TALLER DE PROGRAMACIÓN I

Trabajo Final Testing

Integrantes:

- Ivan Ahumada
- Ezequiel Arce
- Gian Franco Magliotti
- Micaela Rasso

13/11/2024

Índice

Introducción **3**

Metodología **4**

Test de capa de datos **5**
- Clase Cliente: 5
- Clase Chofer: 6
- Clase ChoferTemporario: 7
- Clase ChoferPermanente: 7
- Clase Combi: 10
- Clase Auto: 12
- Clase Moto: 14
- Clase Pedido: 16
- Clase Viaje: 17
- Resultados test de capa de datos 20

Test de capa de negocio **20**
- Escenario N°1 21
- Resultados del Escenario N°1 39
- Escenario N°2 39
- Resultados del Escenario N°2 46
- Escenario N°3 46
- Resultados del Escenario N°3 51
- Escenario N°4 52
- Resultados del Escenario N°4 60

Test de controlador/integración **60**
- Casos de uso 61
- Resultados test de integración 81

Test de persistencia **81**
- Resultados test de persistencia 85

Test de GUI **86**
- Resultados test de GUI 87

Conclusiones **88**

Introducción

La finalidad del presente trabajo es informar el proceso de testing del software perteneciente a una aplicación de remises de nombre “Subí Que Te Llevo” proveída por la cátedra de Taller de Programación I, Facultad de Ingeniería, U.N.M.D.P.

A lo largo del mismo, se presenta la forma en que se desarrollaron las pruebas para cada módulo del sistema así como los resultados obtenidos luego de ejecutar las correspondientes batería de pruebas. El testing abarca pruebas unitarias, de integración, de persistencia e interfaces gráficas.

La principal herramienta utilizada para automatizar la ejecución de las pruebas unitarias fue la biblioteca JUnit. La misma se eligió puesto que proporciona un framework que obliga a implementar las pruebas en un formato estándar que puede ser reutilizable y entendible por cualquiera que esté familiarizado con la librería. El aplicar este framework también ayudó a tener una batería de pruebas ordenada, que pueda ser ejecutada fácilmente y que muestre los resultados de forma clara mediante una interfaz gráfica que proporciona la herramienta.

Metodología

Para las pruebas unitarias, se optó por el método de *caja negra*, dado que no se tenía acceso al código fuente del sistema. A partir de los requisitos y especificaciones proporcionadas, se elaboraron los casos de prueba para cada clase, utilizando herramientas como las tablas de particiones y las baterías de prueba. Una vez definidos los casos de prueba, se ejecutaron las baterías de prueba con ayuda de la biblioteca JUnit.

En cuanto a las pruebas de integración, se utilizaron los casos de uso del sistema como base para su definición. A partir de estos casos de uso, se generaron los diagramas de secuencia, de los cuales se derivaron los casos de prueba a evaluar.

Por último, las pruebas de GUI se automatizaron con ayuda de algunas clases proveídas por la biblioteca AWT de Java, que permitieron simular el comportamiento del usuario en las interfaces.

Test de capa de datos

A continuación se presentan las tablas de particiones y batería de pruebas correspondiente a cada método de cada clase dentro del modelo de dominio.

Clase Cliente:

Metodo: Cliente**(**String nombreUsuario, String pass, String nombreReal**)**

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

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
nombreUsuario	nombreUsuario == null	No	-
nombreUsuario	nombreUsuario != null	Si	1
nombreUsuario	len(nombreUsuario)<1	No	-
nombreUsuario	len(nombreUsuario)>=1	Si	2
pass	pass == null	No	-
pass	pass != null	Si	3
pass	len(pass)<1	No	-
pass	len(pass)>=1	Si	4
nombreReal	nombreReal == null	No	-
nombreReal	nombreReal != null	Si	5
nombreReal	len(nombreReal)<1	No	-
nombreReal	len(nombreReal)>=1	Si	6

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	nombreUsuario	lionel10	Se instancio correctamente un Cliente con el estado deseado	1
	pass	12345678		
	nombreReal	Lionel Cuccitini		

Clase Chofer:

Método: void setSueldoBasico(double sueldoBasico)

Precondiciones: el parámetro sueldoBasico es positivo

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
sueldoBasico no positivo	sueldoBasico<=0	No	-
sueldoBasico positivo	sueldoBasico>0	Si	1

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	sueldoBasico	1000	Se cambio el valor del atributo sueldoBasico de la clase Chofer a 1000	1

Método: double getSueldoNeto()

Retorno: el valor del sueldo del Neto del chofer, el cual corresponde al 86% del sueldo bruto, luego de realizadas las correspondientes retenciones.

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
chofer váildo	chofer != null	Si	1

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	chofer	chofer = new ChoferTemporario("12345678","John Smith")	860	1

Clase ChoferTemporario:

Método: ChoferTemporario(String dni,String nombre)

Precondiciones: Los parámetros dni y nombre son distintos de null y tienen al menos un carácter.

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
dni	dni == null	No	-
dni	dni != null	Si	1
dni	len(dni) < 1	No	-
dni	len(dni) > = 1	Si	2
nombre	nombre == null	No	-
nombre	nombre != null	Si	3
nombre	len(nombre)<1	No	-
nombre	len(nombre)>=1	Si	4

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	dni	12345678	Se instancio correctamente un ChoferTemporario con el estado deseado	1
	Nombre	John Smith		

Clase ChoferPermanente:

Método: ChoferPermanente(String dni,String nombre, int anioIngreso, int cantidadHijos)

Precondiciones:

- Los parámetros dni y nombre son distintos de null y tienen al menos un carácter.
- El parámetro anioIngreso está comprendido entre 1900 y 3000
- El parámetro cantidadHijos es mayor o igual que cero.

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
dni	== null	No	-
dni	== vacio	No	-
dni	!= null & len(dni)	Si	1
nombre	== null	No	-
nombre	== vacio	No	-
nombre	!= null & len(nombre)	Si	2
anioIngreso	anioIngreso<=1900	No	-
anioIngreso	3000<=anioIngreso	No	-
anioIngreso	1900 < anioIngreso < 3000	Si	3
cantidadHijos	cantidadHijos<0	No	-
cantidadHijos	0<cantidadHijos	Si	4

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	dni	"23249123"	Se instancio correctamente el chofer permanente con el estado deseado	1,2,3,4
	nombre	"Martin Salamanca"		
	anioIngreso	1990		
	Cantidad de hijos	3		

Método: int getAntiguedad()

Retorno: diferencia entre el año actual y el año de ingreso

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	choferPermanente	ChoferPermanente("23249123","Martin Salamanca",1990,3)	34	1

Método: void setCantidadHijos(int cantidadHijos)

Precondiciones: El parámetro cantidadHijos es mayor o igual que cero.

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
cantidadHijos	x<0	No	-
cantidadHijos	x>=0	Si	1

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	cantidadHijos	5	Se cambio el atributo cantidadHijos por 5	1

Metodo: double getSueldobruto()

Contrato: 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.

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
ChoferPermanente válido	ChoferPemanente !=null	Si	1

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	ChoferPermanente válido	ChoferPermanente("23249123","Martin Salamanca",2020,3,1000)	1410	1.1
2	ChoferPermanente válido	ChoferPermanente("23249123","Martin Salamanca",1990,3,1000)	2210	1.2

Clase Combi:

Método: Combi(String patente, int cantidadPlazas, boolean mascota)

Precondiciones:

- El parámetro Patente es distinto de null.
- El parámetro cantidadPlazas es mayor que 4 y menor que 11

Tabla de Particiones

Dato de entrada	Descripcion de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
patente	patente == null	No	-
patente	patente != null	Si	1
cantidadPlazas	cantidadPlazas < 4	No	-
cantidadPlazas	cantidadPlazas > 10	No	-
cantidadPlazas	cantidadPlazas >= 4 && <=10	Si	2
mascota	boolean	SI	3

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	patente	"AABBCC123"	Se instancio correctamente el estado de Combi deseado	1
	cantidadPlazas	7		
	mascota	TRUE		

Metodo: Integer getPuntajePedido(Pedido pedido)

Precondiciones: el parámetro pedido es distinto de null

Retorno: el puntaje del vehículo en relación al pedido en cuestión de acuerdo a la siguiente fórmula: El valor es 10 * cantPax

Tabla de Particiones

Dato de entrada	Descripcion de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
Pedido nulo	== null	No	-
Pedido realizable	!= null y el vehiculo puede realizar el pedido	Si	1

Pedido no realizable	!= null y el vehiculo no puede realizar el pedido	Si	2

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	Pedido con baul	cliente = Cliente("Esteban Gutierrez", "12345678", "estaban1200") cantidadPasajeros = 3 mascota = False baul = true km = 3 zona = "ZONA_SIN_ASFALTAR"	130	1.1
2	Pedido sin baul	cliente = Cliente("Esteban Gutierrez", "12345678", "estaban1200") cantidadPasajeros = 3 mascota = false baul = false km = 3 zona = "ZONA_SIN_ASFALTAR"	30	1.2
3	Pedido con más pasajeros de la cap del vehiculo	cliente = Cliente("Esteban Gutierrez", "12345678", "estaban1200") cantidadPasajeros = 9 mascota = False baul = true km = 3 zona = "ZONA_SIN_ASFALTAR"	null	2.1
4	Pedido con mascota	cliente = Cliente("Esteban Gutierrez", "12345678", "estaban1200") cantidadPasajeros = 9 mascota = true	null	2.2

		baul = true		
		km = 3		
		zona = "ZONA_SIN_ASFALTAR"		

Clase Auto:

Método: Auto(String patente, int cantidadPlazas, boolean mascota)

Precondiciones:

- El parámetro Patente es distinto de null.
- El parámetro cantidadPlazas es positivo y menor que 5

Tabla de Particiones

Dato de entrada	Descripcion de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
patente	== null	No	-
patente	== vacio	No	-
patente	!= vacio	Si	1
cantidadPlazas	0<=cantidadPlazas	No	-
cantidadPlazas	5<=cantidadPlazas	No	-
cantidadPlazas	0<cantidadPlazas<5	Si	2
mascota	boolean	Si	3

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	patente	"NFG 123"	Se creó exitosamente una instancia de Auto con el estado deseado	1,2,3
	cantidadPlazas	"2"		
	mascota	TRUE		

Metodo: getPuntajePedido(Pedido pedido)

Precondiciones: el parámetro pedido es distinto de null

Retorno: el puntaje del vehículo en relación al pedido en cuestión de acuerdo a la siguiente formula: Si el pedido solicita uso de baúl el valor es 40 * cantPax

Tabla de Particiones

Dato de entrada	Descripcion de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
pedido	== null	No	-
pedido	!= null y el vehiculo puede realizar el pedido	Si	1
pedido	!= null y el vehiculo no puede realizar el pedido	Si	2

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	pedido	Pedido(Cliente("Esteban Gutierrez", "12345678", "estaban1200") ,	120	1.1
		3, FALSE, TRUE, 3, "ZONA_SIN_ASFALTAR"		
2	pedido	Pedido(Cliente("Esteban Gutierrez", "12345678", "estaban1200") ,	90	1.2
		3, FALSE, FALSE, 3, "ZONA_SIN_ASFALTAR"		
3	pedido	Pedido(Cliente("Esteban Gutierrez", "12345678", "estaban1200") ,	null	2.1
		4, FALSE, FALSE, 3, "ZONA_SIN_ASFALTAR"		
4	pedido	Pedido(Cliente("Esteban Gutierrez", "12345678", "estaban1200") ,	null	2.2
		4, FALSE, TRUE, 3, "ZONA_SIN_ASFALTAR"		

Clase Moto:

Método: Moto(String patente)

Precondiciones:

- El parámetro Patente es distinto de null.
- El atributo cantidadPlazas se inicializa en 1
- El atributo mascota se inicializa en false

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
patente	patente !=null	Si	1
patente	patente == null	No	-

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	Patente válida	JWL 456	se instancio correctamente una moto con la patente JWL 456	1

Metodo: getPuntajePedido(Pedido pedido)

Precondiciones: el parámetro pedido es distinto de null.

Retorno: Si el pedido solicita solo 1 pasajero sin uso de baúl y sin traslado de mascota se retorna 1000. Se retorna null en caso contrario.

Tabla de Particiones

Dato de entrada	Descripcion de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
Pedido	== null	No	-
Pedido	!= null y el vehiculo puede realizar el pedido	Si	1
Pedido	!= null y el vehiculo no puede realizar el pedido	Si	2

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases de equivalencias abarcadas
1	Pedido	cliente = Cliente("Esteban Gutierrez", "12345678", "estaban1200") cantidadPasajeros = 1 mascota = false baul = false km = 3 zona = "ZONA_SIN_ASFALTAR"	1000	1
2	Pedido	cliente = Cliente("Esteban Gutierrez", "12345678", "estaban1200") cantidadPasajeros = 1 mascota = False baul = true km = 3 zona = "ZONA_SIN_ASFALTAR"	null	2.1
3	Pedido	cliente = Cliente("Esteban Gutierrez", "12345678", "estaban1200") cantidadPasajeros = 2 mascota = False baul = false km = 3 zona = "ZONA_SIN_ASFALTAR"	null	2.2
4	Pedido	cliente = Cliente("Esteban Gutierrez", "12345678", "estaban1200") cantidadPasajeros = 1 mascota = true baul = false km = 3 zona = "ZONA_SIN_ASFALTAR"	null	2.3

Clase Pedido:

Método: Pedido(Cliente cliente, int cantidadPasajeros, boolean mascota, boolean baul, int km, String zona)

Precondiciones:

- El parámetro cliente es distinto de null
- El parámetro cantidadPasajeros es mayor que 1
- El parámetro km es mayor o igual que cero
- El parámetro zona corresponde a alguna de las siguientes constantes: Constantes.ZONA_PELIGROSA , Constantes.ZONA_SIN_ASFALTAR o Constantes.ZONA_STANDARD

Tabla de Particiones

Dato de entrada	Descripcion de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
cliente	cliente == null	No	-
cliente	cliente != null	Si	1
cantidadPasajeros	cantidadPasajeros < 1	No	-
cantidadPasajeros	cantidadPasajeros >= 1	Si	2
km	km <= 0	No	-
km	km > 0	Si	3
mascota	boolean	Si	4
baul	boolean	Si	5
zona	"ZONA_PELIGROSA" || "ZONA_SIN_ASFALTAR" || "ZONA_STANDARD"	SI	6
zona	String no perteneciente a zonas	No	-

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	Cliente	Cliente("ramonDiaz","12345678","Ramon Diaz")	Se instancio correctamente un pedido con el estado deseado	1,2,3,4,5,6
	cantidadPasajeros	4		
	mascota	TRUE		

	baul	TRUE		
	km	10		
	zona	"ZONA_STANDARD"		

Clase Viaje:

Método: Viaje(Pedido pedido, Chofer chofer, Vehículo vehiculo)

Precondiciones: Los parámetros pedido, chofer, y vehículo son distintos de null Setea el atributo calificación en cero y el atributo finalizado en false.

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
pedido	pedido == null	No	-
pedido	pedido != null	Si	1
chofer	chofer == null	No	-
chofer	chofer != null	Si	2
vehiculo	vehiculo == null	No	-
vehiculo	vehiculo != null	Si	3

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	Pedido	Pedido(Cliente("ramonDiaz","12345678","Ramon Diaz"),	Se instancio correctamente un Viaje con el estado deseado	1,2,3
		4, true, true, 10, "ZONA_STANDARD")		
	Chofer	ChoferTemporario("22312151","Joaquin Valiente")		
	Vehiculo	Auto("ASD 333",4,true)		

Método: finalizarViaje(int calificacion)

Precondiciones: el parametro calificacion es mayor o igual que cero y menor o igual que 5 Setea el atributo calificacion de acuerdo al parametro calificacion.

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
calificacion	calificacion >=0 && <=5	Si	1
calificacion	calificacion <0	No	-
calificacion	calificacion >5	No	-

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	Calificacion	4	finalizado = true y calificaicion = 4	1

Método: getValor()

Retorno:
Retorna el valor del viaje de acuerdo a la siguiente fórmula:

- Al valor valorBase del viaje se le realizan incrementos por cantidad de pasajeros y por km recorridos.
- Si la zona del pedido es Constantes.ZONA_STANDARD se incrementa un 10% por cada pasajero y 10% por cada km.
- Si la zona del pedido es Constantes.ZONA_SIN_ASFALTAR se incrementa un 20% por cada pasajero y 15% por cada km.
- Si la zona del pedido es Constantes.ZONA_PELIGROSA se incrementa un 10% por cada pasajero y 20% por cada km.
- Por traslado de Mascota:
    - En caso de trasladar mascota se incrementa un 10% por cada pasajero y 20% por cada km.
- Por uso de baul:
    - En caso de traslado de equipaje en baúl se incrementa un 10% por cada pasajero y 5% por cada km.

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
Viaje	Viaje por Zona Estandar	Sí	1
Viaje	Viaje por Zona Peligrosa	Sí	2
Viaje	Viaje por Zona sin asfaltar	Sí	3
Viaje	Viaje con mascota	Sí	4
Viaje	VIaje con baul	Sí	5

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	Viaje	Viaje(Pedido(Cliente("ramonDiaz","12345678","Ramon Diaz"), 4,FALSE,FALSE,10,"ZONA_STANDARD"), ChoferTemporario("22312151","Joaquin Valiente"), Auto("ASD 333",4,true))	3600	1
2	Viaje	Viaje(Pedido(Cliente("ramonDiaz","12345678","Ramon Diaz"), 4,FALSE,FALSE,10,"ZONA_PELIGROSA"), ChoferTemporario("22312151","Joaquin Valiente"), Auto("ASD 333",4,true))	5100	2
3	Viaje	Viaje(Pedido(Cliente("ramonDiaz","12345678","Ramon Diaz"), 4,FALSE,FALSE,10,"ZONA_SIN_ASFALTAR"), ChoferTemporario("22312151","Joaquin Valiente"), Auto("ASD 333",4,true))	4950	3
4	Viaje	Viaje(Pedido(Cliente("ramonDiaz","12345678","Ramon Diaz"), 4,TRUE,FALSE,10,"ZONA_STANDARD"), ChoferTemporario("22312151","Joaquin Valiente"), Auto("ASD 333",4,true))	8700	4
5	Viaje	Viaje(Pedido(Cliente("ramonDiaz","12345678","Ramon Diaz"), 4,FALSE,TRUE,10,"ZONA_STANDARD"), ChoferTemporario("22312151","Joaquin Valiente"), Auto("ASD 333",4,true))	4920	5

Resultados test de capa de datos

A continuación se presentan los errores encontrados luego de ejecutar las baterías de prueba diseñadas para el testeo de la capa de datos.

Nombre de la prueba	Error
getValor_STANDARD	El valor del viaje debería ser 3600, pero fue 1500
getValor_CONBAUL	El valor del viaje debería ser 4950 pero fue 1500
getValor_CONMASCOTA	El valor del viaje debería ser 8700 pero fue 5100
getPuntajePedido_Clase1_2	El puntaje debe ser 90 pero fue 120
getPuntajePedido_Clase1_1	El puntaje debe ser 130 pero fue 10
getPuntajePedido_Clase2_1	El puntaje debe ser null pero fue 90
getSueldoBruto_Clase1_2	El sueldo bruto debería ser 2210 pero fue 2910

Test de capa de negocio

El test de la capa de negocio fue realizado en 4 escenarios distintos para facilitar la prueba de métodos cuyos parámetros dependen del estado de la Empresa. En el caso del software bajo testing, la capa de negocio está conformada por una clase llamada Empresa, por lo tanto, para realizar el estudio de la capa de negocio se prueban todos los métodos de la clase Empresa.

En el caso de métodos como agregarPedido() o crearViaje(), los cuales tienen lanzamiento de muchas excepciones distintas que dependen del estado de la Empresa y no de los parámetros que recibe, la tabla de particiones fue separada de forma que las clases de equivalencia fueran acorde al estado de la Empresa propuesto en el escenario.Esto quiere decir que para determinado escenario, no necesariamente se cubren la totalidad de las clases de equivalencia.

En algunos casos en las tablas de particiones y en las baterías de pruebas se utilizaron abreviaturas con respecto a las objetos que referencian, con el objetivo de no poner reiteradamente el estado del objeto para definirlo, la definición de cada una de estas abreviaturas se realizó en los distintos escenarios desarrollados de la siguiente manera: abreviatura = Objeto().

A continuación se presentan las tablas de particiones y batería de pruebas correspondiente a cada método de la clase Empresa, además el escenario en el que se desarrollaron las pruebas.

Escenario N°1

HashMap clientes	{}
HashMap choferes	{}
HashMap vehiculos	{}
HashMap pedidos	{}
HashMap viajesIniciados	{}
HashMap viajesTerminados	{}
HashMap choferesDisponibles	{}
HashMap vehiculosDisponibles	{}
usuarioLogueado	NULL

Método: void agregarChofer(Chofer chofer)
Precondiciones: chofer es distinto de null

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
chofer	chofer == null	No	-
chofer	chofer != null	Si	1

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	Chofer	ChoferTemporario("32909289", "Julian")	Se añadió correctamente a la colección de choferes	1

Método: void agregarCliente(String usuario, String pass, String nombreReal)
Precondiciones:

- usuario distinto de null y con al menos un carácter
- pass distinto de null y con al menos un carácter
- nombreReal distinto de null y con al menos un carácter

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
usuario	usuario != ""	Si	1
usuario	usuario != null	Si	2
usuario	usuario == null	No	-
usuario	usuario == ""	No	-
pass	pass != ""	Si	3
pass	pass != null	Si	4
pass	pass == null	No	-
pass	pass == ""	No	-
nombreReal	nombreReal != ""	Si	5
nombreReal	nombreReal != null	Si	6
nombreReal	nombreReal == null	No	-
nombreReal	nombreReal == ""	No	-

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
	usuario	"marki3000"		
	pass	"12345678"		
1	nombreReal	"Marcos Gutierrez"	Se añadio correctamente a la colección de clientes	1,2,3,4,5,6

Método: void agregarVehiculo(Vehiculo vehiculo)
Precondiciones: vehiculo es distinto de null

Tabla de Particiones

Dato de entrada	Descripción de la	Aplica	Identificador de clase

	clase de equivalencia		de equivalencia
vehiculo	vehiculo == null	No	-
vehiculo	vehiculo != null	Si	1

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	vehiculo	Auto("JJ888JJ" , 3, false)	Se añadio correctamente a la colección de clientes	1

Método: void agregarPedido(Pedido pedido)
Precondiciones: pedido es distinto de null

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
pedido	pedido == null	No	-
pedido	pedido != null	Si	1
pedido	pedido con cliente que no existe en colección	Si	2

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	pedido	Pedido(Cliente("Esteban Gutierrez", "12345678", "estaban1200"), true, ture, 10,ZONA_ESTANDAR)	ClienteNoExisteException	1,2

Método: boolean validarPedido(Pedido pedido)
Precondiciones: pedido es distinto de null

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

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
pedido	pedido == null	No	-
pedido	pedido != null	Si	1

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	pedido	Pedido(Cliente("Esteban Gutierrez", "12345678", "estaban1200"), 1, true, ture, 10,ZONA_ESTANDAR)	FALSE	1

Método: void crearViaje(Pedido pedido, Chofer chofer, Vehiculo vehiculo)
Precondiciones:

- pedido es distinto de null
- chofer es distinto de null
- vehiculo es distinto de null

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
pedido	pedido == null	No	-
pedido	pedido != null	Si	1
pedido	pedido que no pertenece a la coleccion pedidos	Si	2
pedido	pedido con cliente que no tiene viaje	Si	3

	iniciado		
chofer	chofer == null	No	-
chofer	chofer != null	Si	4
chofer	chofer que no pertenece a la coleccion choferesDisponibles	Si	5
vehiculo	vehiculo == null	No	-
vehiculo	vehiculo != null	Si	6
vehiculo	vehiculo que satisface las caracteristicas del pedido	Si	7
vehiculo	vehiculo que no satisface las caracteristicas del pedido	Si	8
vehiculo	vehiculo que no pertenece a la coleccion vehiculosDisponibles	Si	9

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
	pedido	Pedido (Cliente("Esteban Gutierrez", "12345678", "estaban1200"), 1, true, true, 10, ZONA_ESTANDAR)		
	chofer	ChoferTemporario("32909289", "Julian")		
1	vehiculo	Auto("JJ888JJ" , 3, false)	Se lanzó una excepción (pedidoInexistenteException, ChoferNoDisponibleException, VehiculoNoValidoException, VehiculoNoDisponible)	1,2,3,4,5,6, 7,8,9

Método: HashMap<String,Chofer> getChoferes()
Retorno: HashMap[String,Chofer] donde el dni del chofer es la clave de cada chofer, contiene todos los choferes registrados de la empresa

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	HashMap<String,Chofer> = {}	-

Método: ArrayList<Chofer> getChoferesDesocupados()
Retorno: ArrayList de Chofer representa los choferes que no están realizando un viaje

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	ArrayList<Chofer> = {}	-

Método: HashMap<String,Cliente> getClientes()
Retorno: HashMap[String,Cliente] donde el nombre de usuario es la clave de cada cliente, contiene todos los clientes registrados en la empresa

Tabla de Particiones

Dato de entrada	Descripción de la clase de	Aplica	Identificador de clase de equivalencia

	equivalencia		
NO TIENE DATOS DE ENTRADA	-	-	-

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	HashMap<String,Cliente> = {}	-

Método: ArrayList<Viaje> getHistorialViajeChofer(Chofer chofer)
Precondiciones: chofer es distinto de null
Retorno: ArrayList de Viaje correspondiente a los viajes realizados por el chofer en cuestión

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
Chofer	== null	No	-
Chofer	!= null	Si	1

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	Chofer	ChoferTemporario("32909289", "Julian")	ArrayList <Viaje> = {}	1

Método: ArrayList<Viaje> getHistorialViajeCliente(Cliente cliente)
Precondiciones: cliente es distinto de null
Retorno: ArrayList de Viaje correspondiente a los viajes realizados por el cliente en cuestión

Tabla de Particiones

Dato de entrada	Descripción de la clase de	Aplica	Identificador de clase de equivalencia

	equivalencia		
Cliente	== null	No	-
Cliente	!= null	Si	1

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	Cliente	Cliente("Matis Ramirez", "12345678", "matiRAMI")	ArrayList <Viaje> = {}	1

Método: HashMap<Cliente,Pedido> getPedidos()
Retorno: HashMap [Cliente, Pedido], donde la clave es el cliente que realizo el pedido, representa todos los pedidos realizados por los clientes

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
NO TIENE DATOS DE ENTRADA	-	-	-

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	HashMap<Cliente, Pedido> = {}	-

Método: Usuario getUsuarioLogeado
Retorno: Retorna el usuario logueado(Administrador o Cliente) en el sistema

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
NO TIENE DATOS DE ENTRADA	-	-	-

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	null	-

Método: HashMap<String,Vehiculo> getVehiculos()
Retorno: HashMap[String, Vehiculo] donde la patente es la clave para cada vehículo, representa todos los vehículos registrados en el sistema

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
NO TIENE DATOS DE ENTRADA	-	-	-

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	HashMap<String, Vehiculo> = {}	-

Método: ArrayList<Vehiculo> getVehiculosDesocupados()
Retorno: ArrayList[Vehiculo] que representa los vehiculos que no están realizando un viaje

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
NO TIENE DATOS DE ENTRADA	-	-	-

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	ArrayList<Vehiculo> = {}	-

Método: HashMap<Cliente,Viaje> getViajesIniciados()
Retorno: HashMap[Cliente,Viaje], donde la clave es el Cliente que solicitó el Viaje, representa los viajes iniciados pero no terminados

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
NO TIENE DATOS DE ENTRADA	-	-	-

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	HashMap<Cliente, Viaje> = {}	-

Método: ArrayList<Viaje> getViajesTerminado()
Retorno: ArrayList[Viaje], representa los viajes ya finalizados

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	HashMap<Cliente, Viaje> = {}	-

Método: void setChoferes(HashMap<String,Chofer> choferes)

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
choferes	HashMap<String,Chofer> != null	Si	1

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	choferes	HashMap<String,Chofer> = {}	Se cambió exitosamente el valor del atributo choferes por la referencia pasada por parámetro	1

Método: void setChoferesDesocupados(ArrayList<Chofer> choferesDesocupados)

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
choferesDesocupados	ArrayList<Chofer> != null	Si	1

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	choferesDesocupados	ArrayList<Chofer> = {}	Se cambió exitosamente el valor del atributo choferesDesocupados por la referencia pasada por parámetro	1

Método: void setCliente(HashMap<String,Cliente> clientes)

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
clientes	HashMap<String,Cliente> != null	Si	1

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	clientes	HashMap<String,Cliente> = {}	Se cambió exitosamente el valor del atributo clientes por la referencia pasada por parámetro	1

Método: void setPedidos(HashMap<Cliente,Pedido> pedidos)

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
pedidos	HashMap<Cliente,Pedido> != null	Si	1

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	pedidos	HashMap<Cliente,Pedido> = {}	Se cambió exitosamente el valor del atributo pedidos por la referencia pasada por parámetro	1

Método: void setUsuarioLogeado(Usuario usuario)

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
usuarioLogeado	Usuario !=null	Si	1

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	usuarioLogeado	Cliente("Esteban Gutierrez", "12345678", "estaban1200")	Se cambió exitosamente el valor del atributo usuarioLogeado por la referencia pasada por parámetro	1

Método: void setVehiculos(HashMap<String,Vehiculo> vehiculos)

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
vehiculos	HashMap<String,Vehiculo> != null	Si	1

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	vehiculos	HashMap<String,Vehiculo> = {}	Se cambio exitosamente el valor del atributo vehiculos por la referencia pasada por parámetro	1

Método: void setVehiculosDesocupados(ArrayList<Vehiculo> vehiculosDesocupados)

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
vehiculosDesocupados	ArrayList<Vehiculo> != null	Si	1

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas

1	vehiculos	ArrayList<Vehiculo> = {}	Se cambió exitosamente el valor del atributo vehiculosDesocupados por la referencia pasada por parámetro	1

Método: void setViajesIniciados(HashMap<Cliente,Viaje> viajesIniciados)

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
viajesIniciados	HashMap<Cliente, Viaje> != null	Si	1

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	viajesIniciados	HashMap<Cliente,Viaje> = {}	Se cambió exitosamente el valor del atributo viajesIniciados por la referencia pasada por parámetro	1

Método: setViajesTerminados(ArrayList<Viaje> viajesTerminados)

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
viajesTerminados	ArrayList<Viaje> != null	Si	1

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	viajesTerminados	ArrayList<Viaje> = {}	Se cambió exitosamente el valor del atributo viajesTerminados por la referencia pasada por parámetro	1

Método: void login(String usserName, String pass)
Precondiciones:

- usserName es distinto de null y tiene al menos un carácter
- pass es distinto de null y tiene al menos un carácter

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
usserName	usserName == null	No	-
usserName	usserName != null & no tiene caracteres	No	-
usserName	usserName != null & tiene al menos un caracter	Si	1
usserName	usserName que no tenga ningun cliente registrado	Si	2
pass	pass != null & tiene al menos un caracter	Si	3
pass	pass == null	No	-
pass	pass != null & no tiene caracteres	No	-

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias

				abarcadas
	usserName	"piojo_3000"		
1	pass	"12345678"	UsuarioNoExisteException	1

Método: Empresa getInstance()
Retorno: Instancia de clase Empresa, aplica Singleton

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	Una instancia de Empresa	1

Método: boolean isAdmin()
Retorno: Retorna true si el usuario logueado es el administrador en caso contraria devuelve false

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	FALSE	-

Método: Iterator<Chofer> iteratorChoferes()
Retorno: Iterator de Chofer del HashMap de choferes

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas

1	-	-	Iterator<Chofer> = {}	-

Método: Iterator<Cliente> iteratorClientes()
Retorno: Iterator de Cliente del HashMap de clientes

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	Iterator<Cliente> = {}	-

Método: Iterator<Pedido> iteratorPedidos()
Retorno: Iterator de Pedido del HashMap de pedidos

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	Iterator<Pedido> = {}	-

Método: Iterator<Vehiculo> iteratorVehiculos()
Retorno: Iterator de Vehiculo del HashMap de vehiculos

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	Iterator<Vehiculo> = {}	-

Método: Iterator<Viaje> iteratorViajesInciados()
Retorno: Iterator de Viaje del HashMap de viajesIniciados

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	Iterator<Viaje> = {}	-

Método: ArrayList<Viaje> iteratorViajesTerminados()
Retorno: Iterator de objetos de tipo Viaje del ArrayList de viajesTerminados

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	ArrayList<Viaje> = {}	-

Método: void logout()

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	usuarioLogeado = null	-

Resultados del Escenario N°1

Para la capa de negocio en este escenario, no se han encontrado errores al utilizar las pruebas unitarias automatizadas.

Escenario N°2

HashMap clientes	{Tesla = Cliente(“Test”,”1234”,”Nikola Tesla”), Mandarina = Cliente(“Mandarina”,”miau”,”Guchiuan”)}
HashMap choferes	{Fangio = ChoferPermanente(“987789”,”Fangio”,1900,0), Colapinto = ChoferTemporario(“23900189”,”Colapinto”)}
HashMap vehiculos	{Auto1 = Auto(“ZZ777AA”,4,true), Moto1 = Moto(“HH111HH”)}
HashMap pedidos	{}
HashMap viajesIniciados	{Viaje1 = Viaje(Pedido(Tesla, 4, false, true, 1000, “ZONA_PELIGROSA”),Fangio,Auto1)}
HashMap viajesTerminados	{}
HashMap choferesDisponibles	{ChoferTemporario(“23900189”,”Colapinto”}
HashMap vehiculosDisponibles	{Moto1 = Moto(“HH111HH”)}
usuarioLogueado	NULL

Método: void agregarCliente(String usuario, String pass, String nombreReal)
Precondiciones:

- usuario distinto de null y con al menos un carácter
- pass distinto de null y con al menos un carácter
- nombreReal distinto de null y con al menos un carácter

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
usuario	usuario != ""	Si	1
usuario	usuario != null	Si	2
usuario	usuario == null	No	-
usuario	usuario == ""	No	-
usuario	usuario repetido	Si	3

pass	pass != ""	Si	4
pass	pass != null	Si	5
pass	pass == null	No	-
pass	pass == ""	No	-
nombreReal	nombreReal != ""	Si	6
nombreReal	nombreReal != null	Si	7
nombreReal	nombreReal == null	No	-
nombreReal	nombreReal == ""	No	-

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
	usuario	"Tesla"		
	pass	"12345678"		
1	nombreReal	"Carlos Martinez"	UsuarioExistenteException	1,2,3,4,5,6,7

Método: boolean validarPedido(Pedido pedido)
Precondiciones: pedido es distinto de null
Retorno: Indica si un pedido tiene al menos un vehículo registrado con las características necesarias para satisfacer el pedido

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
pedido	pedido == null	No	-
pedido	pedido != null	Si	1
pedido	pedido con atributos que algun vehiculo existente satisfaga	Si	2

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas

1	pedido	Pedido(Fito, 5, true, true, 10, ZONA_ESTANDAR)	FALSE	1,2

Método: void crearViaje(Pedido pedido, Chofer chofer, Vehiculo vehiculo)
Precondiciones:

- pedido es distinto de null
- chofer es distinto de null
- vehiculo es distinto de null

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
pedido	pedido == null	No	-
pedido	pedido != null	Si	1
pedido	pedido que pertenece a la colección pedidos	Si	2
pedido	pedido con cliente que no tiene viaje iniciado	Si	3
chofer	chofer == null	No	-
chofer	chofer != null	Si	4
chofer	chofer que pertenece a la colección choferesDisponibles	Si	5
vehiculo	vehiculo == null	No	-
vehiculo	vehiculo != null	Si	6
vehiculo	vehiculo que satisface las caracteristicas del pedido	Si	7
vehiculo	vehiculo que no satisface las caracteristicas del pedido	Si	8
vehiculo	vehiculo que pertenece a la coleccion vehiculosDisponibles	Si	9
vehiculo	vehiculo que no pertenece a la coleccion vehiculosDisponibles	Si	10

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	pedido	Pedido(Mandarina, 4, false, true, 1000, ZONA_STANDARD)	VehiculoNoValidoException	1,2,3,4,5,6,8,9
	chofer	Fangio		
	vehiculo	Auto1		
2	pedido	Pedido (Mandarina, 3, true, true, 10, ZONA_ESTANDAR)	VehiculoNoDisponibleExcepti on	1,2,3,4,5,7,10
	chofer	Fangio		
	vehiculo	Auto1		
3	pedido	Pedido (Mandarina, 1, true, true, 10, ZONA_ESTANDAR)	ChoferNoDisponibleException	1,2,3,4,5,6,7,8
	chofer	Fangio		
	vehiculo	Auto1		

Método: void login(String usserName, String pass)
Precondiciones:

- usserName es distinto de null y tiene al menos un carácter
- pass es distinto de null y tiene al menos un carácter

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
usserName	usserName == null	No	-
usserName	usserName != null	Si	1
usserName	usserName que tenga algun cliente registrado	Si	2
usserName	usserName que no tenga ningun cliente	Si	3

	registrado		
pass	pass == null	No	-
pass	pass != null	Si	4
pass	password correcta	Si	5
pass	password incorrecta	Si	6

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	usserName	"Tesla"	referencia al cliente Tesla	1,2,4,5
	pass	"1234"		
2	usserName	"Tesla"	PasswordErroneaException	1,2,4,6
	pass	"999"		

Método: void agregarPedido(Pedido pedido)
Precondiciones: pedido es distinto de null

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
pedido	pedido == null	No	-
pedido	pedido != null	Si	1
pedido	pedido con atributos válidos	Si	2
pedido	pedido con atributos que ningun vehiculo satisfaga	Si	3

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	pedido	Pedido(Mandarina, 1, false, false, 10,	Se agrega un pedido a la	1,2

		ZONA_ESTANDAR)	coleccion de pedidos	
2	pedido	Pedido(Mandarina, 9, true, true, 10, ZONA_ESTANDAR)	SinVehiculoParaPedidoException	1,3

Método: HashMap<String,Chofer> getChoferes()
Retorno: HashMap[String,Chofer] donde el dni del chofer es la clave de cada chofer, contiene todos los choferes registrados de la empresa

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	ArrayList<Chofer> = {Fangio,Colapinto}	-

Método: ArrayList<Chofer> getChoferesDesocupados()
Retorno: ArrayList de Chofer representa los choferes que no están realizando un viaje

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	ArrayList<Chofer> = {Colapinto}	-

Método: Iterator<Chofer> iteratorChoferes()
Retorno: Iterator de Chofer del HashMap de choferes

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	Iterator<Chofer> = {Fangio,Colapinto}	-

Método: Iterator<Cliente> iteratorClientes()
Retorno: Iterator de Cliente del HashMap de clientes

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	Iterator<Cliente> = {Tesla,Mandarina}	-

Método: ArrayList<Vehiculo> getVehiculosDesocupados()
Retorno: ArrayList[Vehiculo] que representa los vehiculos que no están realizando un viaje

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	ArrayList<Vehiculo> = { Moto1 }	-

Resultados del Escenario N°2

Para la capa de negocio en este escenario, no se han encontrado errores al utilizar las pruebas unitarias automatizadas

Escenario N°3

HashMap clientes	{ Cliente ("Tesla", "1234","Nikola Tesla"),Cliente("Mandarina", "miau", "Guchiuan") }
HashMap choferes	{ChoferPermanente("987789", "Fangio", 1900, 0) }
HashMap vehiculos	{Combi1 = Combi ("AC789BB" , 10, false),

	Auto1 = Auto("ZZ777AA" , 4, true), Moto1 = Moto("HH111HH") }
HashMap pedidos	{ Pedido1 = Pedido (Tesla, 4, true, false, 50, ZONA_PELIGROSA) }
HashMap viajesIniciados	{Viaje1 = Viaje(Pedido(Tesla, 4, false, true, 1000, “ZONA_PELIGROSA”),Fangio,Auto1)}
HashMap viajesTerminados	{}
HashMap choferesDisponibles	{}
HashMap vehiculosDisponibles	{Combi1 = Combi ("AC789BB" , 10, false), Moto1 = Moto("HH111HH")}
usuarioLogueado	NULL

Método: void agregarChofer(Chofer chofer)
Precondiciones: el parámetro chofer es distinto de null
Throws: ChoferRepetidoException - en caso de que el dni del chofer pasado por parametro oincida con el dni de un chofer previamente registrado

Tabla de particiones

Dato de entrada	Descripcion de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
chofer	chofer == null	No	-
chofer	chofer != null	Si	1
chofer	chofer no esta en la colección	Si	2
chofer	chofer con dni repetido	Si	3

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	chofer	ChoferTemporario("45620890", "Marcos")	Se añadio correctamente a la coleccion de choferes	1,2
2	chofer	ChoferPermanente(	ChoferRepetido	1,3

		"987789", "Fangio", 1900, 0)	Exception	

Método: void agregarVehiculor(Vehiculo vehiculo)
Precondiciones: el parámetro vehiculo es distinto de null
Throws: VehiculoRepetidoException - si la patente del parametro vehiculo coincide con la de algun vehiculo previamente registrado

Tabla de particiones

Dato de entrada	Descripcion de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
vehiculo	vehiculo == null	No	-
vehiculo	vehiculo != null	Si	1
vehiculo	vehiculo no esta en la coleccion	Si	2
vehiculo	vehiculo con patente repetida	Si	3

Bateria de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	vehiculo	Auto("JJ888JJ" , 3, false)	Se añadio correctamente a la colección de clientes	1,2
2	vehiculo	Moto("HH111HH")	VehiculoRepetidoException	1,3

Método: void agregarPedido(Pedido pedido)
Precondiciones: el parametro pedido es diferente de null
Throws:

- SinVehiculoParaPedidoException - Se lanza en caso de que la empresa no tenga registrado ningun vehículo con las características necesarias para satisfacer el pedido.
- ClienteConViajePendienteException - Se lanza si el Cliente tiene un viaje iniciado
- ClienteConPedidoPendienteException - Se lanza si el Cliente tiene un pedido iniciado
- ClienteNoExisteException - Se lanza si el atributo cliente del parametro pedido, no es un cliente registrado en la empresa

Tabla de particiones

Dato de entrada	Descripcion de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
pedido	pedido == null	No	-
pedido	pedido != null	Si	1
pedido	pedido con cliente con pedido hecho	Si	2

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	pedido	Pedido(Tesla, 3, true, true, 10,ZONA_SIN_ASFALTAR)	ClienteConPedidoPendienteException	1,4

Metodo: booleanvalidarPedido(Pedido pedido)
Precondiciones: el parámetro pedido es diferente de null.
Retorno: true si existe al menos un vehiculo que satisfaga el pedido, false en caso contrario

Tabla de particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
pedido	pedido == null	No	-
pedido	pedido != null	Si	1
pedido	pedido con atributos que algun vehiculo	Si	2

	existente satisfaga		

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	pedido	Pedido(Fito, 5, true, true, 10, ZONA_ESTANDAR)	TRUE	1,2

Metodo: ArrayList<Vehiculo> vehiculosOrdenadosPorPedido(Pedido pedido)
Precondiciones: el parametro pedido es diferente de null
Retorno: ArrayList de objetos de tipo Vehiculo 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

Dato de entrada	Descripcion de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
cliente de pedido	cliente == null	No	-
cliente de pedido	cliente != null	Si	1
cliente de pedido	cliente que tiene un pedido	Si	2

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	Pedido 1	MandarinaPedido (Tesla, 4, true, false, 50, ZONA_PELIGROSA)	ArrayList<Vehiculo> = { Auto1 }	1,2

Metodo: Pedido getPedidoDeCliente(Cliente cliente)
Precondiciones: el parametro cliente es distinto de null
Retorno: el Pedido realizado por el cliente. Si el cliente no tiene ningun pedido pendiente se retorna null

Tabla de particiones

Dato de entrada	Descripcion de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
cliente	cliente == null	No	-
cliente	cliente != null	Si	1
cliente	cliente que tiene un pedido	Si	2

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	cliente	Tesla	Pedido1	1,2

Metodo: HashMap<String,Vehiculo> getVehiculos()
Retorno: la totalidad de los vehiculos registrados

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1			ArrayList<Vehiculo> = { Combi1, Auto1,Moto1 }	

Metodo: HashMap<Cliente,Pedido> getPedidos()
Retorno: la totalidad de los pedidos pendientes

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1			ArrayList<Pedido> = { Pedido1 }	

Metodo: Iterator<Vehiculo> iteratorVehiculos()
Retorno: la totalidad de los vehículos registrados

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1			Iterator<Vehiculo> = { Combi1, Auto1, Moto1 }	

Metodo: Iterator<Pedido> iteratorPedidos()
Retorno: la totalidad de los pedidos pendientes

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1			Iteratort<Pedido> = { Pedido1 }	

Resultados del Escenario N°3

Nombre de la prueba	Error
testAgregarChofer_Clase2	Deberia lanzar excepcion ChoferRepetidoException

Escenario N°4

HashMap clientes	{Cliente ("Tesla", "1234","Nikola Tesla") Cliente("Mandarina", "miau", "Guchiuan") Cliente("Eloncito", "$$$", "Elon Musk") Cliente("Fito", "teknikolor","Fito Paez")}
HashMap choferes	{ChoferPermanente("987789", "Fangio", 1900, 0) ChoferTemporario ("23900189" , "Colapinto") ChoferPermanente("776677", "Toretto", 2000, 4) ChoferTemporario ("93456712" , "Mercedes")}

HashMap vehiculos	{Combi1 = Combi ("AC789BB" , 10, false), Auto1 = Auto("ZZ777AA" , 4, true), Moto1 = Moto("HH111HH") Combi2 = Combi("KL898LK" , 8, true)}
HashMap pedidos	{ Pedido1 = Pedido (Tesla, 4, true, false, 50, ZONA_PELIGROSA) }

HashMap viajesIniciados	{
	Pedido1 = Pedido(Eloncito 1, false, true, 1000, ZONA_PELIGROSA) -> Viaje1 = Viaje(Pedido1, Toretto, Auto1)
	
	Pedido2 = Pedido(Mandarina, 1, false, false, 100, ZONA_PELIGROSA) -> Viaje2 = Viaje(Pedido2, Mercedes, Moto1)
	
	Pedido3 = Pedido(Eloncito, 9, false, true, 800, ZONA_ESTANDAR) -> Viaje3 = Viaje(Pedido3, Colapinto, Combi1)
	}

HashMap viajesTerminados	{
	Pedido4 = Pedido(Mandarina, 1, false, false, 290, ZONA_SIN_ASFALTAR) -> Viaje4 = Viaje(Pedido4, Fangio, Moto1)
	
	Pedido5 = Pedido(Tesla, 3, false, false, 90, ZONA_ESTANDAR) -> Viaje5 = Viaje(Pedido5, Colapinto, Auto1)
	
	Pedido6 = Pedido(Mandarina, 8, false, false, 90, ZONA_SIN_ASFALTAR) -> Viaje6 = Viaje(Pedido6, Colapinto, Combi2)
	}into, Combi1)
	}

HashMap choferesDisponibles	{}
HashMap vehiculosDisponibles	{}
usuarioLogueado	Cliente ("Tesla", "1234","Nikola Tesla")

Método: void agregarPedido(Pedido pedido)
Precondiciones: el parametro pedido es diferente de null
Throws:

- SinVehiculoParaPedidoException - Se lanza en caso de que la empresa no tenga registrado ningun vehículo con las características necesarias para satisfacer el pedido.
- ClienteConViajePendienteException - Se lanza si el Cliente tiene un viaje iniciado
- ClienteConPedidoPendienteException - Se lanza si el Cliente tiene un pedido iniciado

- ClienteNoExisteException - Se lanza si el atributo cliente del parametro pedido, no es un cliente registrado en la empresa

Tabla de particiones

Dato de entrada	Descripcion de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
pedido	pedido == null	No	-
pedido	pedido != null	Si	1
pedido	pedido con atributos invalidos	No	-
pedido	pedido con atributos validos	Si	2
pedido	pedido con cliente con viaje iniciado	Si	3

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	pedido	Pedido(Eloncito, 12, true, true, 10,ZONA_PELIGROSA)	ClienteConViajePendienteException	1,2,3

Metodo: void crearViaje(Pedido pedido, Chofer chofer, Vehiculo vehiculo)
Precondiciones: los parametros pedido, chofer y vehiculo son distintos de null
Throws:

- PedidoInexistenteException - Se lanza si el pedido pasado como parametro no pertence al HashMap de pedidos
- ChoferNoDisponibleException - Se lanza si el chofer no pertenece al ArrayList de choferesDisponibles
- VehiculoNoDisponibleException - Se lanza si el vehiculo no pertenece al ArrayList de vehiculosDisponibles
- VehiculoNoValidoException - Se lanza si el vehiculo no puede satisfacer el pedido
- ClienteConViajePendienteException - Se lanza si el Cliente está realizando un Viaje

Tabla de particiones

Dato de entrada	Descripcion de la clase de equivalencia	Aplica	Identificador de clase de

			equivalencia
pedido	pedido == null	No	-
pedido	pedido != null	Si	1
pedido	pedido que no pertenece a la coleccion pedidos	Si	2
pedido	pedido con cliente que tiene viaje iniciado	Si	3
chofer	chofer == null	No	-
chofer	chofer != null	Si	4
chofer	chofer que pertenece a la coleccion choferesDisponibles	Si	5
vehiculo	vehiculo == null	No	-
vehiculo	vehiculo != null	Si	6
vehiculo	vehiculo que satisface las caracteristicas del pedido	Si	7
vehiculo	vehiculo que pertenece a la coleccion vehiculosDisponibles	Si	8

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
3	pedido	pedido = Pedido (Eloncito, 1, true, true, 10, ZONA_ESTANDAR)	ClienteConViajePendienteException	1,2,3,4,5,6,7,8
	chofer	Fangio		
	vehiculo	Combi2		

Metodo: double calificacionDeChofer(Chofer chofer)
Precondiciones: el parametro chofer es distinto de null
Retorno: el promedio de las calificaciones de los viajes realizados por el chofer en cuestion
Throws: SinViajesException - se Lanza si el chofer no tiene ningún viaje realizado.

Tabla de particiones

Dato de entrada	Descripcion de la clase de equivalencia	Aplica	Identificador de clase de

			equivalencia
chofer	chofer == null	No	-
chofer	chofer != null	Si	1
chofer	chofer con viajes realizados	Si	2
chofer	chofer sin viajes realizados	Si	3

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	chofer	Colapinto	4	1,2
2	chofer	ChoferTemporario ("23900189" , "Lucas")	SinViajesException	1,3

Método: void pagarYFinalizarViaje(int calificacion)
Precondiciones: hay un usuario de tipo Cliente logeado en la Empresa
calificacion está comprendido entre 0 y 5 inclusive
Throws: ClienteSinViajePendienteException - Se lanza si el Cliente no está realizando un viaje.

Tabla de particiones

Dato de entrada	Descripcion de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
calificacion	calificacion < 0	No	-
calificacion	calificacion > 5	No	-
calificacion	calificacion > 0 && <=5	Si	1

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	logeas como cliente a Eloncito	5	Califica y termina el viaje	1

			de Eloncito	
2	logeas como cliente a Fito	4	ClienteSinViajePendienteException	1

Método: ArrayList<Viaje> getHistorialViajeChofer(Chofer chofer)
Precondiciones: el parametro chofer es distinto de null
Retorno: ArrayList de objetos de tipo Viaje correspondiente a los viajes realizados por el chofer en cuestion

Tabla de particiones

Dato de entrada	Descripcion de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
chofer	chofer == null	No	-
chofer	chofer != null	Si	1
chofer	chofer con viajes realizados	Si	2
chofer	chofer sin viajes realizados	Si	3

Bateria de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	chofer	Colapinto	ArrayList<Viaje> = { Viaje5,Viaje6 }	1,2
2	chofer	Toretto	ArrayList<Viaje> == NULL	1,3

Método: ArrayList<Viaje> getHistorialViajeCliente(Cliente cliente)
Precondiciones: el parámetro cliente es distinto de null
Retorno: ArrayList de objetos de tipo Viaje correspondiente a los viajes realizados por el cliente en cuestion

Tabla de particiones

Dato de entrada	Descripcion de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
cliente	cliente == null	No	-
cliente	cliente != null	Si	1
cliente	cliente con viajes realizados	Si	2
cliente	cliente sin viajes realizados	Si	3

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	cliente	Mandarina	ArrayList<Viaje> = { Viaje4,Viaje6 }	1,2
2	cliente	Fito	ArrayList<Viaje> == NULL	1,3

Método:Viaje getViajeDeCliente(Cliente cliente)
Precondiciones: el parámetro cliente es distinto de null
Retorno: el Viaje no terminado que está realizando por el cliente. Si el cliente no esta realizando un viaje se retorna null

Tabla de particiones

Dato de entrada	Descripcion de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
cliente	cliente == null	No	-
cliente	cliente != null	Si	1
cliente	cliente que esta realizando un viaje	Si	2
cliente	cliente que no esta realizando un viaje	Si	3

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada
1	cliente	Eloncito	Viaje1

2	cliente	Fito	NULL

Método: double getTotalSalarios()
Retorno: double que representa la suma de los salarios de los choferes registrados.

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	(hay q calcular cuanto daria con los choferes q hay y eso)	-

Método: HashMap<Cliente,Viaje> getViajesIniciados()
Retorno: los viajes iniciados pero no terminados.

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	ArrayList<Viaje> = { VIaje1,Viaje2,Viaje3}	-

Método: HashMap<Cliente,Viaje>getViajesTerminados()
Retorno: los viajes ya finalizados. Ya han sido pagados y calificados.

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	ArrayList<Viaje> = { VIaje4,Viaje5,Viaje6}	-

Método: Iterator<Viaje> iteratorViajesIniciados()
Retorno: los viajes iniciados pero no terminados.

Bateria de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	Iterator<Viaje> = {	-

			VIaje1,Viaje2,Viaje3}	

Método: void logout()

Bateria de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	usuarioLogeado = null	-

Resultados del Escenario N°4

Nombre de la prueba	Error
testAgregarPedido_Clase1	Deberia saltar la excepcion de cliente con viaje pendiente
testCalificacionDeChofer_Clase1	Calificacion promedio mal calculada, esperaba 4 pero devolvio 2
testCalificacionDeChofer_Clase2	Chofer sin viajes, entonces deberia salir por la excepción de chofer sin viajes
testGetTotalSalarios_Clase1	No se calcula bien el total de salarios

Test de controlador/integración

El test de integración tiene como finalidad probar la interacción entre los distintos componentes del sistema. En el caso del software bajo estudio, al presentar una arquitectura de tres capas (Vista + Negocio + Modelo), existe una clase Controlador, que es aquella que cumple el rol de interfaz entre la capa de Vista y la de Negocio-Modelo. Por este motivo, se llevó adelante un testeo de dicha clase controladora, ya que nos permite comprobar el correcto funcionamiento en conjunto de los componentes.

Para llevar adelante el testeo de los métodos pertenecientes a la clase Controlador, se definieron los distintos casos de usos correspondientes, luego se continuó con el desarrollo de los diagramas de secuencia, de los cuales se desprendieron los casos de prueba a testear. De forma similar al testeo de la capa de datos, para definir las particiones y los resultados esperados se utilizó de apoyo la información especificada en la documentación de los métodos a testear.

A continuación, se presentan los casos de uso y diagramas de secuencia correspondientes a algunos métodos de la clase Controlador a testear. Se tuvieron en consideración aquellos casos de uso y diagramas de secuencia que reflejan el

comportamiento principal del sistema, excluyendo aquellos, como por ejemplo nuevoChofer() y nuevoVehiculo() , cuyos métodos y objetivos son similares.

Para la elaboración de casos de uso se emplearon los diagramas de actividad ya que en su forma de grafo permiten una fácil visualización de los distintos flujos que se pueden seguir y es posible seleccionar a partir de ellos los diferentes casos de prueba.

Casos de uso

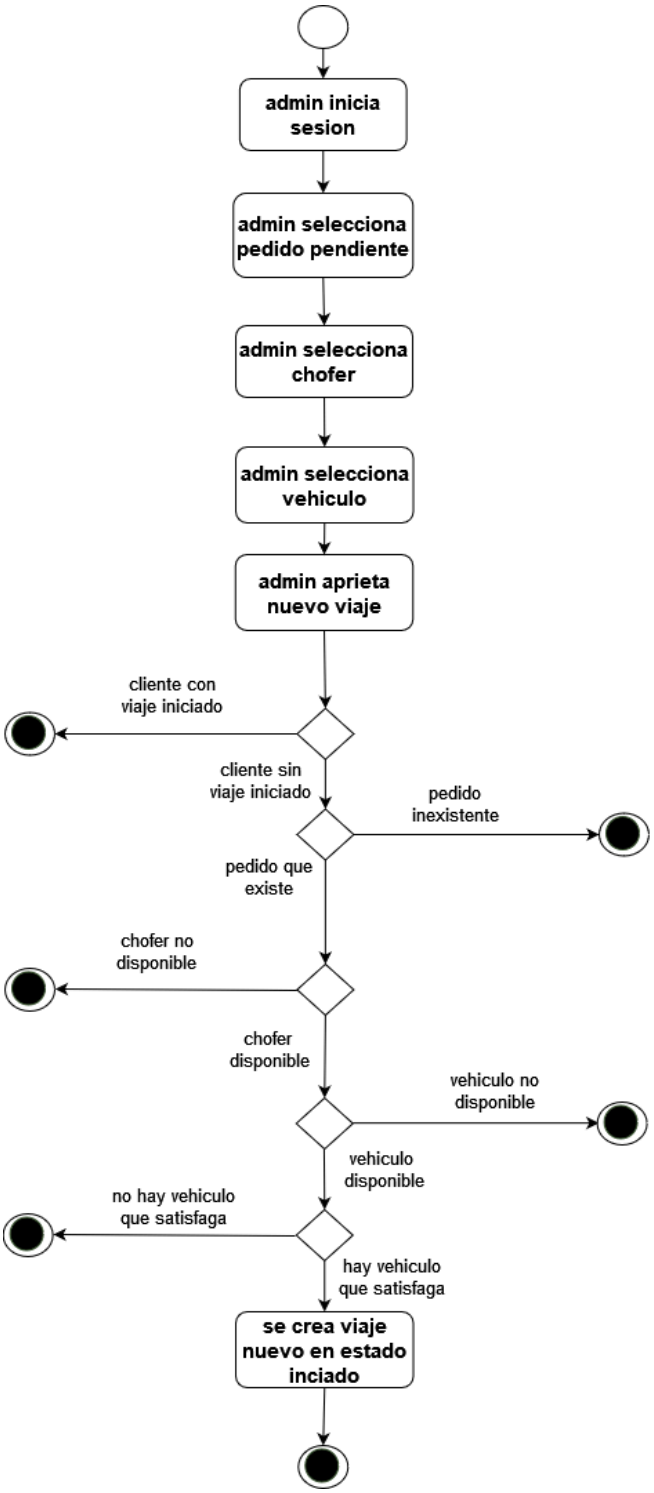

Imagen 1 - Diagrama de actividad - Creacion de viaje

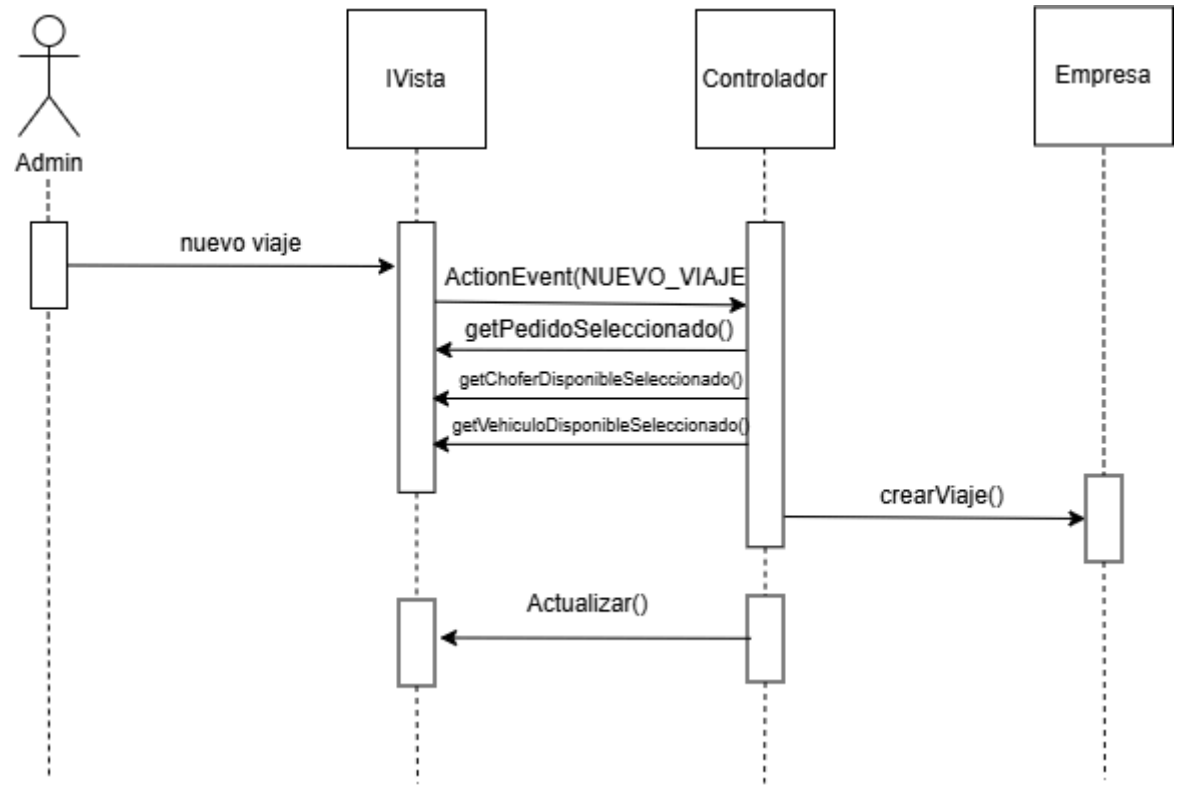

Imagen 2 - Diagrama de secuencia - Creación de viaje exitosa

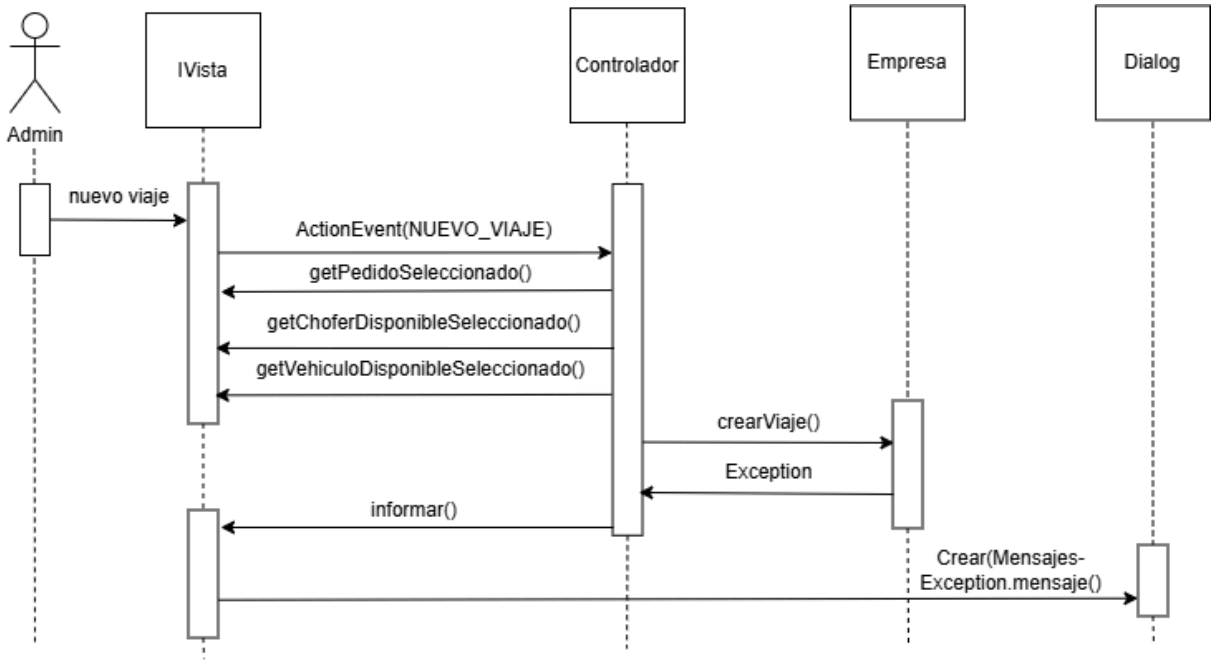

Imagen 3 - Diagrama de secuencia - Creación de viaje fallida

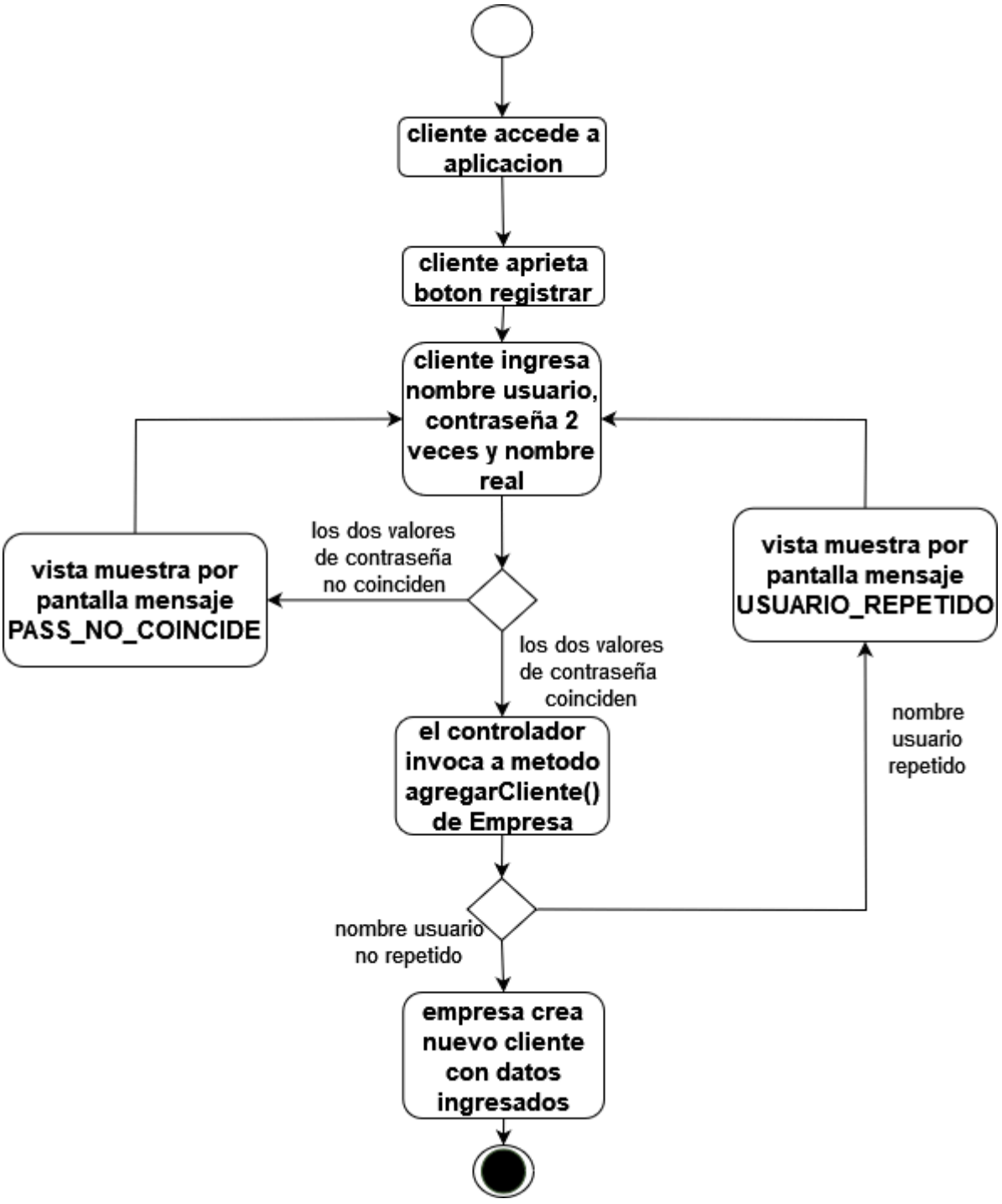

Imagen 4 - Diagrama de actividad - Registro de cliente

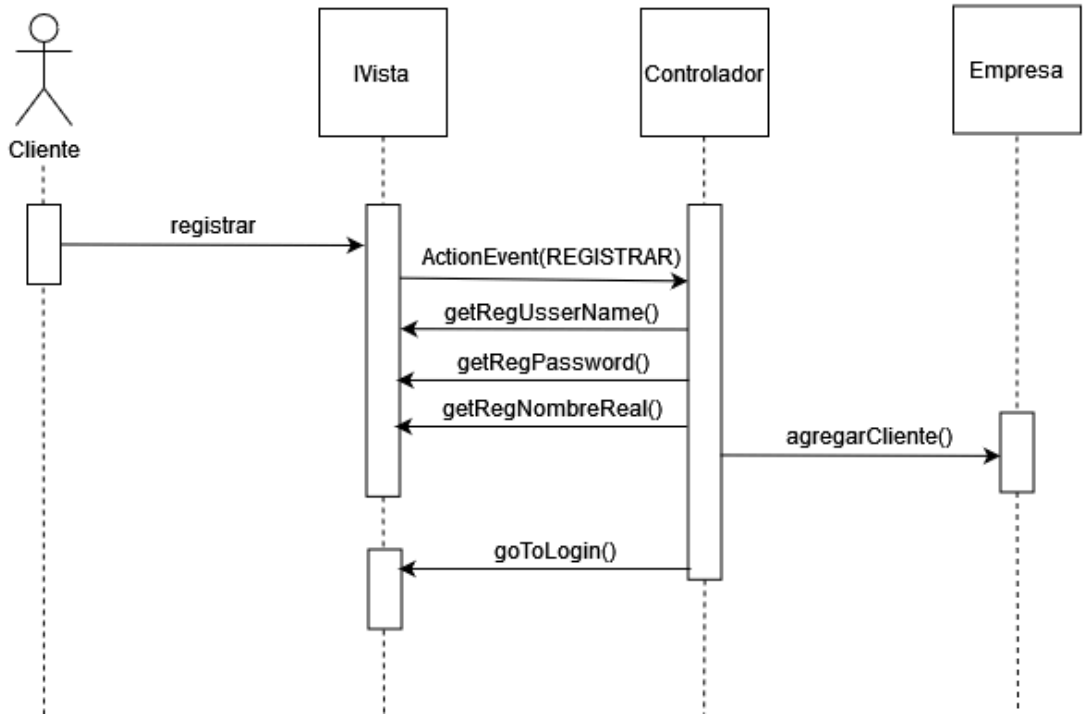

Imagen 5 - Diagrama de secuencia - Registro de cliente exitoso

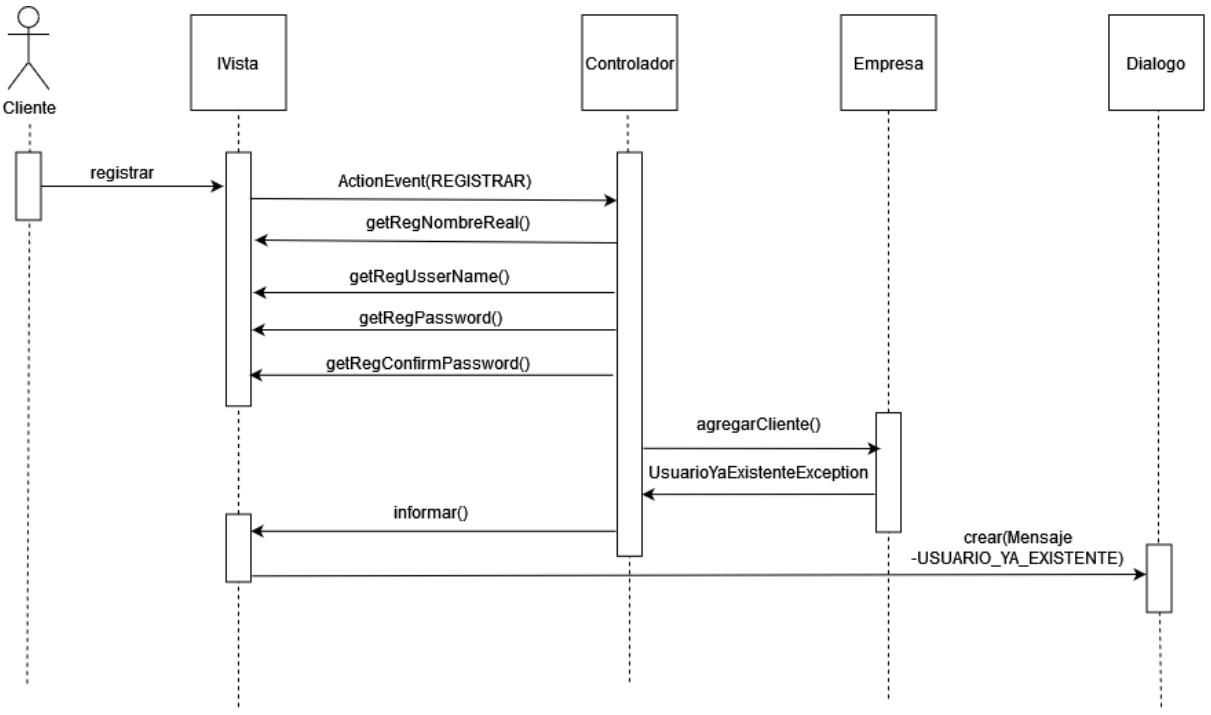

Imagen 6 - Diagrama de secuencia - Registro de cliente usuario repetido

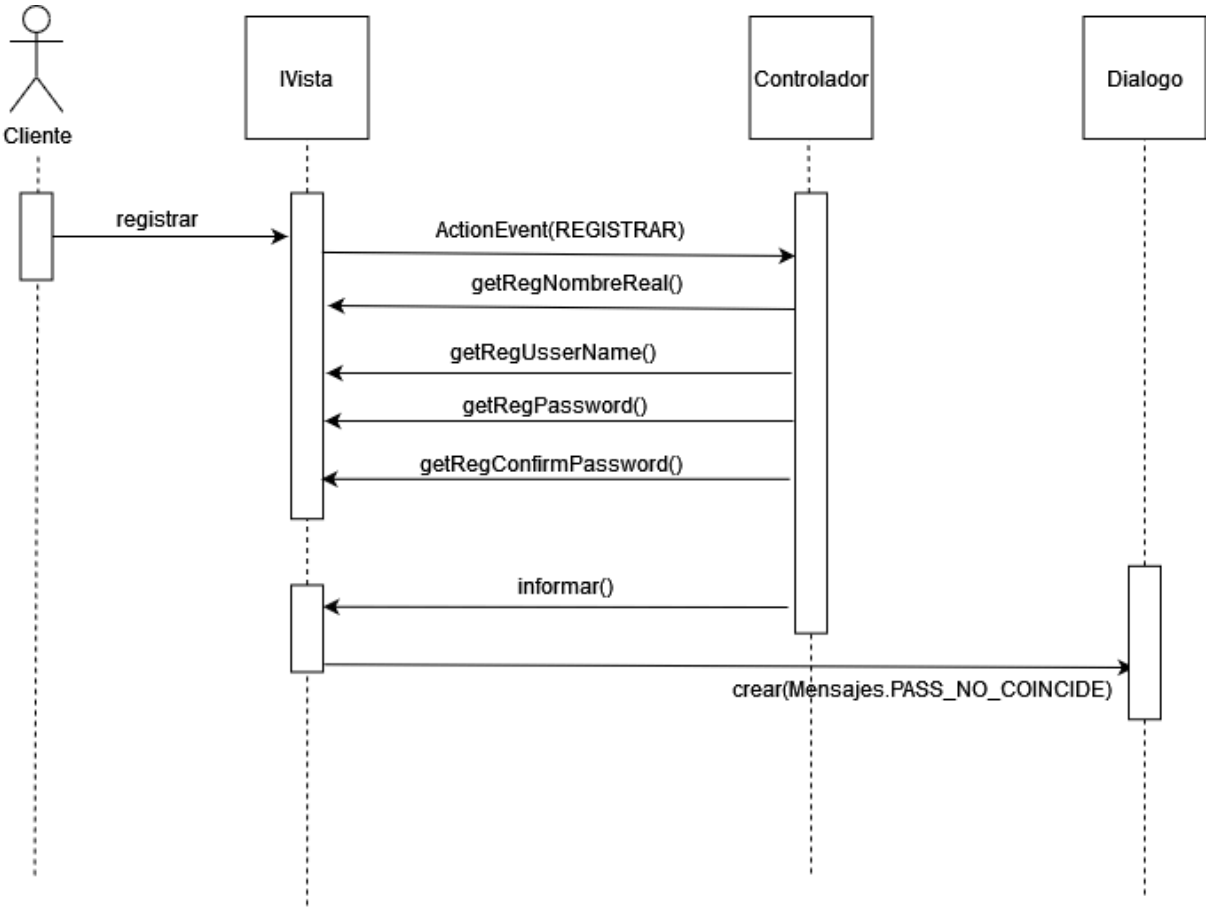

Imagen 7 - Diagrama de secuencia - Registro de cliente contraseñas erroneas

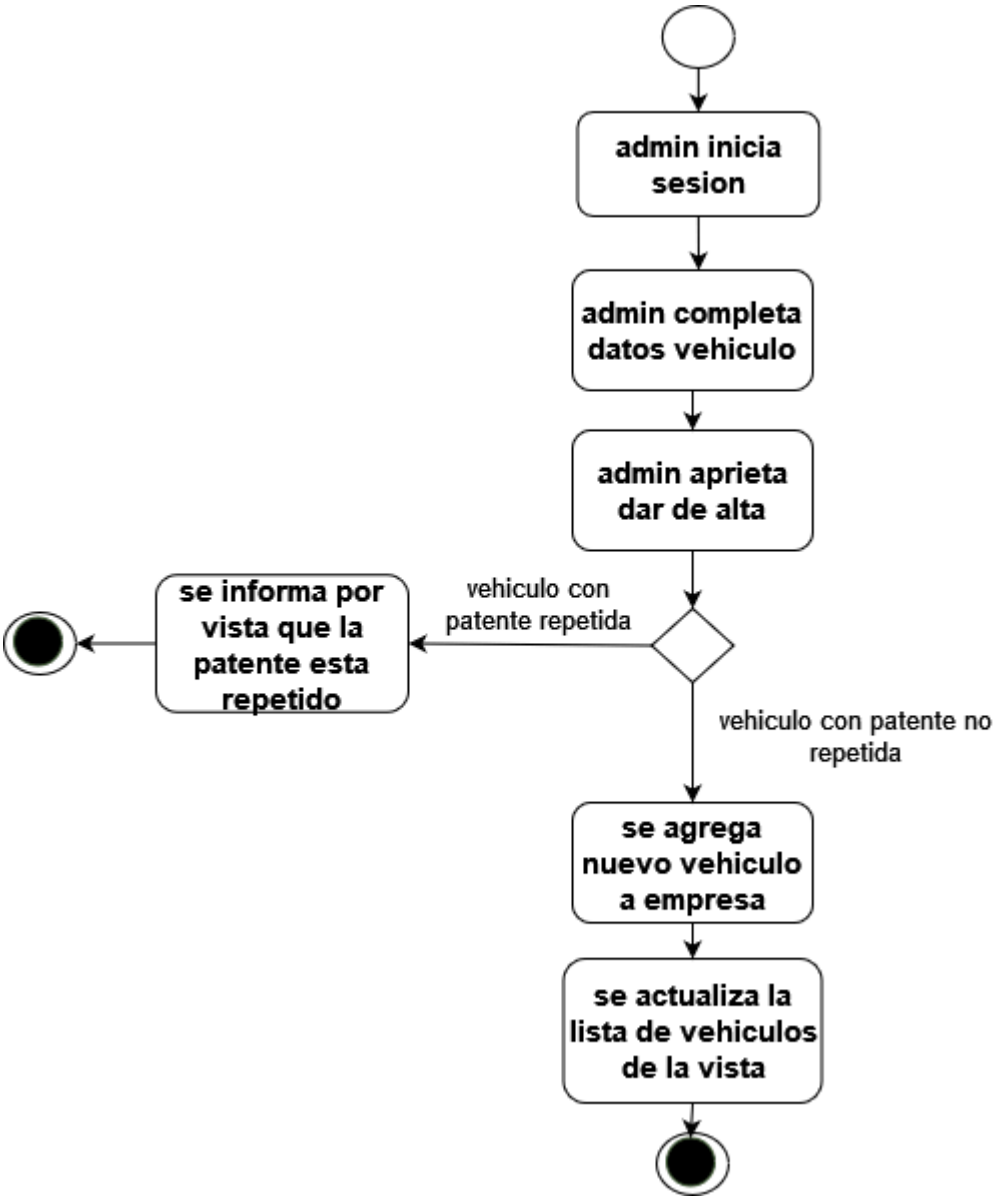

Imagen 8 - Diagrama de actividad - Alta de vehículo

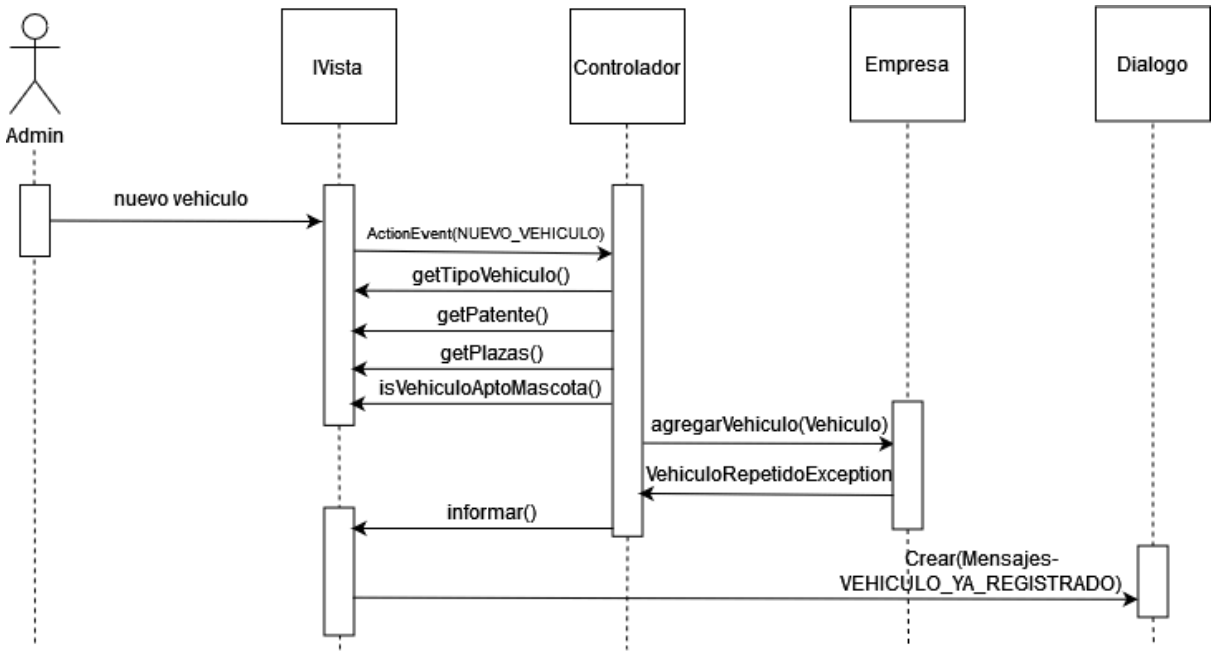

Imagen 9 - Diagrama de secuencia- Alta de vehículo fallido

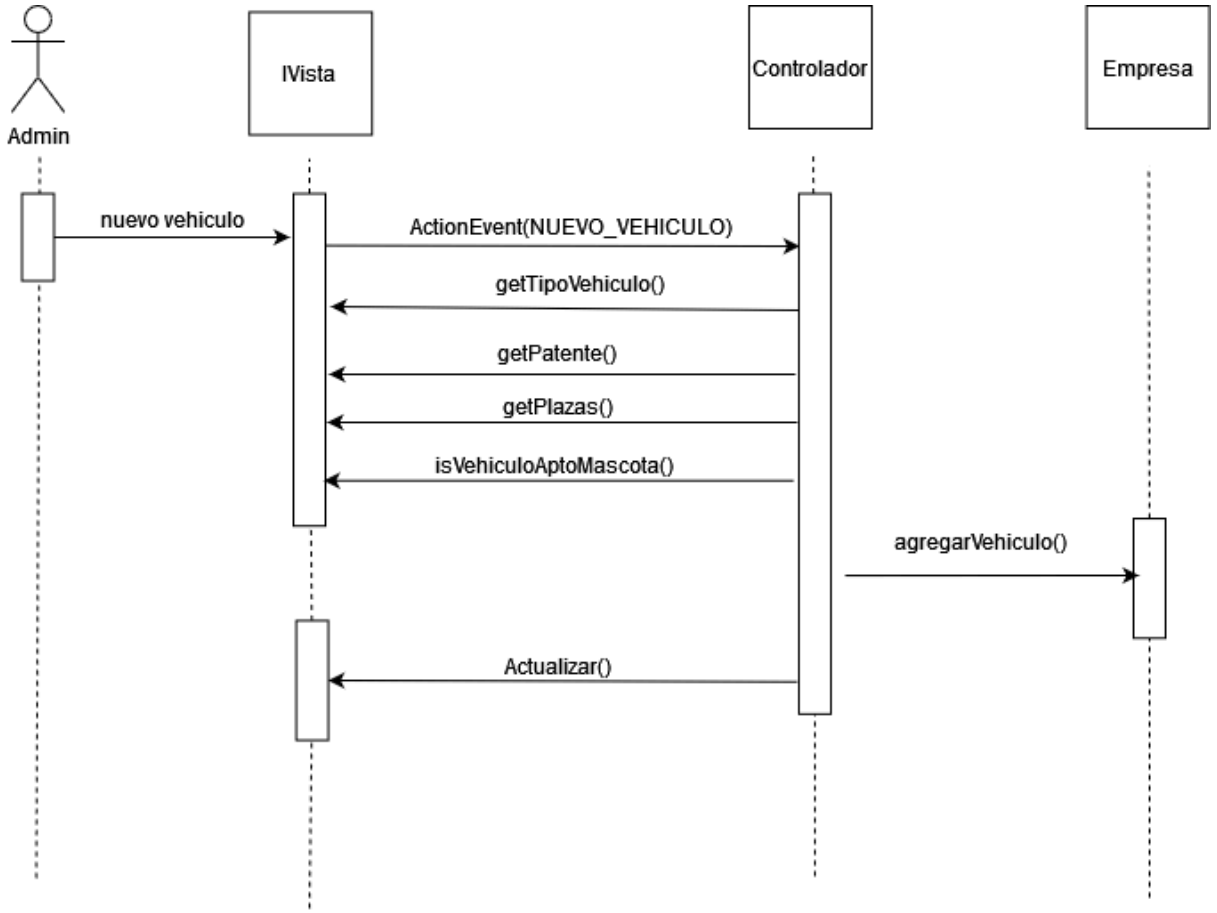

Imagen 10 - Diagrama de secuencia- Alta de vehículo exitosa

A continuación se define el escenario con el cual se llevaron adelante todos los casos de pruebas definidos.

Escenario

HashMap clientes	{Mandarina = Cliente ("Mandarina", "miau","Mandarina"), Eloncito = Cliente("Eloncito", "password", "Eloncito"), cr7 = Cliente("cr7","misterChampions","Ronaldo")}
HashMap choferes	{Toretto = ChoferTemporario("44555666","Toretto"), Colapinto = ChoferTemporario ("33888777", "Colapinto")}
HashMap vehiculos	{Combi1 = Combi ("AA777AA" , 9, true), Auto1 = Auto("KK999KK" , 4, true)}
HashMap pedidos	{Pedido1 = Pedido (Mandarina, 9, true, false, 10, ZONA_STANDARD), Pedido2 = Pedido(cr7,4,false,false,5,ZONA_SIN_ASFALTAR)}
HashMap viajesIniciados	{Viaje = Viaje(Pedido1,Toretto,Combi1)}
HashMap viajesTerminados	{}
HashMap choferesDisponibles	{Colapinto = ChoferTemporario ("33888777", "Colapinto")}
HashMap vehiculosDisponibles	{Auto1 = Auto("KK999KK" , 4, true)}
usuarioLogueado	Cliente ("Mandarina", "miau","Mandarina")

En algunos casos en las tablas de particiones y en las baterías de pruebas se utilizaron abreviaturas con respecto a las objetos que referencian, con el objetivo de no poner reiteradamente el estado del objeto para definirlo, por ejemplo, el Pedido (Mandarina, 9, true, false, 10, ZONA_STANDARD) será mencionado posteriormente como Pedido1, la definición de cada una de estas abreviaturas se realizó en el escenario de la siguiente manera: abreviatura = Objeto().

Para lograr testear el controlador, se creó un objeto VistaMock que implementa IVista, para que el controlador tenga a quien pedir los datos que sus métodos utilizará

A continuación se presentan las tablas de particiones y batería de pruebas correspondiente a cada método de la clase Controlador y desarrolladas en el escenario descrito anteriormente.

Método: void ActionPerformed**(**ActionEvent e**)**

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
ActionEvent	ActionEvent.command = "LOGIN"	Si	1
ActionEvent	ActionEvent.command = "REG_BUTTON_REGISTRAR"	Si	2
ActionEvent	ActionEvent.command = "CALIFICAR_PAGAR"	Si	3
ActionEvent	ActionEvent.command = "NUEVO_PEDIDO"	Si	4
ActionEvent	ActionEvent.command = "NUEVO_VIAJE"	Si	5
ActionEvent	ActionEvent.command = "NUEVO_CHOFER"	Si	6
ActionEvent	ActionEvent.command = "NUEVO_VEHICULO"	Si	7
ActionEvent	ActionEvent.command = "CERRAR_SESION_CLIENTE"	Si	8
ActionEvent	ActionEvent.command = "CERRAR_SESION_ADMIN"	Si	9

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	ActionEvent	ActionEvent(JButton(), ActionEvent.ACTION_PERFORMED,"LOGIN")	Se logueó un cliente con el siguiente estado = Cliente("Eloncito", "password", "Eloncito")	1
2	ActionEvent	ActionEvent(JButton(), ActionEvent.ACTION_PERFORMED,"REG_BUTTON_REGISTRAR")	Se registró un cliente con el siguiente estado = Cliente("pepito45", "password", "Martinez")	2
3	ActionEvent	ActionEvent(JButton(), ActionEvent.ACTION_PERFORMED,"CALIFICAR_PAGAR")	Se finalizó el único viaje iniciado con una calificación de 4	3
4	ActionEvent	ActionEvent(JButton(), ActionEvent.ACTION_PERFORMED,"NUEVO_PE	El cliente Eloncito realizó el siguiente pedido = Pedido (Eloncito,4,true,true,100,Z	4

		DIDO")	ONA_PELIGROSA	
5	ActionEvent	ActionEvent(JButton(), ActionEvent.ACTION_PERFORMED,"NUEVO_VIAJE")	Se creó el siguiente viaje para el cliente Eloncito Pedido = (Eloncito ,4,true,true, 100, ZONA_PELIGROSA) Viaje(Pedido,Colapinto,Auto1)	5
6	ActionEvent	ActionEvent(JButton(), ActionEvent.ACTION_PERFORMED,"NUEVO_CHOFER")	Se registró un Chofer con el siguiente estado = ChoferPermanente("11888555","Fangio",2024,2)	6
7	ActionEvent	ActionEvent(JButton(), ActionEvent.ACTION_PERFORMED,"NUEVO_VEHICULO")	Se registro un Vehiculo con el siguiente estado = Auto("ZZ444ZZ",4,true)	7
8	ActionEvent	ActionEvent(JButton(), ActionEvent.ACTION_PERFORMED,"CERRAR_SESION_CLIENTE")	Usuario Logeado = null	8
9	ActionEvent	ActionEvent(JButton(), ActionEvent.ACTION_PERFORMED,"CERRAR_SESION_ADMIN")	Usuario Logeado = null	9

Método: void calificarPagar()

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
calificacion	Usuario Logeado tiene un viaje iniciado	Si	1
calificacion	Usuario Logeado no tiene un viaje iniciado	Si	2

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	calificacion	4	Mandarina califico con 4 y pagó su viaje iniciado	1

2	calificacion	4	Eloncito no tiene viajes pendientes	2

Método: void escribir()

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
fileName	es posible la persistencia	Si	1
fileName	no es posible la persistencia	Si	2

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	fileName	"empresa.bin"	Se logró persistir correctamente	1
2	fileName	"empresa.bin"	Error al escribir en el archivo	2

Método: String getFileName()

Retorno: Atributo fileName

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
fileName	!=null	Si	1

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	fileName	"empresa.bin"	"empresa.bin"	1

Método: IPersistencia getPersistencia()

Retorno: Atributo persistencia

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia

persistencia	!=null	Si	1

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	persistencia	PersistenciaBIN()	misma referencia que en controlador.persistencia	1

Método: IVista getVista()

Retorno: Atributo vista

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
vista	!=null	Si	1

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	vista	IVista()	misma referencia que en controlador.vista	1

Método: void leer()

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
fileName	despersistencia posible	Si	1
fileName	despersistencia no posible	Si	2

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	fileName	"empresa.bin"	se despersistio	1

			correctamente	
2	fileName	"empresa.bin"	error al leer el archivo	2

Método: void login()

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
usserName	==null	No	-
usserName	!=null	Si	1
usserName	existe en el sistema	Si	2
usserName	no existe en el sistema	Si	3
password	==null	No	-
password	!=null	SI	4
password	coincide con la password del usuario a logear	Si	5
password	no coincide con la password del usuario a logear	Si	6

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	usserName	"Eloncito"	Se logeo correctamente el cliente Eloncito	1,2,4,5
	password	"password"		
2	usserName	"Eloncito"	Password incorrecto	1,2,4,6
	password	"$$$"		
3	usserName	"Messi"	Usuario Inexistente	1,3,4
	password	"password"		

Método: void logout()

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
usuarioLogeado	Cliente logeado	Si	1
usuarioLogeado	Administrador logeado	Si	2
usuarioLogeado	= null	Si	3

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	usuarioLogeado	Cliente ("Mandarina", "miau","Mandarina")	usuarioLogeado=null y se persistío	1
2	usuarioLogeado	Administrador()	usuarioLogeado=null y se persistío	2
3	usuarioLogeado	null	usuarioLogeado=null y se persistío	3

Método: void nuevoChofer()

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
tipoChofer	"TEMPORARIO" | "PERMANENTE"	Si	1
nombreChofer	==null	No	-
nombreChofer	!=null	Si	2
nombreChofer	==vacio	No	-
nombreChofer	!=vacío	Si	3
dniChofer	==null	No	-
dniChofer	!=null	SI	5
dniChofer	no existe en la colección de choferes	Si	6

dniChofer	existe en la colección de choferes	Si	7
anioIngreso	1900<anioIngreso<3000	Si	8
cantidadHijos	<0	No	-
cantidadHijos	>=0	Si	9

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	tipoChofer	"PERMANENTE"	Se agregó un chofer a la colección de choferes	1,2,3,5,6,8,9
	nombreChofer	"Fangio"		
	dniChofer	"11888555"		
	anioIngreso	2024		
	cantidadHijos	2		
2	tipoChofer	"PERMANENTE"	Chofer Ya Registrado	1,2,3,5,7,8,9
	nombreChofer	"Fangio"		
	dniChofer	"44555666"		
	anioIngreso	2024		
	cantidadHijos	2		

Método: void nuevoPedido()

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
cliente	= null	No	-
cliente	!= null	SI	1
cliente	no tiene viajes pendientes	Si	2
cliente	tiene viajes pendientes	Si	3
cliente	tiene pedidos	Si	4

	pendientes		
cantidadPasajeros	>0	Si	5
mascota	boolean	Si	6
baul	boolean	Si	7
km	>0	Si	8

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	cliente	"Eloncito"	Se agregó un pedido a la colección de pedidos	1,2,5,6,7,8
	cantidadPasajeros	4		
	mascota	TRUE		
	baul	TRUE		
	km	100		
2	cliente	"Mandarina"	Cliente con viaje pendiente	1,3,5,6,7,8
	cantidadPasajeros	4		
	mascota	TRUE		
	baul	TRUE		
	km	100		
3	cliente	"cr7"	Cliente con pedido pendiente	1,4,5,6,7,8
	cantidadPasajeros	4		
	mascota	TRUE		
	baul	TRUE		
	km	100		

Método: void nuevoVehiculo()

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia

patente	=null	No	-
patente	!=null	Si	1
patente	no existe en la colección de vehiculos	Si	2
patente	existe en la colección de vehículos	Si	3
tipo	"MOTO" | "COMBI" | "AUTO"	Si	4
plazas	>0	Si	5
mascota	boolean	Si	6

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	patente	"ZZ444ZZ"	Se agrego un auto a la colección de vehículos	1,2,4,5,6
	tipo	"AUTO"		
	plazas	4		
	mascota	TRUE		
2	patente	"AA777AA"	Vehiculo Ya registrado	1,3,4,5,6
	tipo	"AUTO"		
	plazas	4		
	mascota	TRUE		

Método: void nuevoViaje()

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
vehiculo	=null	No	-
vehiculo	!=null	Si	1
vehiculo	disponible	Si	2
vehiculo	no disponible	Si	3

chofer	=null	No	-
chofer	!=null	Si	4
chofer	disponible	Si	5
chofer	no disponible	Si	6
pedido	=null	No	-
pedido	!=null	Si	7

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	vehiculo	Auto("KK999KK" , 4, true)	Se agregó un nuevo viaje a la colección de viajes	1,2,4,5,7
	chofer	ChoferTemporario ("33888777", "Colapinto")		
	pedido	Pedido(cr7,4,false,false,5,ZONA_SIN_ASFALTAR)		
2	vehiculo	Combi ("AA777AA" , 9, true)	Vehiculo no disponible	1,3,4,5,7
	chofer	ChoferTemporario ("33888777", "Colapinto")		
	pedido	Pedido(cr7,4,false,false,5,ZONA_SIN_ASFALTAR)		
3	vehiculo	Auto("KK999KK" , 4, true)	Chofer No disponible	1,2,4,6,7
	chofer	ChoferTemporario("44555666","Toretto")		
	pedido	Pedido(cr7,4,false,false,5,ZONA_SIN_ASFALTAR)		

Método: void registrar()

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
nombreReal	=null	No	-
nombreReal	=vacio	No	-
nombreReal	!=null & !=vacío	Si	1
usserName	=null	No	-

usserName	=vacio	No	-
usserName	!=null & !=vacío	Si	2
password	=null	No	-
password	=vacio	No	-
password	!=null & !=vacio	Si	3
confirmPassword	=null	No	-
confirmPassword	!=null	Si	4
confirmPassword	=password	Si	5
confirmPassword	!=password	Si	6

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
	nombreReal	"Martinez"		
	usserName	"pepito45"		
	password	"password"		
1	confirmPassword	"password"	Se agregó un cliente a la colección de clientes	1,2,3,4,5
	nombreReal	"Martinez"		
	usserName	"pepito45"		
	password	"password"		
2	confirmPassword	"$$$"	La contraseña y su confirmación no coinciden	1,2,3,4,6

Método: void setFileName(String fileName)

Precondiciones: fileName != null

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
fileName	= null	No	-

fileName	!= null	Si	1

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	fileName	"Empresa.bin"	Se cambio el valor del atributo fileName a	1

Método: void setPersistencia(IPersistencia persistencia)

Precondiciones: persistencia != null

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
persistencia	!=null	Si	1
persistencia	=null	No	-

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	persistencia	PersistenciaBIN()	Se cambio el valor del atributo persistencia a PersistenciaBIN ()	1

Método: void setVista(IVista vista)

Precondiciones: vista != null

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
vista	= null	No	-
vista	!= null	Si	1

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	vista	Ventana()	Se cambio el valor del atributo vista a Ventana()	1

Resultados test de integración

A continuación se presentan los errores encontrados luego de ejecutar las baterías de prueba diseñadas para el testeo de integración.

Nombre de la prueba	Error
testActionPerformed_NUEVO PEDIDO	Pedido nuevo con cantidad de km incorrecta
testNuevoChoferFallo	Mensaje de fallo de registro incorrecto, es null y debería mostrar Mensajes.CHOFER_YA_REGISTRADO
testNuevoPedidoFalloPedido Pendiente	Mensaje de fallo de registro incorrecto, deberia mostrar “cliente con pedido pendiente” y muestra “cliente con viaje pendiente”
testNuevoViajeFalloChoferNo Disponible	Mensaje de fallo de registro incorrecto, deberia mostrar “el chofer no esta disponible” pero muestra “el pedido no esta en la lista”
testNuevoViajeFalloViajePen diente	Mensaje de option pane incorrecto, es null y deberia mostrar que tiene viaje pendiente el cliente
testNuevoPedidoExito	Pedido nuevo con cantidad de km incorrecta

Test de persistencia

Para realizar el test de persistencia, se utilizó un escenario en que se creó un objeto empresaDTO para comprobar el correcto funcionamiento de los métodos de la persistencia.

Clase PersistenciaTest()

Método testEmpresaDTOfromEmpresa()

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
listas de choferes, clientes, choferes desocupados, pedidos, usuarios logueados, vehiculos desocupados, viajes iniciados, viajes terminados	listas vacías	Si	1
listas de choferes, clientes, choferes desocupados, pedidos, usuarios logueados, vehiculos desocupados, viajes iniciados, viajes terminados	listas con datos	Si	2
listas de choferes, clientes, choferes desocupados, pedidos, usuarios logueados, vehiculos desocupados, viajes iniciados, viajes terminados	null	No	

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	choferes	vacío	Archivo correctamente persistido	1
	choferesDesocupados	vacío		
	pedidos	vacío		
	usuariosLogueados	vacío		
	vehiculosDesocupados	vacío		
	viajesIniciados	vacío		
	viajesTerminados	vacío		
2	choferes	!vacío	Archivo correctamente persistido	2
	choferesDesocupados	!vacío		
	pedidos	!vacío		
	usuariosLogueados	!vacío		
	vehiculosDesocupados	!vacío		

	viajesIniciados	!vacío		
	viajesTerminados	!vacío		

Método: testEmpresaFromEmpresaDTO()

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
listas de choferes, clientes, choferes desocupados, pedidos, usuarios logueados, vehiculos desocupados, viajes iniciados, viajes terminados	listas vacías	Si	1
listas de choferes, clientes, choferes desocupados, pedidos, usuarios logueados, vehiculos desocupados, viajes iniciados, viajes terminados	listas con datos	Si	2
listas de choferes, clientes, choferes desocupados, pedidos, usuarios logueados, vehiculos desocupados, viajes iniciados, viajes terminados	null	No	

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	choferes	vacío	Objeto empresa cargado con los valores correspondientes en cada uno de sus atributos	1
	choferesDesocupados	vacío		
	pedidos	vacío		
	usuariosLogueados	vacío		
	vehiculosDesocupados	vacío		
	viajesIniciados	vacío		
	viajesTerminados	vacío		
2	choferes	!vacío	Objeto empresa cargado con los valores correspondientes en cada uno de sus atributos	2
	choferesDesocupados	!vacío		
	pedidos	!vacío		
	usuariosLogueados	!vacío		
	vehiculosDesocupados	!vacío		
	viajesIniciados	!vacío		
	viajesTerminados	!vacío		

Método: testCrearArchivo_Escritura()

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
NO HAY DATOS DE ENTRADA	fileName = “empresaTest.bin”	Sí	1

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	Archivo creado correctamente	1

Método: testAbrirArchivoSinArchivo_Lectura()

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
NO HAY DATOS DE ENTRADA	fileName = null	Sí	1

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	No debería existir el archivo empresaTest.bin	1

Método: testLeerArchivoSinArchivo_Lectura()

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	Debería lanzar una excepción FileNotFound	1

Método: testEscribirArchivoSinArchivo_Lectura()

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	Debería lanzar una excepción FileNotFound	1

Método: testPersistenciaEmpresa()

Tabla de Particiones

Dato de entrada	Descripción de la clase de equivalencia	Aplica	Identificador de clase de equivalencia
NO HAY DATOS DE ENTRADA	existe el archivo “empresaTest.bin”	Sí	1
NO HAY DATOS DE ENTRADA	no existe el archivo “empresaTest.bin”	No	

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	Clases de equivalencias abarcadas
1	-	-	Todas las colecciones de la empresa deberían ser iguales	1

Resultados test de persistencia

A continuación se presentan los errores encontrados luego de ejecutar las baterías de prueba diseñadas para el testeo de la persistencia del sistema.

Nombre de la prueba	Error
testLeerArchivoSinArchivo	Al no seleccionar un archivo para leer, debería lanzar una excepción FileNotFound
testEscribirArchivoSinArchivo	Al no seleccionar un archivo para escribir, debería lanzar una excepción FileNotFound

Test de GUI

Para abordar el test de interfaces gráficas de usuario, se empleó una de las tres categorías presentadas en teoría: métodos automatizados.

Sin embargo, el test de GUI trae aparejado un nivel de complejidad alto debido a que todos los posibles caminos que puede seguir la interacción del usuario, incluso en interfaces no muy complejas, conducen inevitablemente a una explosión en el número de estados. Por ello mismo, para llevar adelante la elaboración de pruebas funcionales se hizo uso de la clase Robot proporcionada por Java dentro del paquete AWT. La elección de esta clase también se vio reforzada ante la necesidad de simular eventos de mouse y teclado, debido al tipo de arquitectura basada en eventos que presenta el software testeado.

Debido a que la clase robot debe recibir órdenes precisas de movimiento sobre la interfaz, se nuclearon las actividades comunes en una clase denominada *TestUtil*. Entre dichas actividades comunes se definieron métodos que resolvieran la tarea de: posicionarse sobre un componente, clickear un componente, obtener un componente en base al padre, tipear texto y borrar texto.

```
public static void seleccionarElementoJList(JList<?> lista, int indice, Robot robot) {
    if (lista != null && indice >= 0 && indice < lista.getModel().getSize()) {
        // Obtener la ubicación de la JList
        Point puntoLista = lista.getLocationOnScreen();

        // Calcular la altura de cada elemento (asumiendo que todos tienen la misma altura)
        int alturaElemento = lista.getCellBounds(indice, indice).height;

        // Calcular la posición Y del elemento que queremos seleccionar
        int posY = puntoLista.y + lista.getCellBounds(indice, indice).y;

        // Mover el ratón a la posición del elemento y hacer clic
        robot.mouseMove(puntoLista.x + 5, posY + (alturaElemento / 2)); // Ajuste de 5px para centrar
        robot.mousePress(InputEvent.BUTTON1_DOWN_MASK);
        robot.delay(getDelay());
        robot.mouseRelease(InputEvent.BUTTON1_MASK);
        robot.delay(getDelay());
    }
}
```

Imagen 1 - método de TestUtil para seleccionar un elemento de un JList

Por otra parte, a la hora de comenzar con la elaboración de pruebas unitarias, se tuvieron en cuenta tres criterios de separación de las mismas:

1. Pruebas para verificar que los botones se habiliten y deshabiliten en el momento correcto.
2. Pruebas para verificar la correcta disposición visual de información y componentes, teniendo datos cargados en la empresa.
3. Pruebas para verificar la correcta disposición visual de información y componentes, teniendo la empresa vacía.

En base a estas 3 categorías, se desarrollaron tres clases para el test de GUI: *enabledDisabled, empresaConDatos* y *empresaVacia*.

Además, para cada una de estas clases, se subdividieron las pruebas unitarias en pruebas relacionadas a los distintos paneles con los que cuenta la interfaz de usuario: PanelAdmin, PanelLogin, PanelRegistro y PanelCliente.

En primer lugar, la clase *enabledDisabled* contiene pruebas para cada panel de interfaz, en el que se verifica por distintas combinaciones de ingreso de datos en los campos presentes (las más comunes) la correcta habilitación/deshabilitación de los botones. Por su parte, para las pruebas incluidas en *empresaConDatos* y *empresaVacia*, se priorizo la cobertura de los caminos más comunes que surgen de utilizar la interfaz presentada (logear exitosamente, logear con contraseña errónea, dar de alta un chofer o vehículo etc).

Resultados test de GUI

A continuación se presentan los errores encontrados luego de ejecutar las baterías de prueba diseñadas para el testeo de interfaz gráfica de usuario. Los mismos se obtuvieron al contrastar el comportamiento obtenido en la prueba con el comportamiento esperado que fue declarado en la documentación provista por la cátedra bajo el nombre de 'Anexo de comportamiento de vista'.

Nombre de la prueba	Error
testAdminNuevoAutoLlenoPlazasMenorA1	El botón de aceptarVehiculo debería estar deshabilitado, permite crear autos con 0 plazas
calificarPagar	El JTextField valor deberia estar vacío
testNuevoChoferRepetido	Mensaje incorrecto, debería mostrar Chofer Ya Registrado esperado:<Chofer Ya Registrado> pero fue:<nul
testNuevoChoferTemporario	El JTextField dni chofer debería estar vacío El JTextField nombre de chofer debería estar vacío
testNuevoChoferPermanente	El JTextField dni chofer debería estar vacío El JTextField nombre de chofer debería estar vacío El JTextField cantidad de hijos de chofer debería estar vacío El JTextField anio ingreso de chofer debería estar vacío
testNuevaCombi	El JTextField patente de vehiculo deberia estar vacío El JTextField cantidad de plazas vehiculo deberia estar vacío
testNuevaMoto	El JTextField patente de vehiculo deberia estar vacío
testNuevoAuto	El JTextField patente de vehiculo deberia estar vacío El JTextField cantidad de plazas vehiculo deberia estar vacío
testPanelAdminTextFieldsVacios	El JTextField sueldos totales debería estar vacío El JTextField patente debería estar vacío El JTextField dni chofer debería estar vacío El JTextField nombre chofer debería estar vacío El JTextField año ingreso debería estar vacío El JTextField cantidad de plazas debería estar vacío

Conclusiones

Con la totalidad de las pruebas y resultados ya expuestos, y teniendo en cuenta el proceso de desarrollo del trabajo, es posible concluir que la actividad de testing del software no implica un nivel de dificultad alto referido al entendimiento de conceptos o decisiones, como bien podría incluirlo la fase de diseño del sistema, en la cual deben evaluarse distintos tipos de arquitecturas y asignación de responsabilidades. Sin embargo, sí representa una tarea tediosa, debido al gran número de módulos y casos de prueba que se presentan en cada uno, lo cual conlleva una tarea repetitiva de definición y ejecución de pruebas.

De todos modos, también es posible afirmar que la actividad de testing es indispensable para cualquier proyecto de software. Esto se debe a que es el momento en el cual se identifican errores antes que el producto alcance al usuario final, logrando así una mejor experiencia de usuario, permitiendo también mejorar la calidad del sistema desarrollado al detectar incongruencias con las especificaciones y, en casos de proyectos más grandes, permitiendo reducir costos económicos a largo plazo.

El desarrollo de este trabajo permitió adquirir una nueva perspectiva sobre el código, distinta a la que se había trabajado hasta este punto de la carrera. En lugar de centrarse únicamente en el diseño o en la codificación del sistema, y enfocarse en aspectos como la eficiencia del código o una adecuada distribución de responsabilidades en la Programación Orientada a Objetos (POO), se comenzó a considerar y anticipar los posibles errores que pueden surgir de la interacción del usuario con el sistema que estamos desarrollando. Este cambio de enfoque ayudó a comprender la importancia de prever y manejar las excepciones y fallos, lo cual es fundamental para mejorar la robustez, fiabilidad y usabilidad del software.