STANDARD"}	FALSE	1
3	1	pedido	{cliente,5,false,false,10,"ZONA_STANDARD"}	FALSE	1

4	2	pedido	{cliente,1,false,false,10,"ZONA_STANDARD"}	TRUE	1
5	2	pedido	{cliente,2,false,false,10,"ZONA_STANDARD"}	FALSE	1
6	2	pedido	{cliente,5,false,false,10,"ZONA_STANDARD"}	FALSE	1
7	3	pedido	{cliente,1,false,false,10,"ZONA_STANDARD"}	TRUE	1
8	3	pedido	{cliente,2,false,true,10,"ZONA_STANDARD"}	TRUE	1
9	3	pedido	{cliente,2,true,true,10,"ZONA_STANDARD"}	FALSE	1
10	3	pedido	{cliente,4,false,false,10,"ZONA_STANDARD"}	FALSE	1
11	3	pedido	{cliente,5,true,false,10,"ZONA_STANDARD"}	FALSE	1
12	4	pedido	{cliente,1,false,false,10,"ZONA_STANDARD"}	TRUE	1
13	4	pedido	{cliente,2,true,true,10,"ZONA_STANDARD"}	TRUE	1
14	4	pedido	{cliente,3,false,false,10,"ZONA_STANDARD"}	TRUE	1
15	4	pedido	{cliente,4,true,true,10,"ZONA_STANDARD"}	FALSE	1
16	4	pedido	{cliente,5,true,false,10,"ZONA_STANDARD"}	FALSE	1
17	5	pedido	{cliente,1,false,false,10,"ZONA_STANDARD"}	TRUE	1
18	5	pedido	{cliente,4,false,true,10,"ZONA_STANDARD"}	TRUE	1
19	5	pedido	{cliente,7,false,true,10,"ZONA_STANDARD"}	TRUE	1
20	5	pedido	{cliente,7,true,true,10,"ZONA_STAN	FALSE	1

			DARD"}		
21	5	pedido	{cliente,9,false,true,10,"ZONA_STANDARD"}	FALSE	1
22	6	pedido	{cliente,1,false,false,10,"ZONA_STANDARD"}	TRUE	1
23	6	pedido	{cliente,4,false,true,10,"ZONA_STANDARD"}	TRUE	1
24	6	pedido	{cliente,4,true,true,10,"ZONA_STANDARD"}	TRUE	1
25	6	pedido	{cliente,7,false,true,10,"ZONA_STANDARD"}	TRUE	1
26	6	pedido	{cliente,7,true,true,10,"ZONA_STANDARD"}	TRUE	1
27	6	pedido	{cliente,9,false,true,10,"ZONA_STANDARD"}	FALSE	1

Errores encontrados: Devuelve que hay vehículo válido para el pedido cuando en realidad no lo hay (solo sucede con el tipo de vehículo “Combi”).

Clase: Empresa

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

Precondiciones: Los parámetros pedido, chofer y vehículo son distintos de null.

Función: Se agrega al hashmap de ViajesIniciados.

tabla de particiones en clases de equivalencia			
Dato de entrada	Descripción	¿cumple el contrato ?	identificador
pedido	pedido instanciado	si	1

	null	no	2
chofer	chofer instanciado	si	3
	null	no	4
vehiculo	vehículo instanciado	si	5
	null	no	6

escenarios	
num escenario	valores
1	Empresa con cliente1(usernames="a"), cliente2(usernames="b") chofer1("13608188","jorge"), chofer2("111","raul") vehiculo1("aaa111",4,true), vehiculo2("bbb111",4,true) en viaje y vehiculo3("ccc111",1,false) pedido1(cliente{nombre_usuario="a"}, 4, true,true,5,"ZONA_STANDARD"), pedido2(cliente2, 4, false, false, 5,"ZONA_STANDARD") viaje(pedido2,chofer2,vehiculo2)
2	Empresa con cliente1(usernames="a"), cliente2(usernames="b") sin choferes, vehiculo1("aaa111",4,true), vehiculo2("bbb111",4,true) en viaje y vehiculo3("ccc111",1,false) pedido1(cliente{nombre_usuario="a"}, 4, true,true,5,"ZONA_STANDARD"), pedido2(cliente2, 4, false, false, 5,"ZONA_STANDARD")
3	Empresa con cliente1(usernames="a"), cliente2(usernames="b") chofer1("13608188","jorge"), chofer2("111","raul"), vehiculo1("aaa111",4,true), vehiculo2("bbb111",4,true) en viaje y vehiculo3("ccc111",1,false), sin pedidos

batería de pruebas

num prueba	Escenario	Datos de entrada	Valor	Salida Esperada	clases de equiv.
1	1	pedido	{cliente{nombre_usuario="a"}, 4, true,true,5,"zona_standard"}	se crea el viaje y se agrega al hashmap de viajesIniciados	1,3,5
		chofer	{"13608188","jorge"}		
		vehiculo	{"aaa111",4,true}		
2	1	pedido	{cliente{nombre_usuario="a"}, 5, false,false,10,"zona_standard"}	PedidoInexistenteException	1,3,5
		chofer	{"13608188","jorge"}		
		vehiculo	{"aaa111",4,true}		
3	1	pedido	{cliente{nombre_usuario="b"}, 4, true,true,5,"zona_standard"}	ClienteConViajePendienteException	1,3,5
		chofer	{"13608188","jorge"}		
		vehiculo	{"aaa111",4,true}		
4	1	pedido	{cliente{nombre_usuario="a"}, 4, true,true,5,"zona_standard"}	ChoferNoDisponibleException	1,3,5
		chofer	{"111","raul"}		
		vehiculo	{"aaa111",4,true}		
5	1	pedido	{cliente{nombre_usuario="a"}, 4, true,true,5,"zona_standard"}	ChoferNoDisponibleException	1,3,5
		chofer	{"333","jorge"}		
		vehiculo	{"bbb111",4,true}		
6	1	pedido	{cliente{nombre_usuario="a"}, 4, true,true,5,"zona_standard"}	VehiculoNoDisponibleException	1,3,5
		chofer	{"13608188","jorge"}		
		vehiculo	{"bbb111",4,true}		

7	1	pedido	{cliente{nombre_usuario="a"}, 4, true,true,5,"zona_standard"}	VehiculoNoDisponibleException	1,3,5
		chofer	{"13608188","jorge"}		
		vehiculo	{"zzz",4,true}		
8	1	pedido	{cliente{nombre_usuario="a"}, 4, true,true,5,"zona_standard"}	VehiculoNoValidoException	1,3,5
		chofer	{"13608188","jorge"}		
		vehiculo	{"ccc111",1,false}		
9	2	pedido	{cliente{nombre_usuario="a"}, 4, true,true,5,"zona_standard"}	PedidoInexistenteException	1,3,5
		chofer	{"13608188","jorge"}		
		vehiculo	{"aaa111",4,true}		
10	3	pedido	{cliente{nombre_usuario="a"}, 4, true,true,5,"zona_standard"}	ChoferNoDisponibleException	1,3,5
		chofer	{"13608188","jorge"}		
		vehiculo	{"aaa111",4,true}		

Errores encontrados: Crea un viaje cuando el cliente ya está en uno.

Clase: Empresa

Método: **public void pagarYFinalizarViaje(int calificacion) throws ClienteSinViajePendienteException**

Precondiciones: Hay un usuario de tipo Cliente logeado en la Empresa. La calificación está comprendida entre 0 y 5 inclusive.

Función: Califica y termina un Viaje Pendiente del Cliente Logueado.

tabla de particiones en clases de equivalencia

Dato de entrada	Descripción	¿cumple el contrato ?	identificador
calificación	int entre 0 y 5	si	1
	int < 0	no	2
	int > 5	no	3
	null	no	3

escenarios	
num escenario	descripción
1	logeado con cliente sin viajes
2	logeado con cliente en viaje
3	logeado con cliente con viaje finalizado

batería de pruebas					
num prueba	Escenario	Datos de entrada	Valor	Salida Esperada	clases de equiv.
1	1	calificacion	3	ClienteSinViajePendienteException	1
2	2	calificacion	0	termina el viaje y califica al chofer con 0	1
3	2	calificacion	5	termina el viaje y califica al chofer con 5	1
4	3	calificacion	3	ClienteSinViajePendienteException	1

Clase: Empresa

Método: **public Usuario login(String usserName, String pass) throws UsuarioNoExisteException, PasswordErroneaException**

Precondiciones: Los parámetros usserName y pass son distintos de null y tienen al menos un carácter.

Función: Realiza el logeo de un Usuario (Cliente o Administrador) al sistema.

tabla de particiones en clases de equivalencia			
Dato de entrada	Descripción	¿cumple el contrato ?	identificador
usserName	al menos un carácter	si	1
	vacio	no	2
	null	no	3
pass	al menos un carácter	si	4
	vacio	no	5
	null	no	6

escenarios	
num escenario	descripción
1	sin usuarios registrados
2	con cliente(username="a",password="b")

batería de pruebas					
num prueba	Escenario	Datos de entrada	Valor	Salida Esperada	clases de equiv.
1	1	usserName	"a"	UsuarioNoExisteException	1,4
		pass	"b"		
2	1	usserName	"admin"	se logea el admin y lo devuelve	1,4
		pass	"admin"		
3	2	usserName	"a"	se logea el usuario y lo	1,4

		pass	"b"	devuelve	
4	2	usserName	"admin"	se logea el admin y lo devuelve	1,4
		pass	"admin"		
5	2	usserName	"a"	PasswordErronesException	1,4
		pass	"z"		
6	2	usserName	"b"	UsuarioNoExisteException	1,4
		pass	"b"		

Clase: Empresa

Método: **public void logout()**

Función: Cierra la sesión del usuario actual.

escenarios	
num escenario	descripción
1	sin usuario logeado
2	con usuario(username="a") logeado
3	con admin logeado

batería de pruebas		
num prueba	Escenario	Salida Esperada
1	1	no hace nada
2	2	el usuario deja de estar logeado
3	3	el admin deja de estar logeado

Clase: Empresa

Método: **public double getTotalSalarios()**

Función: Devuelve un double que representa la suma de los salarios de los choferes registrados.

escenarios	
num escenario	descripción
1	sin choferes
2	un chofer temporario
3	un chofer permanente(5 antigüedad y 2 hijos)
4	un temporario y un chofer permanente(5 antigüedad y 2 hijos)

batería de pruebas		
num prueba	Escenario	Salida Esperada
1	1	0
2	2	430000
3	3	597700
4	4	1027700

Clase: Empresa

Método: **public ArrayList<Viaje> getHistorialViajeChofer(Chofer chofer)**

Precondiciones: El parámetro chofer es distinto de null.

Función: ArrayList de objetos de tipo Viaje correspondiente a los viajes realizados por el chofer en cuestión.

tabla de particiones en clases de equivalencia			
Dato de entrada	Descripción	¿cumple el contrato ?	identificador
chofer	instancia de Chofer	si	1
	null	no	2

escenarios	
num escenario	descripción
1	chofer "1222" sin viajes, "737373" en viaje, "65656" con 2 viajes finalizados

batería de pruebas					
num prueba	Escenario	Datos de entrada	Valor	Salida Esperada	clases de equiv.
1	1	chofer	{dni="1222"}	devuelve ArrayList de viajes vacío	1
2	1	chofer	{dni="737373"}	devuelve ArrayList de viajes vacío	1
3	1	chofer	{dni="65656"}	devuelve ArrayList con 2 viajes	1
4	1	chofer	{dni="777"}	devuelve ArrayList de viajes vacío	1

Clase: Empresa

Método: **public ArrayList<Viaje> getHistorialViajeCliente(Cliente cliente)**

Precondiciones: El parámetro cliente es distinto de null.

Función: ArrayList de objetos de tipo Viaje correspondiente a los viajes realizados por el cliente en cuestión.

tabla de particiones en clases de equivalencia			
Dato de entrada	Descripción	¿cumple el contrato ?	identificador
cliente	instancia de Cliente	si	1
	null	no	2

escenarios	
num escenario	descripción
1	cliente "aatta" sin viajes, "ababa" en viaje, "dasodj" con 2 viajes finalizados

batería de pruebas					
num prueba	Escenario	Datos de entrada	Valor	Salida Esperada	clases de equiv.
1	1	cliente	{"aatta"}	devuelve ArrayList de viajes vacío	1
2	1	cliente	{"ababa"}	devuelve ArrayList de viajes vacío	1
3	1	cliente	{"dasodj"}	devuelve ArrayList con 2 viajes	1
4	1	cliente	{"no_existo"}	devuelve ArrayList de viajes vacío	1

Clase: Empresa

Método: **public double calificacionDeChofer(Chofer chofer) throws SinViajesException**

Precondiciones: El parámetro chofer es distinto de null.

Función: El promedio de las calificaciones de los viajes realizados por el chofer en cuestión.

tabla de particiones en clases de equivalencia			
Dato de entrada	Descripción	¿cumple el contrato ?	identificador
chofer	instancia de chofer	si	1
	null	no	2

escenarios	
num escenario	descripción
1	hashmap sin choferes
2	hashmap con chofer dni="1" sin viajes,chofer dni="2" un viaje realizado con calificación 1 y chofer dni="3" con calificaciones 1,3 y 5

batería de pruebas					
num prueba	Escenario	Datos de entrada	Valor	Salida Esperada	clases de equiv.
1	1	chofer	{dni="1"}	devuelve calificación 0.0	1
2	2	chofer	{dni="1"}	SinViajesException	1
3	2	chofer	{dni="2"}	devuelve calificación 1.0	1
4	2	chofer	{dni="3"}	devuelve calificación 3.0	1
5	2	chofer	{dni="4"}	devuelve calificación 0.0	1

Errores encontrados: No lanza la SinViajesException cuando se le pide la calificación a un chofer que no ha realizado ningún viaje. Calcula mal el promedio del puntaje de un chofer con calificaciones de 1, 3 y 5.

Clase: Vehiculo

Método: **public abstract Integer getPuntajePedido(Pedido pedido)**

Precondiciones: El parámetro pedido es distinto de null.

Función: Retorna el puntaje del vehículo en relación al pedido en cuestión. Si el vehículo no puede satisfacer la necesidades del pedido, se retorna null.

tabla de particiones en clases de equivalencia			
Dato de entrada	Descripción	¿cumple el contrato ?	identificador
pedido	instancia de pedido	si	1
	null	no	2

escenarios	
num escenario	descripción
1	moto registrada
2	auto registrado (cant_plazas = 3)
3	auto registrado (cant_plazas = 3) pet friendly
4	combi registrada (cant_plazas = 7)
5	combi registrada (cant_plazas = 7) pet friendly

batería de pruebas					
num prueba	Escenario	Datos de entrada	Valor	Salida Esperada	clases de equiv.
1	1	pedido	pedido{cliente,1,false,false,5,"ZONA_STANDARD"}	1000	1
2	1	pedido	pedido{cliente,2,false,false,5,"ZONA_STANDARD"}	null	1
3	2	pedido	pedido{cliente,1,false,false,5,"ZONA_STANDARD"}	30	1
4	2	pedido	pedido{cliente,2,false,false,5,"ZONA_STANDARD"}	60	1
5	2	pedido	pedido{cliente,2,true,false,5,"ZONA_STANDARD"}	null	1

6	2	pedido	pedido{cliente,2,false,true,5,"ZONA_STANDARD"}	80	1
7	2	pedido	pedido{cliente,4,false,false,5,"ZONA_STANDARD"}	null	1
8	3	pedido	pedido{cliente,1,false,false,5,"ZONA_STANDARD"}	30	1
9	3	pedido	pedido{cliente,2,false,false,5,"ZONA_STANDARD"}	60	1
10	3	pedido	pedido{cliente,2,true,false,5,"ZONA_STANDARD"}	60	1
11	3	pedido	pedido{cliente,2,false,true,5,"ZONA_STANDARD"}	80	1
12	3	pedido	pedido{cliente,4,false,false,5,"ZONA_STANDARD"}	null	1
13	4	pedido	pedido{cliente,1,false,false,5,"ZONA_STANDARD"}	10	1
14	4	pedido	pedido{cliente,2,false,false,5,"ZONA_STANDARD"}	20	1
15	4	pedido	pedido{cliente,2,true,false,5,"ZONA_STANDARD"}	null	1
16	4	pedido	pedido{cliente,5,false,false,5,"ZONA_STANDARD"}	50	1
17	4	pedido	pedido{cliente,5,false,true,5,"ZONA_STANDARD"}	150	1
18	4	pedido	pedido{cliente,5,true,false,5,"ZONA_STANDARD"}	null	1
19	4	pedido	pedido{cliente,10,false,false,5,"ZONA_STANDARD"}	null	1
20	5	pedido	pedido{cliente,1,true,false,5,"ZONA_STANDARD"}	10	1
21	5	pedido	pedido{cliente,2,true,false,5,"ZONA_STANDARD"}	20	1
22	5	pedido	pedido{cliente,5,true,false,5,"ZONA_STANDA	50	1

			RD"}		
23	5	pedido	pedido{cliente,5,true,true,5,"ZONA_STANDARD"}	150	1
24	5	pedido	pedido{cliente,10,true,false,5,"ZONA_STANDARD"}	null	1

Errores encontrados: Calcula mal ciertos puntajes y no devuelve null cuando la combi no satisface el pedido.

Clase: ChoferPermanente

Método: **public double getSueldoBruto()**

Función: El sueldo bruto se calcula incrementando el sueldo básico a partir de un plus por antigüedad y un plus por cantidad de hijos. Se incrementa un 5% del básico por cada año de antigüedad, hasta llegar a un máximo incremento de 100% que se logra a los 20 años. Se incrementa un 7% del básico por cada hijo.

escenarios	
num escenario	descripción
1	anio_ingreso=1900, 2 hijos
2	anio_ingreso=2024, 0 hijos
3	anio_ingreso=3000, 1 hijo

batería de pruebas		
num prueba	Escenario	Salida Esperada
1	1	1070000
2	2	500000
3	3	535000

Errores encontrados: Incrementa más allá del límite del 100% a los 20 años de antigüedad. Cuando el anio_ingreso es posterior a 2024 la antigüedad devuelve un valor

negativo que provoca una disminución en el valor del sueldo que en el caso de anio_ingreso = 3000 devuelve un sueldo negativo.

Clase: Viaje

Método: **public double getValor()**

Función: Retorna el valor del viaje.

escenarios	
num escenario	descripción
1	viaje con pedido{cliente,5,true,true,5,"ZONA_STANDARD"}
2	viaje con pedido{cliente,2,true,false,5,"ZONA_STANDARD"}
3	viaje con pedido{cliente,10,false,true,5,"ZONA_STANDARD"}
4	viaje con pedido{cliente,1,false,false,5,"ZONA_STANDARD"}
5	viaje con pedido{cliente,5,true,true,5,"ZONA_SIN_ASFALTAR"}
6	viaje con pedido{cliente,2,true,false,5,"ZONA_SIN_ASFALTAR"}
7	viaje con pedido{cliente,10,false,true,5,"ZONA_SIN_ASFALTAR"}
8	viaje con pedido{cliente,1,false,false,5,"ZONA_SIN_ASFALTAR"}
9	viaje con pedido{cliente,5,true,true,5,"ZONA_PELIGROSA"}
10	viaje con pedido{cliente,2,true,false,5,"ZONA_PELIGROSA"}
11	viaje con pedido{cliente,10,false,true,5,"ZONA_PELIGROSA"}
12	viaje con pedido{cliente,1,false,false,5,"ZONA_PELIGROSA"}

batería de pruebas		
num prueba	Escenario	Salida Esperada
1	1	4250
2	2	2900

3	3	3750
4	4	1600
5	5	5000
6	6	3350
7	7	5000
8	8	1950
9	9	4750
10	10	3400
11	11	4250
12	12	2100

Errores encontrados: Calcula mal la mayoría de valores de los viajes.

En los test de la clase controlador se utilizan mocks para simular el ingreso de datos por la GUI.

Clase: Controlador

Método: **public void nuevoViaje()**

Función: Se invoca al método crearViaje(...) de la clase Empresa con los parámetros obtenidos del atributo vista.

num escenario	descripción
1	Empresa con pedido1("ibu",1,false,false,5,"ZONA_STANDARD"), pedido2("teo",3,true,false,10,"ZONA_STANDARD") chofer1("12345"), chofer2("777"), chofer3("999") moto("mmm111"), auto("aaa111",3,true), combi("ccc111",7,false) cliente("teo"), cliente("ibu") y cliente("alfa") y viaje(pedido1,chofer1,combi)

batería de pruebas				
num prueba	Escenario	Datos de entrada	Valor	Salida Esperada
1	1	pedido	Pedido("teo",1,false,false,5,"ZONA_STANDARD")	se crea un nuevo viaje y se agrega al hashmap de viajesIniciados
		chofer	ChoferTemporario("777")	
		vehiculo	Auto("aaa111",3,true)	
2	1	pedido	Pedido(cliente,4,true,true,5,"ZONA_STANDARD")	no se crea el viaje y se lanza un cartel indicando que el vehículo no puede atender el pedido
		chofer	ChoferTemporario("777")	
		vehiculo	Moto("mmm111")	
3	1	pedido	Pedido("teo",1,false,false,10,"ZONA_STANDARD")	no se crea el viaje y se lanza un cartel indicando que el pedido no figura en la lista
		chofer	ChoferTemporario("777")	
		vehiculo	Moto("mmm111")	
4	1	pedido	Pedido("teo",1,false,false,5,"ZONA_STANDARD")	no se crea el viaje y se lanza un cartel indicando que el chofer no está disponible
		chofer	ChoferTemporario("12345")	
		vehiculo	Auto("aaa111",3,true)	
5	1	pedido	Pedido("teo",1,false,false,5,"ZONA_STANDARD")	no se crea el viaje y se lanza un cartel indicando que el vehículo no está disponible
		chofer	ChoferTemporario("777")	
		vehiculo	Combi("ccc111",10,true)	
6	1	pedido	Pedido("ibu",1,false,false,5,"ZONA_STANDARD")	no se crea el viaje y se lanza un cartel indicando que el cliente tiene un
		chofer	ChoferTemporario("777")	

		vehiculo	Auto("aaa111",3,true)	viaje pendiente

Errores encontrados: No indica que el chofer no está disponible para realizar el viaje. Tampoco indica que el cliente tiene un viaje pendiente por lo que no podrá empezar un nuevo viaje.

Clase: Controlador

Método: **public void nuevoVehiculo()**

Función: Se invoca al método agregarVehiculo(Vehiculo vehiculo) de la clase Empresa con los parámetros obtenidos del atributo vista.

escenarios	
num escenario	descripción
1	sin vehiculos en empresa
2	con vehiculos en empresa auto patente="bbb222"
3	con vehiculos en empresa combi patente="ccc111"

batería de pruebas				
num prueba	Escenario	Datos de entrada	Valor	Salida Esperada
1	1	tipo	"MOTO"	se agrega la moto a la colección de vehículos de la clase Empresa
		patente	"mmm111"	
2	2	tipo	"AUTO"	se agrega el auto a la colección de vehículos de la clase Empresa
		patente	"aaa111"	
		cant_plazas	4	
		isAptoMascota	TRUE	

4	2	tipo	"COMBI"	no se agrega y muestra cartel diciendo que el vehículo está repetido
		patente	"bbb222"	
		cant_plazas	10	
		isAptoMascota	FALSE	
3	3	tipo	"COMBI"	se agrega la combi a la colección de vehículos de la clase Empresa
		patente	"ccc111"	
		cant_plazas	10	
		isAptoMascota	FALSE	

Clase: Controlador

Método: **public void nuevoChofer()**

Función: Se invoca al método agregarChofer(Chofer chofer) de la clase Empresa con los parámetros obtenidos del atributo vista.

escenarios	
num escenario	descripción
1	sin choferes en Empresa
2	con chofer temporario dni="12345"

batería de pruebas				
num prueba	Escenario	Datos de entrada	Valor	Salida Esperada
1	1	tipo	"TEMPORARIO"	se agrega el chofer temporario a los
		nombre	"jorge"	

		dni	"12345"	choferes de Empresa
2	1	tipo	"PERMANENTE"	se agrega el chofer permanente a los choferes de Empresa
		nombre	"jorge"	
		dni	"12345"	
		anio_ingreso	2020	
		cant_hijos	5	
3	2	tipo	"PERMANENTE"	no se agrega y muestra cartel diciendo que el chofer está repetido
		nombre	"jorge"	
		dni	"12345"	
		anio_ingreso	2020	
		cant_hijos	5	

Errores encontrados: No detecta que hay un chofer con el mismo DNI ya registrado.

Clase: Controlador

Método: **public void calificarPagar()**

Función: Se invoca al método pagarYFinalizarViaje(int calificacion) de la clase Empresa utilizando el parámetro proporcionado por el atributo vista.

escenarios	
num escenario	descripción
1	cliente en viaje
2	cliente no está en viaje

batería de pruebas				
num prueba	Escenario	Datos de entrada	Valor	Salida Esperada
1	1	calificacion	0	termina el viaje y califica al chofer con 0
2	1	calificacion	5	termina el viaje y califica al chofer con 5
3	2	calificacion	3	muestra cartel diciendo que el cliente no tiene viaje pendiente

---

Clase: Controlador

Método: **public void login()**

Función: Se reciben los parámetros necesarios para realizar el login de un usuario del atributo vista.

escenarios	
num escenario	descripción
1	con Cliente registrado con usserName="pepe123",pass="123"

batería de pruebas				
num prueba	Escenario	Datos de entrada	Valor	Salida Esperada
1	1	usserName	pepe123	se logea el cliente
		pass	123	

2	1	usserName	pepe123	no se logea y muestra un cartel que la password es incorrecta
		pass	abc	
3	1	usserName	aaa	no se logea y muestra un cartel que el usuario no existe
		pass	123	

Clase: Controlador

Método: **public void leer()**

Función: Recupera los datos de la clase Empresa (singleton), conservados en el archivo indicado por el atributo fileName delegando la implementación en el atributo persistencia.

escenarios	
num escenario	descripción
1	fileName = "archivo.bin"
2	fileName = "./nueva/asdasd/dasd/asd/archivo.bin"

batería de pruebas		
num prueba	Escenario	Salida Esperada
1	1	recupera los datos de Empresa que se encontraban en el archivo con nombre del fileName
2	2	muestra cartel con error debido a que no se podia leer el archivo con ese nombre

---

Clase: Controlador

Método: **public void escribir()**

Función: Persiste los datos de la Clase Empresa (singleton) en el archivo indicado por el atributo fileName delegando la implementación en el atributo persistencia.

escenarios	
num escenario	descripción
1	fileName = "escribiendo.bin"
2	fileName = "./nueva/asdasd/dasd/asd/archivo.bin"

batería de pruebas		
num prueba	Escenario	Salida Esperada
1	1	escribe los datos de la empresa con el file name pasado
2	2	no puede escribir los datos de la empresa y muestra cartel indicando que no puede

---

Clase: Controlador

Método: **public void registrar()**

Función: Se reciben los parámetros necesarios para un nuevo cliente del atributo vista.

escenarios	
num escenario	descripción
1	con Cliente registrado con usserName="pepe123",pass="123"

batería de pruebas				
num prueba	Escenario	Datos de entrada	Valor	Salida Esperada
1	1	usserName	jorge123	se registra al cliente
		pass	12345	
		confirm	12345	
		nombreReal	jorge alvarez	
2	1	usserName	jorge123	no se registra y muestra un cartel diciendo que la password y la confirmación no coinciden
		pass	12345	
		confirm	abababb	
		nombreReal	jorge alvarez	
3	1	usserName	pepe123	no registra y muestra un cartel diciendo que el usuario ya existe
		pass	12345	
		confirm	12345	
		nombreReal	jorge alvarez	

Clase: Controlador

Método: **public void nuevoPedido()**

Precondiciones: Hay un usuario de tipo Cliente logeado en la Empresa.

Función: Invoca al método agregarPedido(Pedido pedido) de la clase Empresa.

escenarios	
num escenario	descripción

1	con auto("aaa111",3,true), moto("mmm111",1), con cliente(nombre_usuario="pepe123",nombre_usuario="alfa"), con choferTemporario("777"), con pedido("alfa",1,false,false,5,"ZONA_PELIGROSA")
2	con auto("aaa111",3,true), moto("mmm111",1), con cliente(nombre_usuario="pepe123",nombre_usuario="alfa"), con choferTemporario("777"), con pedido1("alfa",1,false,false,5,"ZONA_PELIGROSA"), con viaje(pedido1,choferTemporario,moto)

batería de pruebas				
num prueba	Escenario	Datos de entrada	Valor	Salida Esperada
1	1	cliente	nombre_usuario="pepe123"	se genera el nuevo pedido y se agrega al hashmap de pedidos
		cantidadPasajeros	3	
		mascota	TRUE	
		baul	TRUE	
		km	5	
2	1	cliente	nombre_usuario="pepe123"	no se genera el nuevo pedido y muestra un cartel que dice que ningun vehiculo satisface el pedido
		cantidadPasajeros	4	
		mascota	TRUE	
		baul	TRUE	
		km	5	
3	1	cliente	nombre_usuario="alfa"	no se genera el nuevo pedido y

		cantidadPasajeros	2	muestra un cartel que dice que el cliente tiene un pedido pendiente
		mascota	FALSE	
		baul	FALSE	
		km	5	
4	1	cliente	nombre_usuario="zzz333"	no se genera el nuevo pedido y muestra un cartel que dice que el cliente no esta registrado
		cantidadPasajeros	2	
		mascota	FALSE	
		baul	FALSE	
		km	2	
5	2	cliente	nombre_usuario="alfa"	no se genera el nuevo pedido y muestra un cartel que dice que el cliente tiene un viaje pendiente
		cantidadPasajeros	2	
		mascota	FALSE	
		baul	FALSE	
		km	2	

Errores encontrados: Cuando crea el pedido correcto, agrega mal la cantidad de kilómetros. No muestra cartel diciendo que el cliente tiene un viaje pendiente ni que tiene viaje pendiente.

Testeo de excepciones

escenarios	
num escenario	descripción

1	con auto("aaa111",3,true), moto("mmm111",1), con cliente1(nombre_usuario="pepe123"),cliente2(nombre_usuario="alfa"), con chofer1("123",temporario), chofer2("555",temporario) con pedido1("alfa",1,false,false,5,"ZONA_STANDARD"), pedido2("pepe123",3,false,false,5,"ZONA_STANDARD"), viaje(pedido1,chofer1,moto)
2	con auto("aaa111",3,true), moto("mmm111",1), con cliente1(nombre_usuario="pepe123"),cliente2(nombre_usuario="alfa"), con choferTemporario("123"), con pedido1("alfa",1,false,false,5,"ZONA_STANDARD")
3	con auto("aaa111",3,true), moto("mmm111",1), con cliente1(nombre_usuario="pepe123"),cliente2(nombre_usuario="alfa"), con chofer1("123",temporario), chofer2("555",temporario) con pedido1("alfa",10,true,true,10,"ZONA_STANDARD")

batería de pruebas			
num prueba	Escenario	Descripcion del caso de prueba	Salida Esperada
1	1	Empresa.getInstance().crearViaje(pedido2, chofer1, auto);	lanza la ChoferNoDisponibleException, la exception contiene la referencia al chofer y el mensaje coincide con Mensajes.CHOFER_NO_DISPONIBLE

2	1	Chofer chofer_repetido = new ChoferTemporario(this.chofer1.getDni(), "carlos"); Empresa.getInstance().agregarChofer(chofer_repetido);	lanza la ChoferRepetidoException, la exception contiene la referencia al chofer existente, contiene el dni pretendido y el mensaje coincide con Mensajes.CHOFER_YA_REGISTRADO
3	1	Pedido pedido2 = new Pedido(this.user1,3,false,false,5,Constantes.ZONA_PELIGROSA); Empresa.getInstance().agregarPedido(pedido2);	lanza la ClienteConPedidoPendienteException y el mensaje coincide con Mensajes.CLIENTE_CON_PEDIDO_PENDIENTE
4	1	Cliente user_no_registrado = new Cliente("no_existo","111","aaa"); Pedido pedido2 = new Pedido(user_no_registrado,3,false,false,5,Constantes.ZONA_PELIGROSA); Empresa.getInstance().agregarPedido(pedido2);	lanza la ClienteNoExisteException y el mensaje coincide con Mensajes.CLIENTE_NO_EXISTE

5	1	Empresa.getInstance().login(this.user1.getNombreUsuario(),this.user1.getPass()); Empresa.getInstance().pagarYFinalizarViaje(3);	lanza la ClienteSinViajePendienteException y el mensaje coincide con Mensajes.CLIENTE_SIN_VIAJE_PENDIENTE
6	1	Pedido pedido_nuevo = new Pedido(this.user2,1,false,false,10,Constantes.ZONA_PELIGROSA); Empresa.getInstance().agregarPedido(pedido_nuevo);	lanza la ClienteConViajePendienteException y el mensaje coincide con Mensajes.CLIENTE_CON_VIAJE_PENDIENTE
7	1	Empresa.getInstance().login(this.user1.getNombreUsuario(),"cualquiera");	lanza la PasswordErroneaException, la exception contiene el nombre de usuario pretendido, contiene la password ingresada y el mensaje coincide con Mensajes.PASS_ERRONEO

8	1	Pedido pedido_no_agregado = new Pedido(this.user1,3,false,false,5,Constantes.ZONA_STANDARD); Empresa.getInstance().crearViaje(pedido_no_agregado, chofer2, auto);	lanza la PedidoInexistenteException, la exception contiene el pedido ingresado y el mensaje coincide con Mensajes.PEDIDO_INEXISTENTE
9	2	Pedido pedido_nuevo = new Pedido(this.user1,10,false,false,5,Constantes.ZONA_STANDARD); Empresa.getInstance().agregarPedido(pedido_nuevo);	lanza la SinVehiculoParaPedidoException, la exception contiene el pedido ingresado y el mensaje coincide con Mensajes.SIN_VEHICULO_PARA_PEDIDO
10	1	Empresa.getInstance().login("no_existe", this.user1.getPass());	lanza la UsuarioNoExisteException, la exception contiene el nombre de usuario ingresado y el mensaje coincide con Mensajes.USUARIO_DESCONOCIDO

11	1	Empresa.getInstance().agregarCliente(this.user1.getNombreUsuario(), "abc", "ariel");	lanza la UsuarioYaExisteException, la exception contiene el nombre de usuario ingresado y el mensaje coincide con Mensajes.USUARIO_REPETIDO
12	1	Empresa.getInstance().crearViaje(this.pedido2, this.chofer2, this.moto);	lanza la VehiculoNoDisponibleException, la exception contiene el vehículo que no está disponible y el mensaje coincide con Mensajes.VEHICULO_NO_DISPONIBLE
13	3	Empresa.getInstance().crearViaje(this.pedido1, chofer1, moto);	lanza la VehiculoNoValidoException, la exception contiene el vehículo que no es válido, contiene el pedido ingresado y el mensaje coincide con Mensajes.VEHICULO_NO_VALIDO

14	1	Combi combi = new Combi(this.auto.getPatente(),10,true); Empresa.getInstance().agregarVehiculo(combi);	lanza la VehiculoRepetidoException, la exception contiene la patente pretendida, contiene el vehículo ya registrado con esa patente y el mensaje coincide con Mensajes.VEHICULO_YA_REGISTRADO

Errores encontrados: En la excepción para el chofer no disponible el mensaje no indica "El chofer no está disponible" si no que dice "El pedido no figura en la lista". Para chofer repetido y para agregar pedido con cliente con viaje pendiente no lanza ninguna excepción. Cuando se quiere agregar un pedido con un cliente con pedido pendiente el mensaje dice "Cliente con viaje pendiente" en lugar de "Cliente con pedido pendiente".

Pruebas de integración

Para las pruebas de integración se decidió implementar mocks para evitar tener que usar el robot a la hora de ingresar los datos en los distintos paneles.

batería de pruebas (admin agrega chofer)				
num prueba	Descripción del caso de prueba	Pasos	Resultado esperado	Módulo involucrados
1	el administrador se logea y agrega un chofer temporario	- se logea como administrador - ingresa los datos del chofer temporario a agregar y lo registra	se agrega al chofer al HashMap de choferes	login y agregar chofer
2	el administrador se logea y agrega un chofer permanente	- se logea como administrador - ingresa los datos del chofer permanente a agregar y lo registra	se agrega al chofer al HashMap de choferes	
3	el administrador se logea y agrega un	- se logea como administrador	no lo agrega la HashMap	

	chofer con dni repetido	- ingresa los datos del chofer temporario a agregar y lo registra	de choferes y muestra un cartel indicando que el chofer ya fue registrado	

Errores encontrados: No muestra ningún cartel indicando que el chofer ya está registrado.

batería de pruebas (admin agrega vehículo)				
num prueba	Descripción del caso de prueba	Pasos	Resultado esperado	Módulo involucrados
1	el administrador se logea y agrega un auto	- se logea como administrador - ingresa los datos del auto a agregar y lo registra	se agrega al auto al HashMap de vehículos	login y agregar vehículo
2	el administrador se logea y agrega una combi con patente repetida	- se logea como administrador - ingresa los datos de la combi a agregar y la registra	no se agrega a la combi al HashMap de vehículos y muestra un cartel indicando que el vehículo ya fue registrado	

batería de pruebas (admin crea viaje)				
num prueba	Descripción del caso de prueba	Pasos	Resultado esperado	Modulo involucrados
1	el administrador se logea y crea un viaje	- se logea como administrador - selecciona el pedido pendiente, el chofer libre y el vehículo disponible - crea el viaje	se crea un viaje y se agrega al HashMap de viajes iniciados	login y crear viaje
2	el administrador se logea y crea un viaje con pedido	- se logea como administrador - selecciona el pedido pendiente, el	no se crea el viaje y muestra un cartel indicando que	

	inexistente	chofer libre y el vehículo disponible - se crea otro viaje con ese pedido seleccionado - crea el viaje	el pedido seleccionado es inexistente	
3	el administrador se logea y crea un viaje con chofer no disponible	- se logea como administrador - selecciona el pedido pendiente, el chofer libre y el vehículo disponible - se crea otro viaje con ese chofer seleccionado - crea el viaje	no se crea el viaje y muestra un cartel indicando que el chofer no está disponible	
4	el administrador se logea y crea un viaje con vehículo no disponible	- se logea como administrador - selecciona el pedido pendiente, el chofer libre y el vehículo disponible - se crea otro viaje con ese vehículo seleccionado - crea el viaje	no se crea el viaje y muestra un cartel indicando que el vehiculo no esta disponible	

Usando el panel de administrador no se puede dar el caso de que se seleccione un vehículo que no pueda satisfacer el pedido.

batería de pruebas (registro y login de cliente)				
num prueba	Descripción del caso de prueba	Pasos	Resultado esperado	Módulo involucrados
1	el cliente se registra y logea	- el cliente ingresa sus datos para registrarse - ingresa sus credenciales para logearse	al registrarse se agrega al HashMap de clientes y al hacer login se setea como el usuario logeado	registro y login
2	el cliente se registra con username "admin" y se logea	- el cliente ingresa sus datos para registrarse con username="admin" - ingresa sus credenciales para logearse	al registrarse se agrega al HashMap de clientes y se setea como el usuario logeado pero no como admin	
3	se registra pero ingresa mal su	- el cliente ingresa sus datos	al registrarse se agrega al	

	username	para registrarse - ingresa mal su username para logearse	HashMap de clientes y al hacer login muestra un cartel indicando que el usuario no existe	
4	se registra pero ingresa mal su password	- el cliente ingresa sus datos para registrarse - ingresa mal su password para logearse	al registrarse se agrega al HashMap de clientes y al hacer login muestra un cartel indicando que la password es errónea	

El testeo de usuario repetido está realizado en las pruebas unitarias.

batería de pruebas (login y crea pedido)				
num prueba	Descripción del caso de prueba	Pasos	Resultado esperado	Módulo involucrados
1	el cliente se logea y realiza un pedido	- se logea como cliente - ingresa los datos del pedido	se agrega el pedido al HashMap de pedidos	login y crea pedido
2	el cliente se logea y realiza un pedido pero no hay vehículo que lo satisfaga	- se logea como cliente - ingresa los datos del pedido	no se agrega el pedido y se muestra un cartel indicando que no hay vehículo para el pedido	

Usando el panel de cliente no se puede dar el caso de que tenga un pedido o viaje iniciado, o que el cliente no esté registrado.

batería de pruebas (login, crea pedido y termina viaje)				
num prueba	Descripción del caso de prueba	Pasos	Resultado esperado	Módulo involucrados
1	el cliente se logea, realiza un pedido y luego lo finaliza	- se logea como cliente - ingresa los datos del pedido - el admin asigna chofer y vehículo para crear el viaje - califica y paga el viaje	se agrega el pedido al HashMap de pedidos, se agrega el viaje al HashMap de viajes iniciados, se asigna calificación al chofer y se agrega el viajes al ArrayList	login, crea pedido y termina viaje

			de viajes terminados	

Usando el panel de administrador no se puede dar el caso de que el cliente esté realizando ya un viaje.

Test de GUI

batería de pruebas (panel registro)			
num prueba	panel	Descripción del caso de prueba	Resultado esperado
1	registro	el usuario ingresa todos los datos necesarios para registrarse	pasa al panel de login
2		el usuario quiere registrar un usuario repetido	muestra cartel indicando que el usuario está repetido
3		el cliente ingresa todos los datos pero la password y confirmación no coinciden	muestra un cartel indicando que la password y confirmación no coinciden
4		el cliente no ingresa ningún dato	el botón de registrar permanece deshabilitado
5		el cliente ingresa nombre pero ningún otro dato	el botón de registrar permanece deshabilitado
6		el cliente ingresa password pero ningún otro dato	el botón de registrar permanece deshabilitado
7		el cliente ingresa confirmación pero ningún otro	el botón de registrar permanece

		dato	deshabilitado
8		el cliente ingresa nombre real pero ningún otro dato	el botón de registrar permanece deshabilitado
9		el cliente ingresa todos los datos menos la confirmación	el botón de registrar permanece deshabilitado

batería de pruebas (panel login)			
num prueba	panel	Descripción del caso de prueba	Resultado esperado
1	login	el cliente ingresa todos los datos necesarios para logearse	pasa al panel de cliente
2		el admin ingresa todos los datos necesarios para logearse	pasa al panel de administrador
3		el administrador ingresa la password incorrecta	muestra cartel indicando que la password es incorrecta
4		el cliente ingresa la password incorrecta	muestra cartel indicando que la password es incorrecta
5		el usuario ingresa un nombre de usuario no registrado	muestra cartel indicando que el usuario es desconocido
6		el usuario no ingresa ningún dato	el botón de login permanece deshabilitado

7		el usuario ingresa nombre de usuario pero no password	el botón de login permanece deshabilitado
8		el usuario ingresa password pero no nombre de usuario	el botón de login permanece deshabilitado

Errores encontrados: Cuando se ingresa mal la password del administrador no muestra ningún cartel indicando que la password es incorrecta ya que confunde al administrador con un cliente con nombre de usuario = “admin”.

batería de pruebas (panel cliente)				
num prueba	panel	Descripción del caso de prueba	escenario	Resultado esperado
1	panel cliente	el usuario se logea y accede al panel de cliente	sin pedidos ni viajes iniciados	Los TextField, JRadioButton y JCheckBox del sub panel Nuevo Pedido estarán habilitados. El TextField CALIFICACION estará deshabilitado, y el TextField VALOR_VIAJE estará vacío.
2		el usuario se logea, accede al panel de cliente y CANT_PAX = 1	sin pedidos ni viajes iniciados	el boton NUEVO_PEDIDO debería estar habilitado
3		el usuario se logea, accede al panel de cliente y CANT_PAX = 0	sin pedidos ni viajes iniciados	el boton NUEVO_PEDIDO debería estar deshabilitado
4		el usuario se logea, accede al panel de cliente y	sin pedidos ni viajes iniciados	el boton NUEVO_PEDIDO debería estar habilitado

		CANT_PAX = 10		
5		el usuario se logea, accede al panel de cliente y CANT_PAX = 11	sin pedidos ni viajes iniciados	el boton NUEVO_PEDIDO debería estar deshabilitado
6		el usuario se logea, accede al panel de cliente y CANT_KM = 0	sin pedidos ni viajes iniciados	el boton NUEVO_PEDIDO debería estar habilitado
7		el usuario se logea, accede al panel de cliente y CANT_KM = -1	sin pedidos ni viajes iniciados	el boton NUEVO_PEDIDO debería estar deshabilitado
8		el usuario se logea, accede al panel de cliente y CANT_PAX=1 y CANT_KM = 5, y apreta en boton NUEVO_PEDIDO	sin pedidos ni viajes iniciados, y con vehículo que satisface	se borran los contenidos de los JTextField CANT_PAX y CANT_KM, y se agrega al JTextArea de viajes realizados
9		el usuario se logea, accede al panel de cliente y CANT_PAX=1 y CANT_KM = 5, y apreta en boton NUEVO_PEDIDO	sin pedidos ni viajes iniciados, y sin vehículo que satisface	se lanza un cartel indicando que no hay vehículo para el pedido
10		el usuario se logea y accede al panel de cliente	con pedido iniciado	Los TextField, JRadioButton y JCheckBox del sub panel Nuevo Pedido estarán deshabilitados. El TextField CALIFICACION estará deshabilitado, y el TextField VALOR_VIAJE estará vacío.
11		el usuario se logea y accede al panel de cliente	está en viaje	EL TextField CALIFICACION estará habilitado, y el TextField VALOR_VIAJE contendrá el importe a pagar. Los TextField, JRadioButton y JCheckBox

				del sub panel Nuevo Pedido estarán deshabilitados.
12		el usuario se logea, accede al panel de cliente e ingresa una calificación de 0	está en viaje	el botón CALIFICAR_PAGAR deberia estar habilitado
13		el usuario se logea, accede al panel de cliente e ingresa una calificación de 5	está en viaje	el botón CALIFICAR_PAGAR deberia estar habilitado
14		el usuario se logea, accede al panel de cliente e ingresa una calificación de 3 y apreta el boton de CALIFICAR_PAGAR	está en viaje	Se elimina el viaje del JTextArea PEDIDO_O_VIAJE_ACTUAL, y se agrega a la JList LISTA_VIAJES_CLIENTE. Se borra el contenido del JTextField CALIFICACION.

Errores encontrados: No permite ingresar una calificación 0 al chofer ya que no se encuentra habilitado el botón de calificar y pagar para terminar el viaje.

batería de pruebas (panel administrador)				
num prueba	panel	Descripción del caso de prueba	escenario	Resultado esperado
1	panel administrador	quiere registrar chofer temporario pero no ingresa dni	se registra como admin	el boton NUEVO_CHOFER debería estar deshabilitado
2		quiere registrar chofer temporario pero no ingresa nombre	se registra como admin	el boton NUEVO_CHOFER debería estar deshabilitado

3		quiere registrar chofer temporario ingresa todos los datos y apreta boton NUEVO_CHOFER	se registra como admin	se agrega a la JList LISTA_CHOFERES_TOTALES. Luego se vacían los JTextField de registro de choferes.
4		quiere registrar chofer permanente con CH_CANT_HIJOS = 0	se registra como admin	el boton NUEVO_CHOFER debería estar habilitado
5		quiere registrar chofer permanente con CH_CANT_HIJOS = -1	se registra como admin	el boton NUEVO_CHOFER debería estar deshabilitado
6		quiere registrar chofer permanente con CH_ANIO = 1900	se registra como admin	el boton NUEVO_CHOFER debería estar habilitado
7		quiere registrar chofer permanente con CH_ANIO = 1899	se registra como admin	el boton NUEVO_CHOFER debería estar deshabilitado
8		quiere registrar chofer permanente con CH_ANIO = 3000	se registra como admin	el boton NUEVO_CHOFER debería estar habilitado
9		quiere registrar chofer permanente con CH_ANIO = 3001	se registra como admin	el boton NUEVO_CHOFER debería estar deshabilitado
10		quiere registrar chofer permanente ingresa todos los datos y apreta boton NUEVO_CHOFER	se registra como admin	se agrega a la JList LISTA_CHOFERES_TOTALES. Luego se vacían los JTextField de registro de choferes.

11		registra un chofer con el mismo dni del ya registrado	se registra como admin y hay un chofer ya registrado con el mismo dni	se lanza un cartel indicando que el chofer ya está registrado
12		quiere registrar moto con patente vacía	se registra como admin	el boton NUEVO_VEHICULO debería estar deshabilitado
13		quiere registrar auto con CANTIDAD_PLAZAS = 0	se registra como admin	el boton NUEVO_VEHICULO debería estar deshabilitado
14		quiere registrar auto con CANTIDAD_PLAZAS = 1	se registra como admin	el boton NUEVO_VEHICULO debería estar habilitado
15		quiere registrar auto con CANTIDAD_PLAZAS = 4	se registra como admin	el boton NUEVO_VEHICULO debería estar habilitado
16		quiere registrar auto con CANTIDAD_PLAZAS = 5	se registra como admin	el boton NUEVO_VEHICULO debería estar deshabilitado
17		quiere registrar combi con CANTIDAD_PLAZAS = 4	se registra como admin	el boton NUEVO_VEHICULO debería estar deshabilitado
18		quiere registrar combi con CANTIDAD_PLAZAS = 5	se registra como admin	el boton NUEVO_VEHICULO debería estar habilitado
19				

		quiere registrar combi con CANTIDAD_PLAZAS = 10	se registra como admin	el boton NUEVO_VEHICULO debería estar habilitado
20		quiere registrar combi con CANTIDAD_PLAZAS = 11	se registra como admin	el boton NUEVO_VEHICULO debería estar deshabilitado
21		se ingresan los datos de la moto y apreta boton NUEVO_VEHICULO	se registra como admin	se agrega al JList LISTA_VEHICULOS_TOTALES. Luego se vacían los JTextField de registro de vehículo.
22		se ingresan los datos del auto y apreta boton NUEVO_VEHICULO	se registra como admin	se agrega al JList LISTA_VEHICULOS_TOTALES. Luego se vacían los JTextField de registro de vehículo.
23		se ingresan los datos de la combi y apreta boton NUEVO_VEHICULO	se registra como admin	se agrega al JList LISTA_VEHICULOS_TOTALES. Luego se vacían los JTextField de registro de vehículo.
24		quiere registrar moto con patente ya registrada por la moto	se registra como admin y hay moto registrada	se lanza un cartel indicando que el vehículo ya fue registrado
25		quiere registrar auto con patente ya registrada por la moto		se lanza un cartel indicando que el vehículo ya fue registrado
26		quiere registrar combi con patente ya registrada por la moto		se lanza un cartel indicando que el vehículo ya fue registrado
27				

		sin chofer seleccionado de la lista LISTA_CHOFERES_TOTALES	con choferes registrados	La JList LISTA_VIAJES_DE_CHOFER Estará vacía, el TextField CALIFICACION_CHOFER estará vacío y El TextField SUELDO_DE_CHOFER estará vacío.
28		con chofer seleccionado de la lista LISTA_CHOFERES_TOTALES sin viajes realizados	con choferes registrados	la JList LISTA_VIAJES_DE_CHOFER debera estar vacía , El TextField muestra CALIFICACION_CHOFER muestra 0 y El TextField SUELDO_DE_CHOFER muestra el sueldo
29		con chofer seleccionado de la lista LISTA_CHOFERES_TOTALES con viajes realizados	con choferes registrados	Se visualizará en la JList LISTA_VIAJES_DE_CHOFER los viajes históricos de dicho chofer, El TextField muestra CALIFICACION_CHOFER muestra la calificación y El TextField SUELDO_DE_CHOFER muestra el sueldo
30		ingresa al panel de administrador	con clientes registrados	JList LISTADO_DE_CLIENTES con los clientes registrados
31		ingresa al panel de administrador	sin clientes registrados	JList LISTADO_DE_CLIENTES vacío
32		ingresa al panel de administrador	con vehículos registrados	JList LISTA_VEHICULOS_TOTALES con los vehiculos registrados
33		ingresa al panel de administrador	sin vehículos registrados	JList LISTA_VEHICULOS_TOTALES vacio

34		ingresa al panel de administrador	con viajes realizados	JList LISTA_VIAJES_HISTORICOS con los viajes realizados
35		ingresa al panel de administrador	sin viajes realizados	JList LISTA_VIAJES_HISTORICOS vacio
36		ingresa al panel de administrador	con choferes registrados	TextField TOTAL_SUELDOS_A_PAGAR mayor a 0
37		ingresa al panel de administrador	sin choferes registrados	TextField TOTAL_SUELDOS_A_PAGAR debe dar 0
38		ingresa al panel de administrador	con pedidos pendientes	JList LISTA_PEDIDOS_PENDIENTES no vacía
39		ingresa al panel de administrador	sin pedidos pendientes	JList LISTA_PEDIDOS_PENDIENTES vacía
40		ingresa al panel de administrador	con choferes libres	JList LISTA_CHOFERES_LIBRES no vacía
41		ingresa al panel de administrador	sin choferes libres	JList LISTA_CHOFERES_LIBRES vacia
42				LISTA_VEHICULOS_DISPONIBLES no vacía

		ingresa al panel de administrador y se selecciona un pedido de LISTA_PEDIDOS_PENDIENTES	con vehículos que satisfacen	
43		ingresa al panel de administrador y se selecciona un pedido de LISTA_PEDIDOS_PENDIENTES	ningún vehículo satisface el pedido	LISTA_VEHICULOS_DISPONIBLES vacia
45		ingresa al panel de administrador y se selecciona un pedido de LISTA_PEDIDOS_PENDIENTES, un chofer de LISTA_CHOFERES_LIBRE y vehículo de LISTA_VEHICULOS_DISPONIBLES	con pedido pendiente, choferes libres y vehículo que satisface	JButton NUEVO_VIAJE habilitado
46		ingresa al panel de administrador y se selecciona un pedido de LISTA_PEDIDOS_PENDIENTES, un vehiculo de LISTA_VEHICULOS_DISPONIBLES pero no chofer de LISTA_CHOFERES_LIBRE	con pedido pendiente, choferes libres y vehículo que satisface	JButton NUEVO_VIAJE deshabilitado
48		ingresa al panel de administrador y se selecciona un pedido de	con pedido pendiente, choferes libres y vehículo que satisface	JButton NUEVO_VIAJE deshabilitado

		LISTA_PEDIDOS_PENDIENTES, un chofer de LISTA_CHOFERES_LIBRE pero no vehiculo de LISTA_VEHICULOS_DISPONIBLES		
49		ingresa al panel de administrador y se selecciona un pedido de LISTA_PEDIDOS_PENDIENTES, un chofer de LISTA_CHOFERES_LIBRE y vehículo de LISTA_VEHICULOS_DISPONIBLES y apreta boton NUEVO_VIAJE	con pedido pendiente, choferes libres y vehículo que satisface	Se actualizan las JList, LISTA_PEDIDOS_PENDIENTES y LISTA_CHOFERES_LIBRES, y se vacía la JList LISTA_VEHICULOS_DISPONIBLES. En consecuencia, se deshabilita el JButton NUEVO_VIAJE

Errores encontrados: Cuando se agrega al chofer, tanto temporario como permanente, correctamente no vacía los JTextField de registro de choferes. No lanza ningún cartel indicando que el chofer ya está registrado. El botón para agregar un auto con cantidad de plazas = 0 se encuentra habilitado cuando no tendría. Cuando se agrega un vehículo correctamente no se vacían los JTextField de registro de vehículo. Además, los casos de prueba que necesitan de seleccionar un pedido en la lista de pedidos pendientes no es posible realizarlos ya que en lugar de seleccionar una opción de allí, se seleccionan en la lista de choferes totales.

Test Caja Blanca

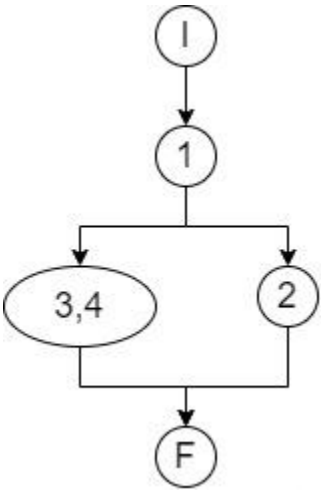
Clase: Empresa

Método: agregarChofer

Test de cobertura

```
public void agregarChofer(Chofer chofer) throws ChoferRepetidoException {
  if (this.choferes.containsValue(chofer)) { #1
    throw new ChoferRepetidoException(chofer.getDni(), (Chofer)this.choferes.get(chofer.getDni())); #2
  } else {
    this.choferes.put(chofer.getDni(), chofer); #3
    this.choferesDesocupados.add(chofer); #4
  }
}
```

Grafo de Flujo



Caminos Básicos

C1 = 1 – 2 – F

C2 = 1 – 3 – 4 – F

C1 cubre nodo #2 que esta sin pisar.

ID	OBJETIVO DE LA PRUEBA	DATOS DE ENTRADA	PROCEDIMIENTO	SALIDA ESPERADA	RESULTADO
T1	Verificar camino 1	Valor de las variables dni y chofer	-Modificar el valor de dni = 1 -Crear chofer usando la variable dni -Agregar chofer a empresa -Ejecutar la prueba	ChoferRepetidoException	pass

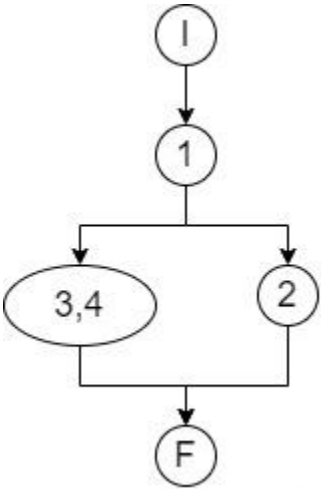
Clase: Empresa

Método: agregarCliente

Test de cobertura

```
public void agregarCliente(String usuario, String pass, String nombreReal) throws
UsuarioYaExisteException {
   if (this.clientes.containsKey(usuario)) { #1
      throw new UsuarioYaExisteException(usuario); #2
   } else {
      Cliente cliente = new Cliente(usuario, pass, nombreReal); #3
      this.clientes.put(cliente.getNombreUsuario(), cliente); #4
   }
}
```

Grafo de Flujo



Caminos Básicos

C1 = 1 – 2 – F

C2 = 1 – 3 – 4 – F

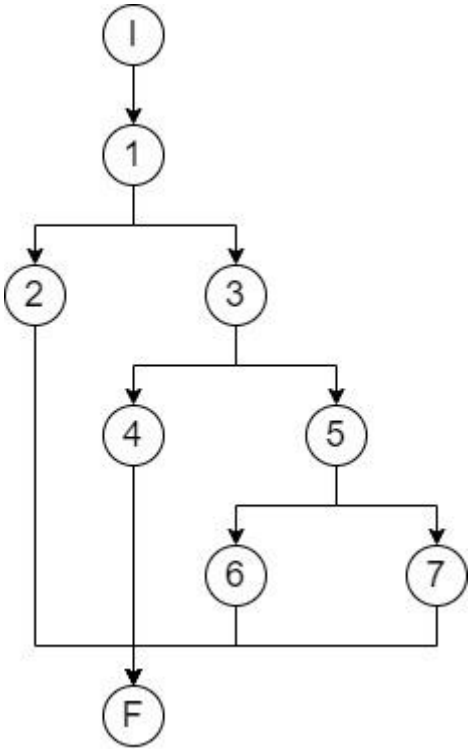
Clase: Empresa

Método: agregarPedido

Test de cobertura

```
public void agregarPedido(Pedido pedido) throws SinVehiculoParaPedidoException,
ClienteNoExisteException, ClienteConViajePendienteException, ClienteConPedidoPendienteException {
   if (!this.clientes.containsValue(pedido.getCliente())) { #1
      throw new ClienteNoExisteException(); #2
   } else if (this.pedidos.containsKey(pedido.getCliente())) { #3
      throw new ClienteConPedidoPendienteException(); #4
   } else if (this.validarPedido(pedido)) { #5
      this.pedidos.put(pedido.getCliente(), pedido); #6
   } else {
      throw new SinVehiculoParaPedidoException(pedido); #7
   }
}
```

Grafo de Flujo



Caminos Básicos

C1 = 1 – 2 – F

C2 = 1 – 3 – 4 – F

C3 = 1 – 3 – 5 – 6 – F

C4 = 1 – 3 – 5 – 7 – F

Clase: Empresa

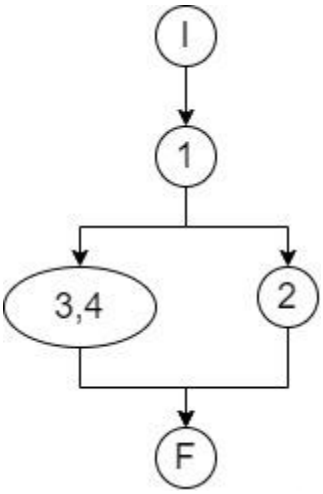
Método: agregarVehiculo

Test de cobertura

```
public void agregarVehiculo(Vehiculo vehiculo) throws VehiculoRepetidoException {
  if (this.vehiculos.containsKey(vehiculo.getPatente())) { #1
    throw new VehiculoRepetidoException(vehiculo.getPatente(),  (Vehiculo)null); #2
  } else {
    this.vehiculos.put(vehiculo.getPatente(), vehiculo); #3
```

```
    this.vehiculosDesocupados.add(vehiculo); #4
  }
}
```

Grafo de Flujo



Caminos Básicos

C1 = 1 – 2 – F

C2 = 1 – 3 – 4 – F

Clase: Empresa

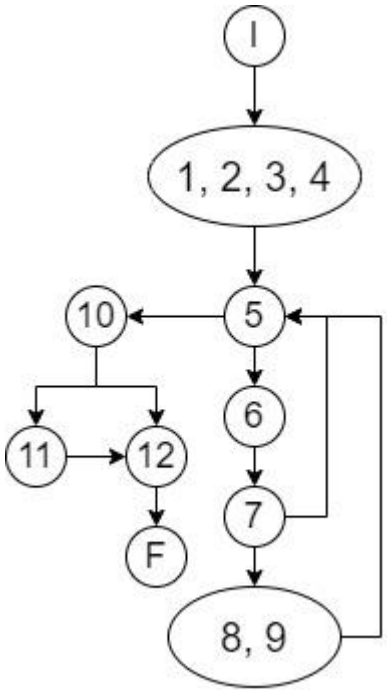
Método: calificacionDeChofer

Test de cobertura

```
public double calificacionDeChofer(Chofer chofer) throws SinViajesException {
  double r = 0.0; #1
  double suma = 0.0; #2
  int cant = 0; #3
  Iterator<Viaje> it = this.viajesTerminados.iterator(); #4

  while(it.hasNext()) { #5
    Viaje v = (Viaje)it.next(); #6
```

```
      if (v.getChofer() == chofer) { #7
        suma = (double)v.getCalificacion(); #8
        ++cant; #9
      }
    }

    if (cant > 0) { #10
      r = suma / (double)cant; #11
    }

    return r; #12
}
```

Grafo de Flujo



Caminos Básicos

C1 = 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 5 – 10 – 11 – 12 – F

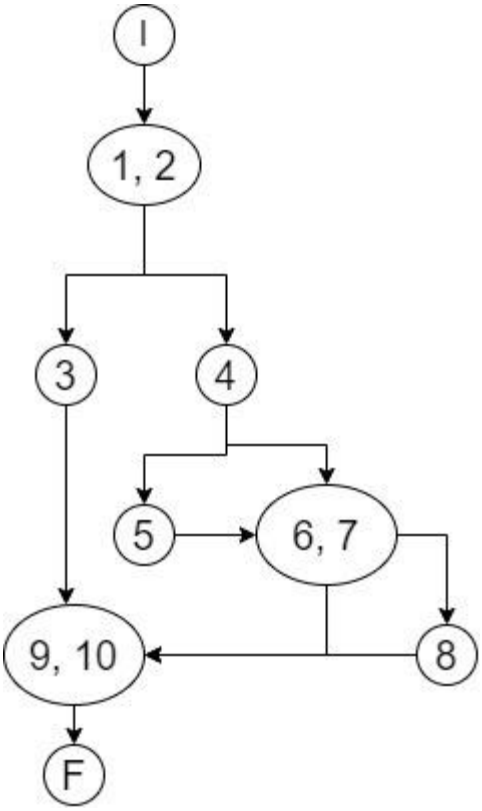
Clase: Empresa

Método: login

Test de cobertura

```
public Usuario login(String usserName, String pass) throws UsuarioNoExisteException,
PasswordErroneaException {
   Usuario u = null; #1
   if (usserName.equals("admin") && pass.equals("admin")) { #2
      u = Administrador.getInstance(); #3
   } else {
      if (!this.clientes.containsKey(usserName)) { #4
         throw new UsuarioNoExisteException(usserName);  #5
      }

      u = (Usuario)this.clientes.get(usserName); #6
      if (!((Usuario)u).getPass().equals(pass)) { #7
         throw new PasswordErroneaException(usserName, pass); #8
      }
   }

   this.usuarioLogeado = (Usuario)u; #9
   return (Usuario)u; #10
}
```

Grafo de Flujo



Caminos Básicos

C1 = 1 – 2 – 3 – 9 – 10 – F

C2 = 1 – 2 – 4 – 5 – 6 – 7 – 8 – 9 – 10 – F

Clase: Empresa

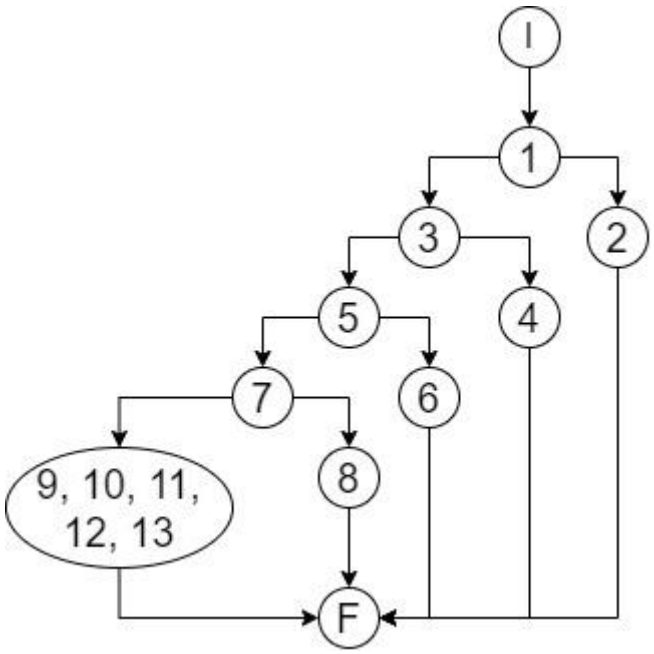
Método: crearViaje

Test de cobertura

```
public void crearViaje(Pedido pedido, Chofer chofer, Vehiculo vehiculo) throws
PedidoInexistenteException, choferNoDisponibleException, VehiculoNoDisponibleException,
VehiculoNoValidoException, ClienteConViajePendienteException {
  if (!this.pedidos.containsValue(pedido)) { #1
    throw new PedidoInexistenteException(pedido); #2
```

```
    } else if (!this.choferesDesocupados.contains(chofer)) { #3
        throw new ChoferNoDisponibleException(chofer); #4
    } else if (!this.vehiculosDesocupados.contains(vehiculo)) { #5
        throw new VehiculoNoDisponibleException(vehiculo); #6
    } else if (vehiculo.getPuntajePedido(pedido) == null) { #7
        throw new VehiculoNoValidoException(vehiculo, pedido); #8
    } else {
        Viaje viaje = new Viaje(pedido, chofer, vehiculo); #9
        this.pedidos.remove(pedido.getCliente()); #10
        this.viajesIniciados.put(viaje.getPedido().getCliente(), viaje); #11
        this.choferesDesocupados.remove(chofer); #12
        this.vehiculosDesocupados.remove(vehiculo); #13
    }
}
```

Grafo de Flujo



Caminos Básicos

C1 = 1 – 2 – F

C2 = 1 – 3 – 4 – F

C3 = 1 – 3 – 5 – 6 – F

C4 = 1 – 3 – 5 – 7 – 8 – F

C5 = 1 – 3 – 5 – 7 – 9 – 10 – 11 – 12 – 13 - F

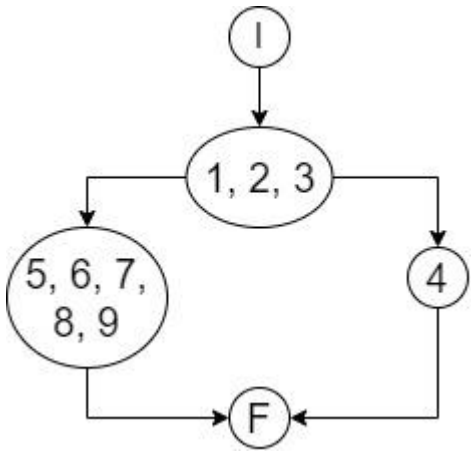
Clase: Empresa

Método: pagarYFinalizarViaje

Test de cobertura

```
public void pagarYFinalizarViaje(int calificacion) throws ClienteSinViajePendienteException {
   Cliente cli = (Cliente)this.usuarioLogeado; #1
   Viaje viaje = this.getViajeDeCliente(cli); #2
   if (viaje == null) { #3
      throw new ClienteSinViajePendienteException(); #4
   } else {
      this.viajesTerminados.add(viaje); #5
    #6 this.viajesIniciados.remove(viaje.getPedido().getCliente());
      viaje.finalizarViaje(calificacion); #7
      this.choferesDesocupados.add(viaje.getChofer()); #8
      this.vehiculosDesocupados.add(viaje.getVehiculo()); #9
   }
}
```

Grafo de Flujo



Caminos Básicos

C1 = 1 – 2 – 3 – 4 – F

C2 = 1 – 2 – 3 – 5 – 6 – 7 – 8 – 9 – F

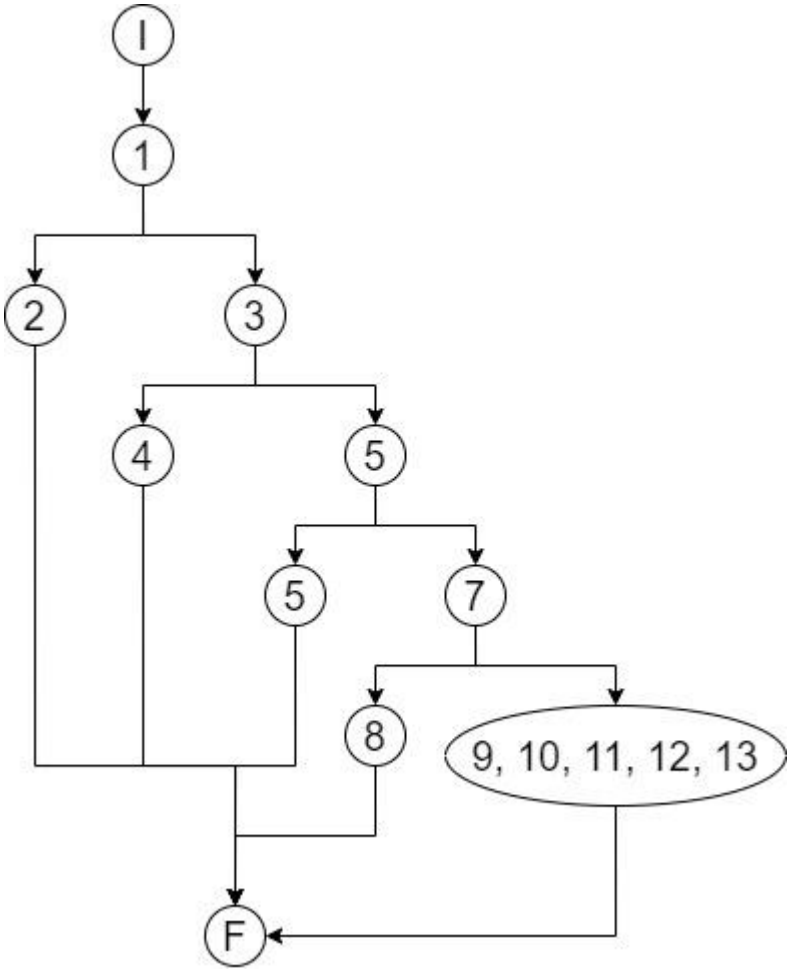
Clase: Empresa

Método: crearViaje

Test de cobertura

```
public void crearViaje(Pedido pedido, Chofer chofer, Vehiculo vehiculo) throws
PedidoInexistenteException, ChoferNoDisponibleException, VehiculoNoDisponibleException,
VehiculoNoValidoException, ClienteConViajePendienteException {
   if (!this.pedidos.containsValue(pedido)) { #1
      throw new PedidoInexistenteException(pedido); #2
   } else if (!this.choferesDesocupados.contains(chofer)) { #3
      throw new ChoferNoDisponibleException(chofer); #4
   } else if (!this.vehiculosDesocupados.contains(vehiculo)) { #5
      throw new VehiculoNoDisponibleException(vehiculo); #6
   } else if (vehiculo.getPuntajePedido(pedido) == null) { #7
      throw new VehiculoNoValidoException(vehiculo, pedido); #8
   } else {
      Viaje viaje = new Viaje(pedido, chofer, vehiculo); #9
      this.pedidos.remove(pedido.getCliente()); #10
      this.viajesIniciados.put(viaje.getPedido().getCliente(), viaje); #11
      this.choferesDesocupados.remove(chofer); #12
      this.vehiculosDesocupados.remove(vehiculo); #13
   }
}
```

Grafo de Flujo



Caminos Básicos

C1 = 1 – 2 – F

C2 = 1 – 3 – 4 – F

C3 = 1 – 3 – 5 – 6 – F

C4 = 1 – 3 – 5 – 7 – 8 – F

C5 = 1 – 3 – 5 – 7 – 9 – 10 – 11 – 12 – 13 – F

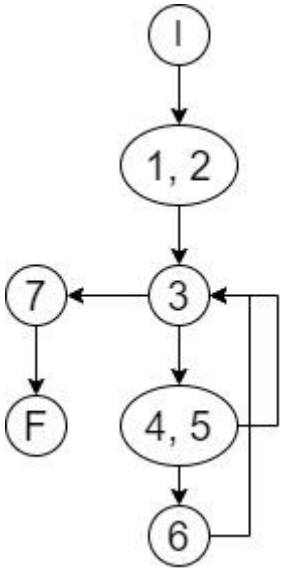
Clase: Empresa

Método: validarPedido

Test de cobertura

```
public boolean validarPedido(Pedido pedido) {
   Iterator<Vehiculo> it = this.iteratorVehiculos(); #1
   Vehiculo vehiculoValido = null; #2

   while(it.hasNext() && vehiculoValido == null) { #3
      Vehiculo actual = (Vehiculo)it.next();#4
      if (actual.getPuntajePedido(pedido) != null) { #5
         vehiculoValido = actual; #6
      }
   }

   return vehiculoValido != null; #7
}
```

Grafo de Flujo



Caminos Básicos

C1 = 1 – 2 – 3 – 4 – 5 – 6 – 3 – 7 - F

Clase: Empresa

Método: logout

Test de cobertura

```
public void logout() {
   this.usuarioLogeado = null; #1
}
```

Grafo de Flujo

Caminos Básicos

C1 = 1 – F

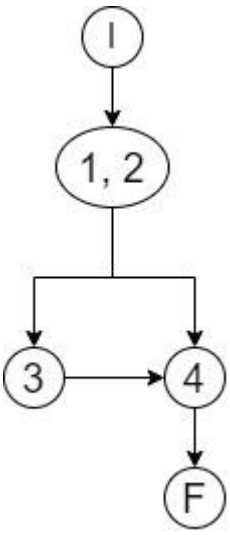
Clase: Moto

Método: getPuntajePedido

Test de cobertura

```
public Integer getPuntajePedido(Pedido pedido) {
   Integer respuesta = null; #1
   if (!pedido.isMascota() && !pedido.isBaul() && pedido.getCantidadPasajeros() == 1) { #2
      respuesta = 1000; #3
   }

   return respuesta; #4
}
```

Grafo de Flujo



Caminos Básicos

C1 = 1 – 2 – 4 – F

C2 = 1 – 2 – 3 – 4 - F

Clase: Auto

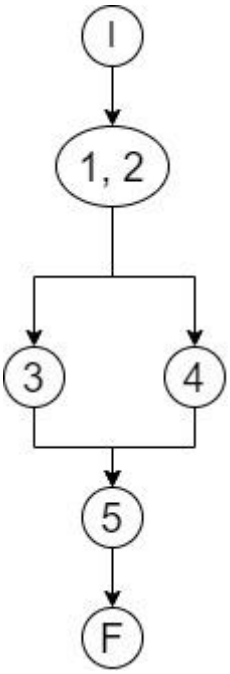
Método: getPuntajePedido

Test de cobertura

```
public Integer getPuntajePedido(Pedido pedido) {
   Integer respuesta; #1
   if ((!pedido.isMascota() || this.isMascota()) && pedido.getCantidadPasajeros() <=
this.getCantidadPlazas()) { #2
      respuesta = pedido.getCantidadPasajeros() * 40; #3
   } else {
      respuesta = null; #4
   }
```

```
    return respuesta; #5
}
```

Grafo de Flujo



Caminos Básicos

C1 = 1 – 2 – 3 – 5 – F

C2 = 1 – 2 – 4 – 5 – F

Clase: Combi

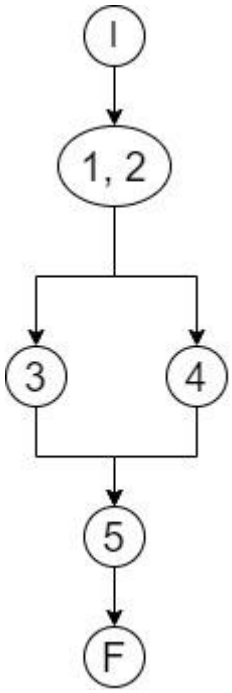
Método: getPuntajePedido

Test de cobertura

```
public Integer getPuntajePedido(Pedido pedido) {
    Integer respuesta; #1
    if (pedido.isMascota() && !this.isMascota()) { #2
```

```
    respuesta = null; #3
  } else {
    respuesta = pedido.getCantidadPasajeros() * 10; #4
  }

  return respuesta; #5
}
```

Grafo de Flujo



Caminos Básicos

C1 = 1 – 2 – 3 – 5 – F

C1 = 1 – 2 – 4 – 5 - F

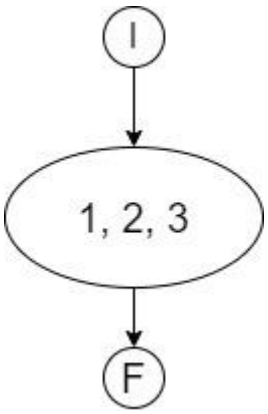
Clase: ChoferPermanente

Método: getSueldoBruto

Test de cobertura

```
public double getSueldoBruto() {
   double ant = 0.05 * (double)this.getAntiguedad(); #1
   double h = 0.07 * (double)this.cantidadHijos; #2
   return Chofer.getSueldoBasico() * (1.0 + ant + h); #3
}
```

Grafo de Flujo



Caminos Básicos

C1 = 1 – 2 – 3 – F

Clase: Viaje

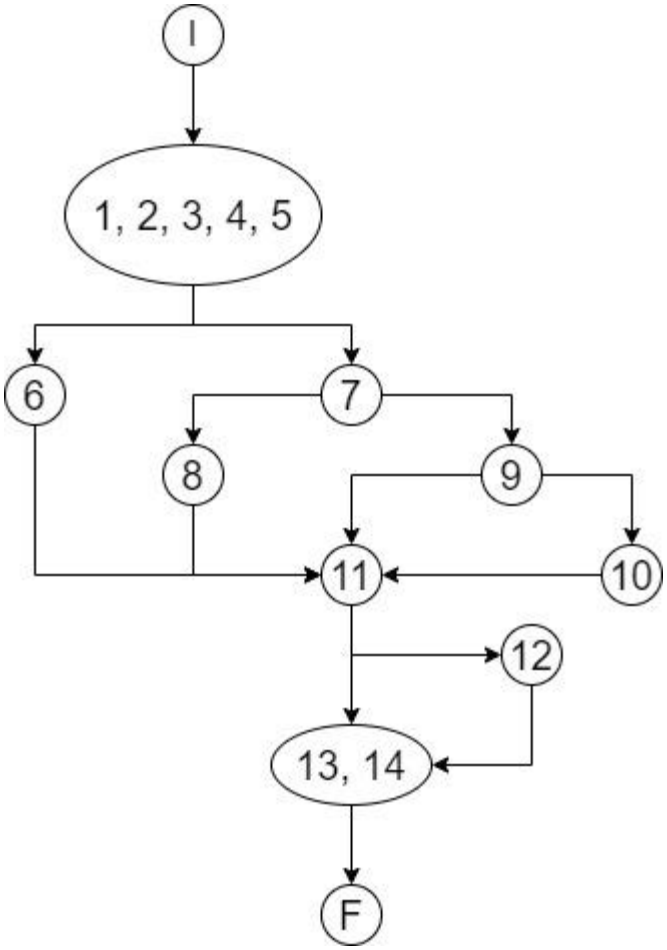
Método: getValor

Test de cobertura

```
public double getValor() {
   double resultado = 0.0; #1
   double incrementoZona = 0.0; #2
   double incrementoBaul = 0.0; #3
   double incrementoMascota = 0.0; #4
   if (this.pedido.getZona().equals("ZONA_PELIGROSA")) { #5
      incrementoZona = 0.1 * (double)this.pedido.getCantidadPasajeros() + 0.2 *
(double)this.pedido.getKm(); #6
```

```
    } else if (this.pedido.getZona().equals("ZONA_SIN_ASFALTAR")){#7
        incrementoZona = 0.2 * (double)this.pedido.getCantidadPasajeros() + 0.15 *
(double)this.pedido.getKm(); #8
    } else if (this.pedido.getZona().equals("ZONA_SIN_ASFALTAR")){#9
        incrementoZona = 0.2 * (double)this.pedido.getCantidadPasajeros() + 0.15 *
(double)this.pedido.getKm(); #10
    }

    if (this.pedido.isMascota()) { #11
        incrementoMascota = 0.1 * (double)this.pedido.getCantidadPasajeros() + 0.2 *
(double)this.pedido.getKm(); #12
    }

    resultado = (1.0 + incrementoZona + incrementoMascota + incrementoBaul) * valorBase; #13
    return resultado; #14
}
```

Grafo de Flujo



Caminos Básicos

C1 = 1 – 2 – 3 – 4 – 5 – 6 – 11 – 12 – 13 – 14 – F

C2 = 1 – 2 – 3 – 4 – 5 – 7 – 8 – 11 – 13 – 14 – F

C3 = 1 – 2 – 3 – 4 – 5 – 7 – 9 – 10 – 11 – 13 – 14 – F

Es imposible acceder al nodo #10 por lo que se lo excluye de la cobertura.

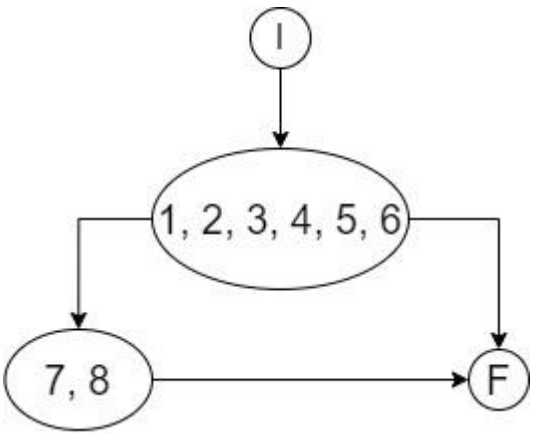
Clase: Controlador

Método: nuevoViaje

Test de cobertura

```
public void nuevoViaje() {
   Pedido pedido = this.vista.getPedidoSeleccionado(); #1
   Chofer chofer = this.vista.getChoferDisponibleSeleccionado(); #2
   Vehiculo vehiculo = this.vista.getVehiculoDisponibleSeleccionado(); #3

   try { #4
    #5 Empresa.getInstance().crearViaje(pedido, chofer, vehiculo);
      this.vista.actualizar(); #6
   } catch (ChoferNoDisponibleException | VehiculoNoDisponibleException | VehiculoNoValidoException |
ClienteConViajePendienteException | PedidoInexistenteException var5) {
      Exception e = var5; #7
      #8     this.vista.getOptionPane().ShowMessage(((Exception)e).getMessage());
   }

}
```

Grafo de Flujo



Caminos Básicos

C1 = 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 - F

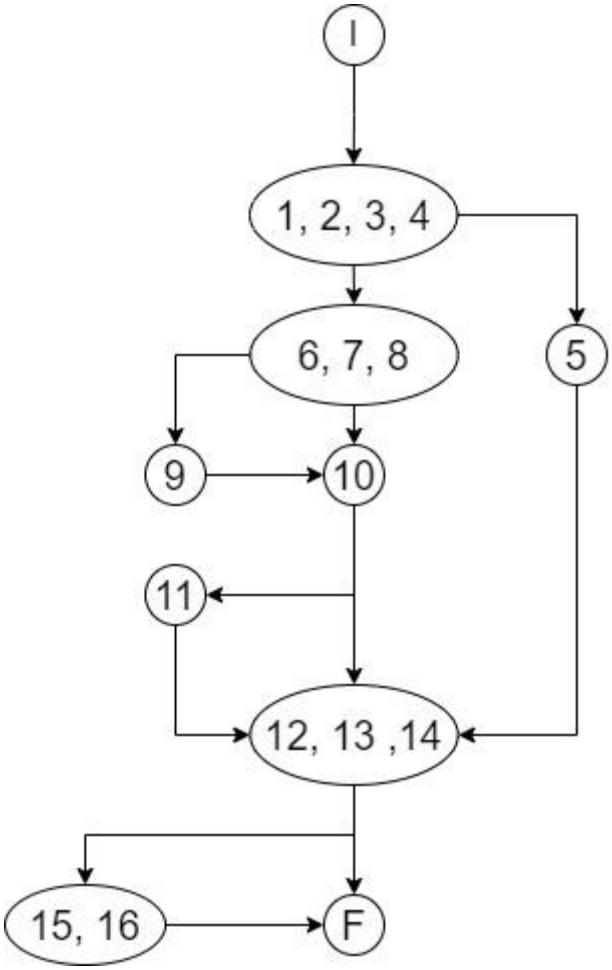
Clase: Controlador

Método: nuevoVehiculo

Test de cobertura

```
public void nuevoVehiculo() {
   String tipo = this.vista.getTipoVehiculo(); #1
   String patente = this.vista.getPatente(); #2
   Vehiculo v = null; #3
   if (tipo.equals("MOTO")) { #4
      v = new Moto(patente); #5
   } else {
      int plazas = this.vista.getPlazas(); #6
      boolean mascota = this.vista.isVehiculoAptoMascota(); #7
      if (tipo.equals("AUTO")) { #8
         v = new Auto(patente, plazas, mascota); #9
      }

      if (tipo.equals("COMBI")) { #10
         v = new Combi(patente, plazas, mascota); #11
      }
   }

   try { #12
      Empresa.getInstance().agregarVehiculo((Vehiculo)v); #13
      this.vista.actualizar(); #14
   } catch (VehiculoRepetidoException var6) {
      VehiculoRepetidoException e = var6; #15
      this.vista.getOptionPane().ShowMessage(e.getMessage()); #16
   }

}
```

Grafo de Flujo



Caminos Básicos

C1 = 1 – 2 – 3 – 4 – 5 – 12 – 13 – 14 – 15 – 16 – F

C2 = 1 – 2 – 3 – 4 – 6 – 7 – 8 – 9 – 10 – 11 – 12 – 13 – 14 – F

C2 contiene al nodo #9 que esta sin pisar.

ID	OBJETIVO DE LA PRUEBA	DATOS DE ENTRADA	PROCEDIMIENTO	SALIDA ESPERADA	RESULTADO
T1	Verificar camino 2	Valores de las variables patente y tipo	-Crear mock de la vista y un Option Panel que será lo que retornará el mock en el getter de OptionPane -Ejecutar la prueba	Éxito	pass

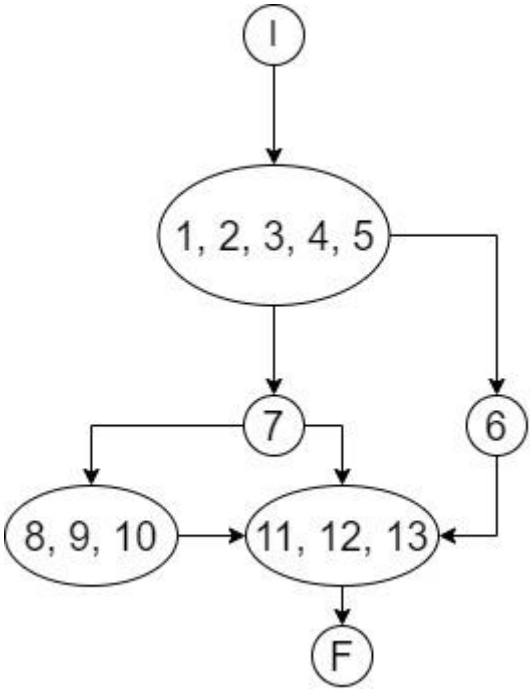
Clase: Controlador

Método: nuevoChofer

Test de cobertura

```
public void nuevoChofer() {
   String tipo = this.vista.getTipoChofer(); #1
   String nombre = this.vista.getNombreChofer(); #2
   String dni = this.vista.getDNIChofer(); #3
   Chofer chofer = null; #4
   if (tipo.equalsIgnoreCase("TEMPORARIO")) { #5
      chofer = new ChoferTemporario(dni, nombre); #6
   } else if (tipo.equalsIgnoreCase("PERMANENTE")) { #7
      int anio = this.vista.getAnioChofer(); #8
      int hijos = this.vista.getHijosChofer(); #9
      chofer = new ChoferPermanente(dni, nombre, anio, hijos); #10
   }

   try { #11
      Empresa.getInstance().agregarChofer((Chofer)chofer); #12
      this.vista.actualizar(); #13
   } catch (ChoferRepetidoException var7) {
   }

}
```

Grafo de Flujo



Caminos Básicos

C1 = 1 – 2 – 3 – 4 – 5 – 6 – 11 – 12 – 13 – F

C2 = 1 – 2 – 3 – 4 – 5 – 7 – 8 – 9 – 10 – 11- 12 – 13 – F

Clase: Controlador

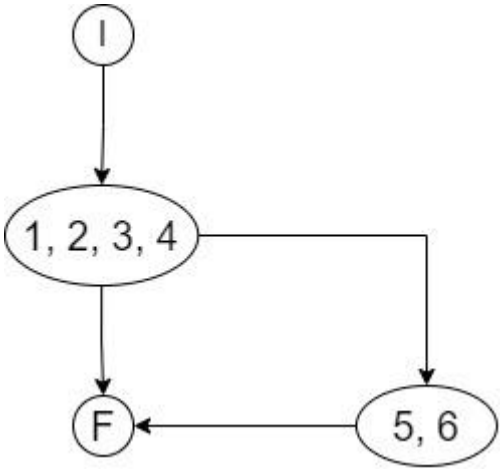
Método: calificarPagar

Test de cobertura

```
public void calificarPagar() {
  int calificacion = this.vista.getCalificacion(); #1

  try { #2
    Empresa.getInstance().pagarYFinalizarViaje(calificacion); #3
    this.vista.actualizar(); #4
  } catch (ClienteSinViajePendienteException var3) {
    ClienteSinViajePendienteException e = var3; #5
```

```
    this.vista.getOptionPane().ShowMessage(e.getMessage());      #6
  }

}
```

Grafo de Flujo



Caminos Básicos

C1 = 1 – 2 – 3 – 4 – 5 – 6 – F

C1 pisa nodos #5 y #6 que están sin pisar.

ID	OBJETIVO DE LA PRUEBA	DATOS DE ENTRADA	PROCEDIMIENTO	SALIDA ESPERADA	RESULTADO

Clase: Controlador

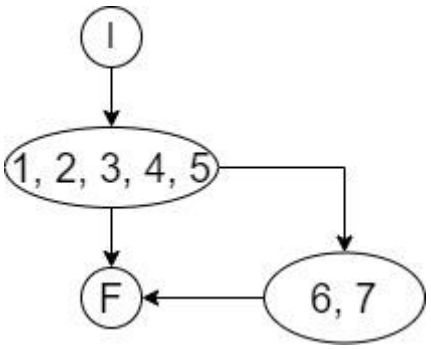
Método: login

Test de cobertura

```
public void login() {
  try { #1
```

```
        String usserName = this.vista.getUsserName(); #2
        String pass = this.vista.getPassword(); #3
        Empresa.getInstance().login(usserName, pass); #4
        this.vista.logearUsuario(); #5
    } catch (PasswordErroneaException | UsuarioNoExisteException var3) {
        Exception e = var3; #6
        #7this.vista.getOptionPane().ShowMessage(((Exception)e).getMessage());
    }
}
```

Grafo de Flujo



Caminos Básicos

C1 = 1 – 2 – 3 – 4 – 5 – 6 – 7 – F

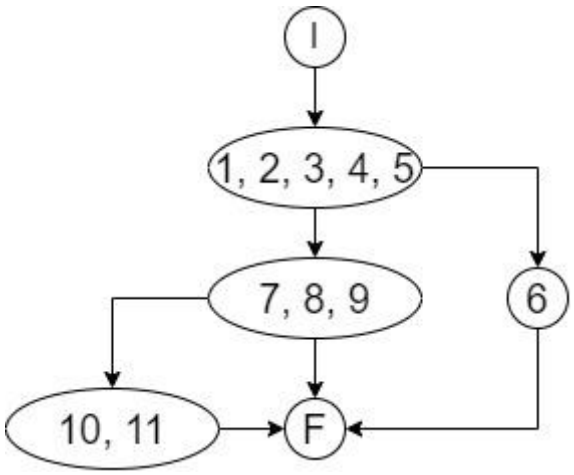
Clase: Controlador

Método: registrar

```
public void registrar() {
    String nombreReal = this.vista.getRegNombreReal(); #1
    String nombreUsuario = this.vista.getRegUsserName(); #2
    String pass = this.vista.getRegPassword(); #3
    String confirm = this.vista.getRegConfirmPassword(); #4
    if (!pass.equals(confirm)) { #5
        this.vista.getOptionPane().ShowMessage(Mensajes.PASS_NO_COINCIDE.getValor()); #6
    } else {
        try { #7
```

```
        Empresa.getInstance().agregarCliente(nombreUsuario, pass, nombreReal); #8
        this.escribir(); #9
      } catch (UsuarioYaExisteException var6) {
        UsuarioYaExisteException e = var6; #10
      #11 this.vista.getOptionPane().ShowMessage(e.getMessage());
      }
    }

}
```

Grafo de Flujo



Caminos Básicos

C1 = 1 – 2 – 3 – 4 – 5 – 6 – F

C2 = 1 – 2 – 3 – 4 – 5 – 7 – 8 – 9 – 10 – 11 – F

Clase: Controlador

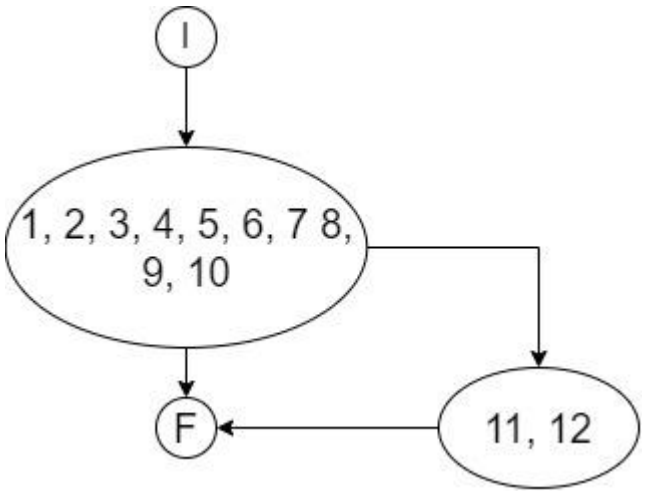
Método: nuevoPedido

Test de cobertura

```
public void nuevoPedido() {
    Cliente cliente = (Cliente)Empresa.getInstance().getUsuarioLogeado(); #1
```

```
    int cantidadPasajeros = this.vista.getCantidadPax(); #2
    boolean mascota = this.vista.isPedidoConMascota(); #3
    boolean baul = this.vista.isPedidoConBaul(); #4
    int km = this.vista.getCantKm(); #5
    String zona = this.vista.getTipoZona(); #6
    Pedido pedido = new Pedido(cliente, cantidadPasajeros, mascota, baul, cantidadPasajeros, zona); #7

    try { #8
        Empresa.getInstance().agregarPedido(pedido); #9
        this.vista.actualizar(); #10
    } catch (ClienteConViajePendienteException | ClienteConPedidoPendienteException |
ClienteNoExisteException | SinVehiculoParaPedidoException var9) {
        Exception e = var9; #11
        #12this.vista.getOptionPane().ShowMessage(((Exception)e).getMessage());
    }

}
```

Grafo de Flujo



Caminos Básicos

C1 = 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10 – 11 – 12 – F

Test de Integración Orientado a Objetos

Caso de Uso 1

Descripción

Un cliente debe iniciar sesión para poder utilizar el sistema, haciendo login con su usuario y password. De no tener tales credenciales, debe registrarse en el sistema para obtener una cuenta.

Un cliente debe iniciar sesión para poder utilizar el sistema, haciendo login con su usuario y password. De no tener tales credenciales, debe registrarse en el sistema para obtener una cuenta.

Actores

Cliente a través de la interfaz de usuario gráfica.

Pre-condiciones

El cliente debe estar registrado en el sistema.

Flujo Normal

1. El cliente accede al sistema.
2. El sistema solicita credenciales.
3. El cliente ingresa escribiendo nombre de usuario y password.

4. El sistema valida las credenciales del cliente y le da la bienvenida.

Flujo Alternativo 1

1. El cliente no tiene una cuenta por lo que decide registrarse en el sistema.
2. El sistema solicita los datos necesarios para crear la cuenta al cliente.
3. El cliente ingresa los datos.
4. El sistema crea la cuenta del cliente.

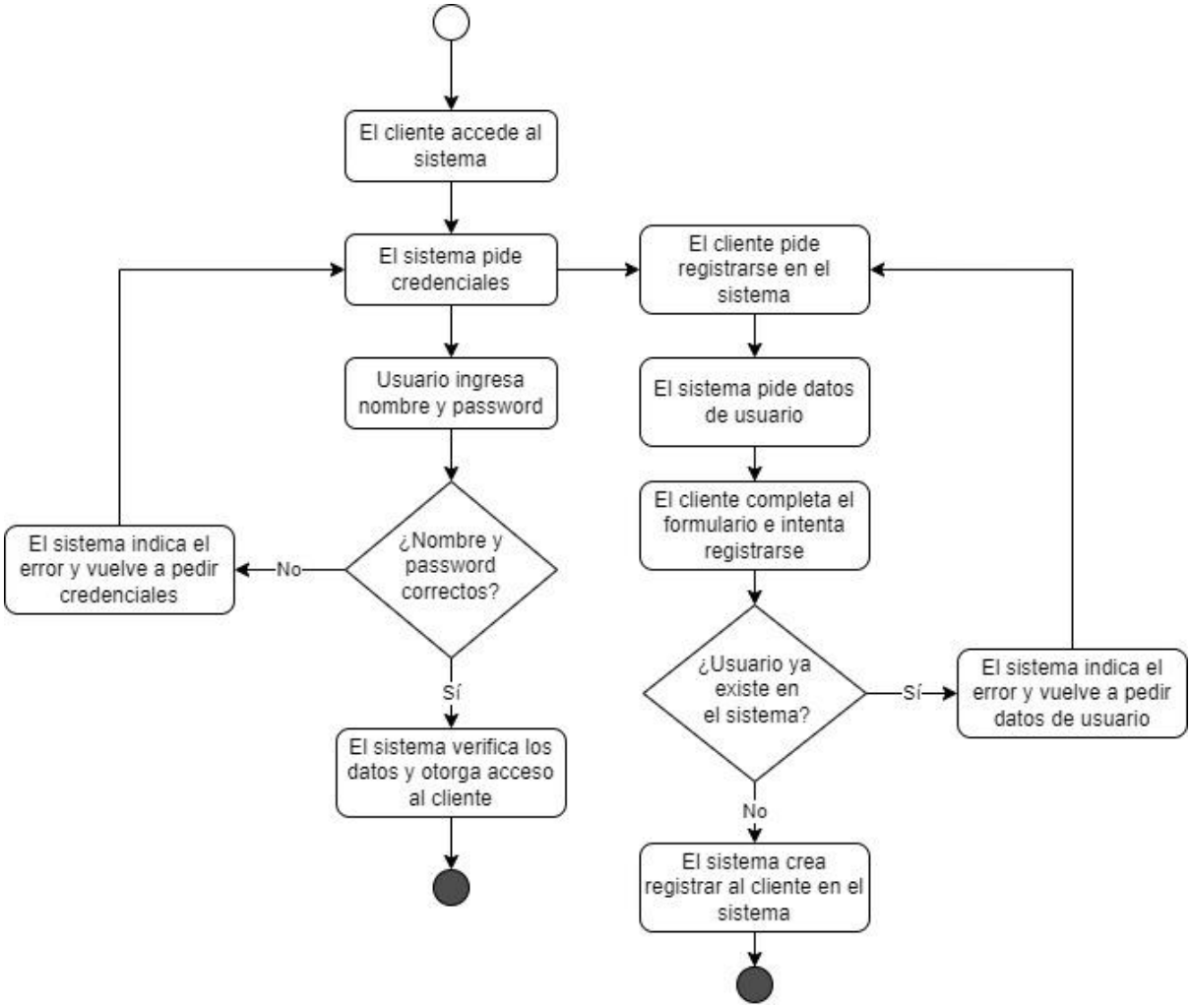
Excepciones

- E1. Password errónea al intentar hacer login. El sistema notifica el error.
- E2. El usuario ingresado no esta registrado en el sistema. El sistema notifica el error.
- E3. El nombre de usuario a registrar ya existe en el sistema. El sistema notifica el error.

Post-condiciones

1. El usuario obtiene acceso al sistema.

Diagrama de actividad



Observaciones

- Al momento de registrarse, si las contraseñas no coinciden se indicará el error al usuario, pero esto no lanzará una excepción, por lo que no se lo tomará en cuenta en el diagrama de actividad ni en los casos de prueba.

Casos de Prueba

Todos los casos de prueba ya fueron contemplados en la prueba de caja negra.

Caso de Uso 2

Descripción

Un cliente necesita transportarse, por lo que utiliza el sistema para solicitar un viaje.

Actores

Cliente a través de la interfaz de usuario gráfica.

Pre-condiciones

El cliente debe haber iniciado sesión.

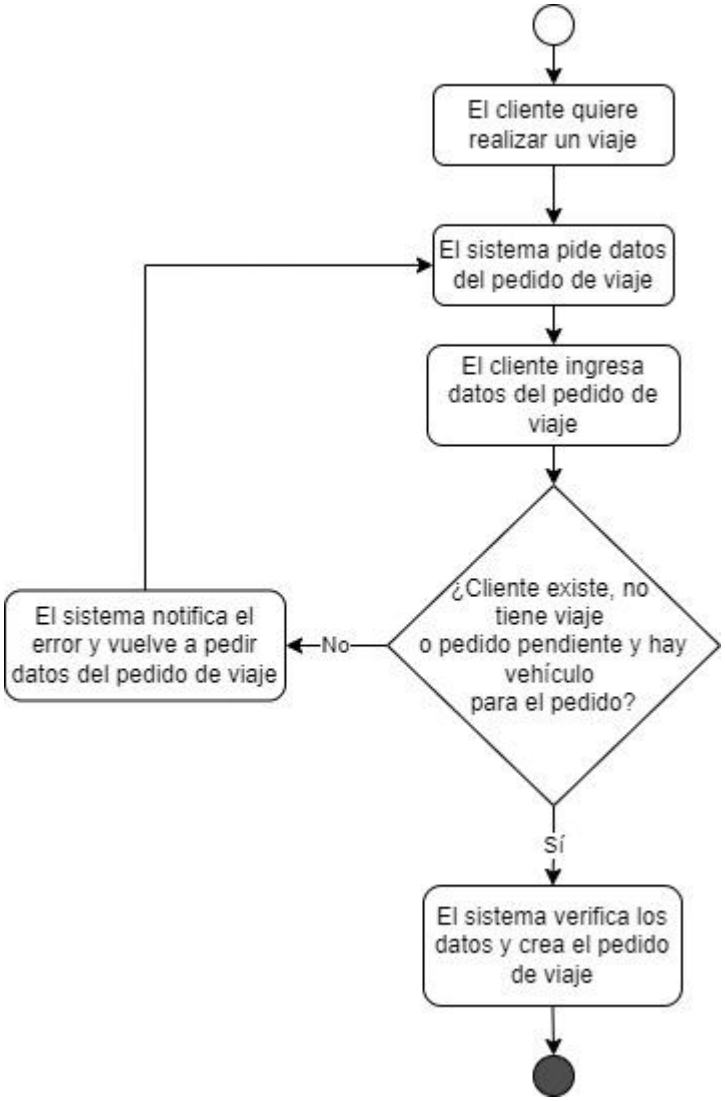
Flujo Normal

1. El cliente realiza un pedido.
2. El sistema solicita datos del pedido.
3. El cliente ingresa los datos del pedido.
4. El sistema crea un pedido, asignándole un chofer, vehículo y cliente.

Excepciones

- E1. El cliente que realizó el pedido no existe en el sistema. El sistema notifica el error.
- E2. El cliente ya tiene un viaje pendiente que debe resolver antes de poder realizar otro pedido. El sistema notifica el error.
- E3. El cliente tiene un pedido pendiente que debe resolver antes de poder realizar otro pedido. El sistema notifica el error.
- E4. No existe vehículo que cumpla con los requisitos del pedido. El sistema notifica el error.

Diagrama de actividad



Casos de Prueba

Todos los posibles casos de prueba ya fueron contemplados en la prueba de caja de negra.

Caso de Uso 3

Descripción

El cliente termina llega a destino y debe finalizar el viaje en el sistema realizando el pago.

Actores

El cliente a través de la interfaz de usuario gráfica.

Pre-condiciones

El cliente debe haber iniciado sesión y ya tiene un viaje pendiente.

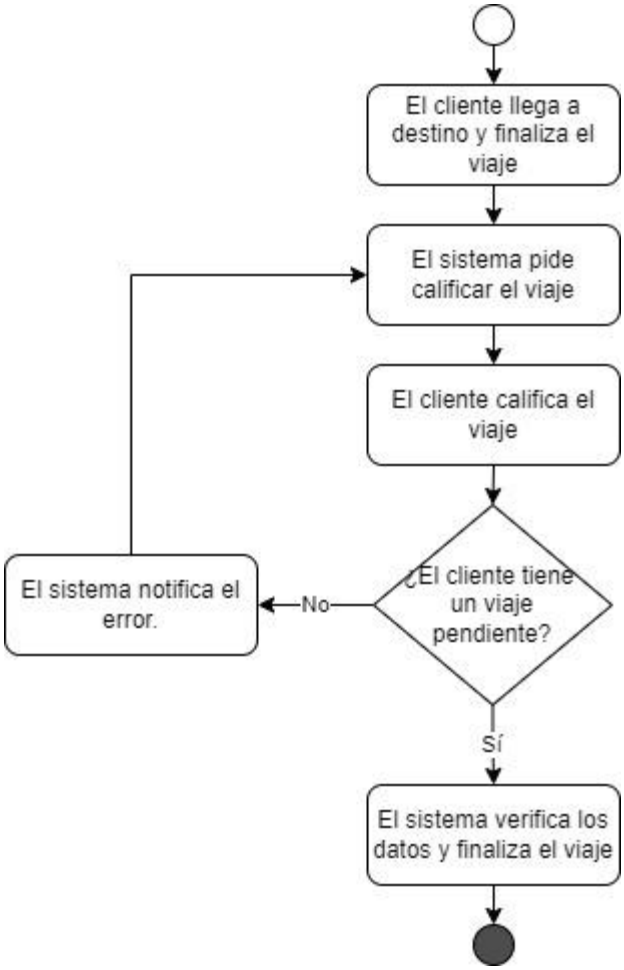
Flujo Normal

1. El cliente finaliza el viaje.
2. El sistema solicita una calificación del viaje.
3. El cliente ingresa la calificación y realiza el pago.
4. El sistema agrega el viaje a la lista de viajes terminados, desocupa al chofer y al vehículo, y quita de la lista de viajes iniciados al cliente.

Excepciones

- E1. El cliente no tiene un viaje pendiente que finalizar. El sistema notifica el error.

Diagrama de actividad



Casos de Prueba

Todos los posibles casos de prueba ya fueron contemplados en la prueba de caja de negra.

Caso de Uso 4

Descripción

El administrador contrata a un nuevo chofer y le da el alta en el sistema.

Actores

El administrador a través del panel de administrador.

Pre-condiciones

El administrador debe haber iniciado sesión en el sistema.

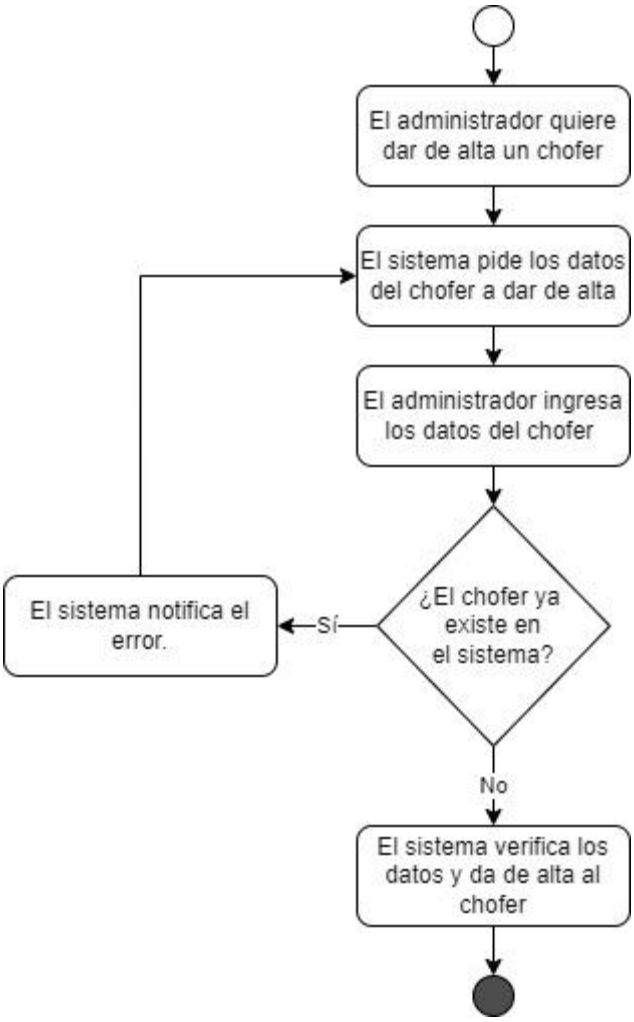
Flujo Normal

1. El administrador da de alta un chofer.
2. El sistema pide datos del chofer.
3. El administrador ingresa datos del chofer.
4. El sistema agrega el chofer a la lista de choferes y a la de choferes desocupados.

Excepciones

- E1. El chofer ya está existe en el sistema.

Diagrama de actividad



Casos de Prueba

Todos los posibles casos de prueba ya fueron contemplados en el test de caja negra.

Caso de Uso 5

Descripción

El administrador quiere de dar de alta un vehículo en el sistema.

Actores

El administrador a través del panel de administrador.

Pre-condiciones

El administrador debe haber iniciado sesión en el sistema.

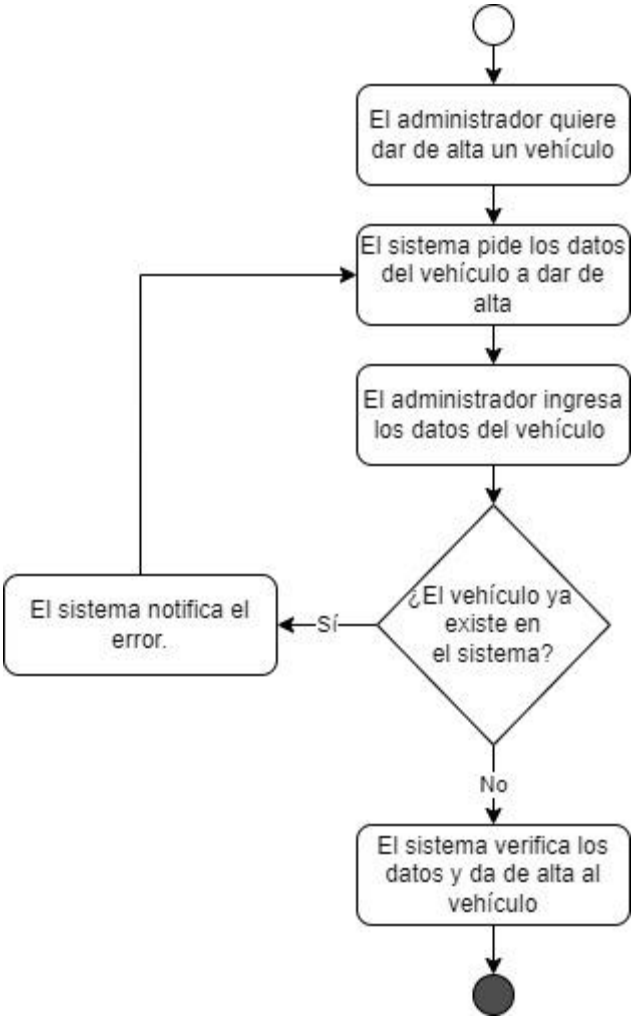
Flujo Normal

1. El administrador quiere dar de alta un vehículo
2. El sistema pide datos del vehículo
3. El administrador ingresa los datos del vehículo
4. El sistema agrega el vehículo a la lista de vehículos y a la de vehículos desocupados.

Excepciones

- E1. El vehículo ya existe en el sistema. El sistema notifica el error al administrador.

Diagrama de actividades



Casos de Prueba

Todos los casos de prueba están contemplados en la prueba de caja negra.

Caso de Uso 6

Descripción

El administrador asigna un vehículo y un chofer al pedido pendiente del cliente para comenzar el viaje.

Actores

El administrador a través del panel de administrador.

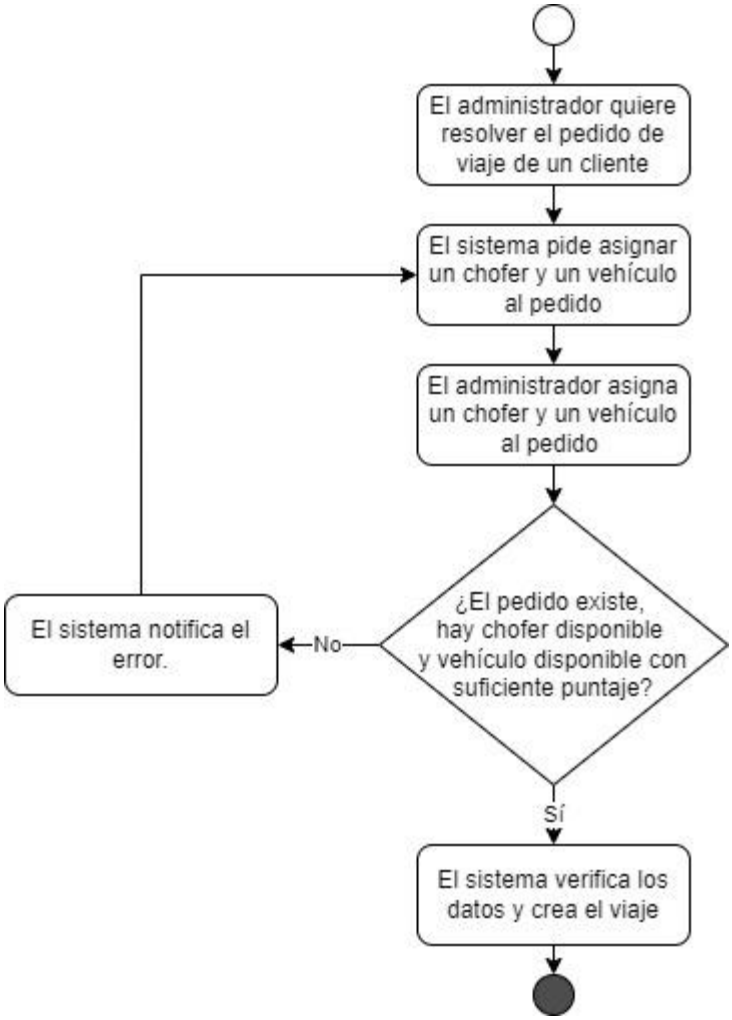
Flujo Normal

1. Asas

Excepciones

- E1. No hay choferes disponibles para realizar el viaje. El sistema notifica el error.
- E2. No hay vehículos disponibles para realizar el viaje. El sistema notifica el error.
- E3. El vehículo no es válido para el viaje debido a que su puntuación es menor a la esperada por el cliente. El sistema notifica el error.
- E4. El pedido no existe en el sistema. El sistema notifica el error.
- E5. El cliente ya tiene un viaje pendiente.

Diagrama de actividad



Casos de prueba

Todos los casos de prueba ya fueron contemplados en la prueba de caja negra.

Conclusión

La realización de este trabajo práctico nos ha servido para poder aplicar los conocimientos vistos en las clases teóricas y así poder ver realmente su uso en la práctica. El nivel de dificultad de la experiencia nos ha parecido correcto ya que si bien no es difícil la implementación de las pruebas si lleva tiempo pensar los casos de prueba y los distintos escenarios para la batería de pruebas.

Este trabajo ha servido para aprender a implementar técnicas tanto de caja negra como de caja blanca. Y también poder automatizar pruebas utilizando el robot que ha servido para facilitar el testing de la GUI.

En conclusión, el principal aprendizaje obtenido a través de este trabajo fue comprender que el testing es un proceso estructurado, respaldado por una teoría sólida, y no simplemente el ingreso de valores aleatorios para probar los distintos métodos. Esto nos permitió no solo evaluar de forma unitaria cada componente, sino también analizar cómo se integran entre sí, validando así la comunicación y el manejo de variables y estados a lo largo del sistema. Así, buscamos verificar que el sistema responda de acuerdo a los requerimientos pre establecidos, identificando las fallas a nivel unitario (métodos) y entre módulos antes de que afecten el funcionamiento general.

De esta manera comprendimos que la SRS y el contrato son lo más importante del testing ya que toda la información necesaria se encuentra allí. Utilizándolos correctamente es posible conocer los valores que el sistema debe devolver o los estados que debe adoptar para determinadas entradas y así poder evaluar el correcto funcionamiento del sistema.