



Trabajo Final de testing

Asignatura: Taller de programación 1

Grupo 10

Integrantes: Bondoni Juan

Giardino Tomás

Agustín Mateo Chazarreta Boada

Ezequiel Facundo Rodríguez

URL del repositorio GitHub:

https://github.com/rodriquel/2024_Grupo_10.git

Desarrollo

Test de caja negra

Clase: *Usuario*

Constructor: `public Usuario(String nombreUsuario, String pass)`

Pre: Los parámetros “*nombreUsuario*”, y “*pass*” son diferentes de null y tienen al menos un caracter.

Parámetros

nombreUsuario: Objeto de tipo String que indica el nombre identificador del usuario

pass: Objeto de tipo String que indica el password del usuario

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
nombreUsuario	nombreUsuario == null	no	1
pass	pass == null	no	2
nombreUsuario	nombreUsuario == ""	no	3
pass	pass == ""	no	4
nombreUsuario	tiene al menos un caracter	si	5
pass	tiene al menos un caracter	si	6

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	nombreUsuario	Juan	Objeto de tipo usuario	5 y 6	no aplica
	pass	1245			

Método: `public String getNombreUsuario()`

Retorna: el atributo *nombreUsuario*

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
------------------	------------------	-------	-----------------	--------	--------------

1	nombreUsuario	juan	juan		
---	---------------	------	------	--	--

Método: `public String getPass()`

Retorna: el atributo `pass`

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	pass	1234	1234		

Método: `public String toString()`

Overrides: `toString` in class `Object`

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1			Objeto de tipo String		

Clase: `Cliente`

Constructor: `public Cliente(String nombreUsuario, String pass, String nombreReal)`

Pre: Los parámetros “*nombreUsuario*”, “*pass*” y “*nombreReal*” son diferentes de null y tienen al menos un carácter.

Parámetros

nombreUsuario: Objeto de tipo String que indica el nombre identificador del cliente

pass: Objeto de tipo String que indica el password del cliente

nombreReal: Objeto de tipo String que indica el nombre real del cliente

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
nombreUsuario	nombreUsuario == null	no	1
pass	pass == null	no	2
nombreReal	nombreReal == null	no	3
nombreUsuario	nombreUsuario == ""	no	4
pass	pass == ""	no	5
nombreReal	nombreReal == ""	no	6
nombreUsuario	tiene al menos un caracter	si	7
pass	tiene al menos un caracter	si	8
nombreReal	tiene al menos un caracter	si	9

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1 (cliente1)	nombreUsuario	Juan95	Objeto de tipo cliente	7, 8 y 9	no aplica
	pass	1245			
	nombreReal	juan			

Método: `public String getNombreReal()`

Retorna: el atributo **nombreReal**

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	nombreReal	juan	juan		

Método: public String toString()

Overrides: toString in class Usuario

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1			Objeto de tipo String		

Clase: Administrador

Método: public static Administrador getInstance()

Retorna: la única instancia del Administrador, donde *nombreUsuario*, y *pass* valen ambas "admin"

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1			Objeto de tipo Administrador (con nombreUsuario == pass == admin)		

Clase: Chofer

Método: public Chofer(String dni, String nombre)

Pre: Los parámetros *dni* y *nombre* son distintos de null y tienen al menos un caracter.

Parámetros

dni: Corresponde al dni del Chofer

nombre: Corresponde al nombre y apellido real del chofer

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
dni	dni == null	no	1
nombre	nombre == null	no	2
dni	dni == ""	no	3
nombre	nombre == ""	no	4
dni	tiene al menos un caracter	si	5
nombre	tiene al menos un caracter	si	6

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1 (chofer1)	dni	43666918	Objeto de tipo Chofer	5 y 6	no aplica
	nombre	Rodrigo			

Método: `public static double getSueldoBasico()`

Retorna: el atributo sueldoBasico.

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	sueldoBasico	100000.0	sueldoBasico == Chofer1.getSueldoBasico()		

Método: `public static void setSueldoBasico(double sueldoBasico)`

Pre: el parámetro **sueldoBasico** es positivo.

Parametros

sueldoBasico: Representa el valor del sueldo basico para realizar los calculos de sueldo.

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
suldoBasico	suldoBasico > 0	si	1
suldoBasico	suldoBasico < 0	no	2
suldoBasico	suldoBasico == 0	no	3

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	sueldoBasico	100000.0	inicializar el atributo "suledoBasico" (sueldoBasico == Chofer1.sueldo Basico)	1	

Método: public String getDni()

Retorna: el atributo dni del chofer

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	dni	23456713	23456713		

Método: public String getNombre()

Retorna: el atributo **nombre** del chofer.

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	nombre	Bautista	Bautista		

Método: public String toString()

Overrides: toString in class Object

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1			Objeto de tipo String		

Método: public abstract double getSueldoBruto()

Se analiza el método para los dos tipos de choferes en sus respectivas clases.

Método: public double getSueldoNeto()

Se analiza el método para los dos tipos de choferes en sus respectivas clases.

Clase: ChoferPermanente

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

Pre:

Los parámetros **dni** y **nombre** son distintos de null y tienen al menos un caracter.

El parámetro **anioIngreso** esta comprendido entre 1900 y 3000.

El parámetro **cantidadHijos** es mayor o igual que cero.

Parámetros

dni: Corresponde al dni del Chofer

nombre: Corresponde al nombre y apellido real del chofer

anioIngreso: Corresponde al a Año de ingreso del chofer permanente

cantidadHijos: Indica la cantidad de hijos del chofer permanente

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
dni	dni == null	no	1
nombre	nombre == null	no	2
dni	dni == ""	no	3
nombre	nombre == ""	no	4
anioIngreso	anioIngreso > 3000	no	5
anioIngreso	anioIngreso < 1900	no	6
cantidadHijos	cantidadHijo < 0	no	7
cantidadHijos	cantidadHijo >= 0	si	8
anioIngreso	1900 < anioIngreso < 3000	si	9
dni	tiene al menos un caracter	si	10
nombre	tiene al menos un caracter	si	11

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1 (choferP1)	dni	24532189	Objeto de tipo ChoferPermanente	8, 9, 10 y 11	No se tiene que tener en cuenta
	nombre	Pedro			
	anioIngreso	2020			

	cantidadHijo	6			
--	--------------	---	--	--	--

Método: public int getCantidadHijos()

Retorna: el atributo *cantidadHijos*.

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1			cantidadHijo >= 0		

Método: public void setCantidadHijos(int cantidadHijos)

Pre: El parámetro *cantidadHijos* es mayor o igual que cero.

Parámetros

cantidadHijos: Indica la cantidad de hijos del chofer permanente

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
cantidadHijo	cantidadHijo < 0	no	1
cantidadHijo	cantidadHijo >= 0	si	2

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	cantidadHijos	4	4	2	

Método: public double getSueldoBruto()

El sueldo bruto se calcula incrementando el sueldo basico 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

Specified by: `getSueldoBruto` in class `Chofer`

Retorna: el valor del **sueldo bruto** del chofer.

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	dni	24532189	suldoBruto == 3240		
	nombre	Pedro			
	anioIngreso	2020 (2024 año ref)			
	sueldoBasico	2000			
	cantidadHijo	6			

Método: `public double getSueldoNeto()`

Retorna: el valor del **sueldoNeto** del chofer, el cual corresponde al 86% del **sueldoBruto**, luego de realizadas las correspondientes retenciones.

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	setSueldoBasico(double sueldoBasico)	sueldoBasico = 300000.0	getSueldoNeto() == 0.86 * getSueldoBruto()		

Método: `public int getAntigüedad()`

Retorna: La diferencia entre el año actual y el año de ingreso

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1			antigüedad >= 0		

Método: `public int getAnioIngreso()`

Retorna: el atributo *anioIngreso*.

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1			1900 <= anioIngreso <= 3000		

Método: `public String toString()`

Overrides: `toString` in class `Chofer`

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1			Objeto de tipo String		

Clase: `ChoferTemporario`

Método: `public ChoferTemporario(String dni, String nombre)`

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

Parámetros:

dni: Corresponde al dni del Chofer

nombre: Corresponde al nombre y apellido real del chofer

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
dni	dni == null	no	1
nombre	nombre == null	no	2
dni	dni == ""	no	3
nombre	nombre == ""	no	4
dni	tiene al menos un caracter	si	5
nombre	tiene al menos un caracter	si	6

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1 (choferT1)	dni	24532189	Objeto de tipo ChoferTemporario	5, 6	
	nombre	Pedro			

Metodo: `public double getSueltoBruto()`

El sueldo bruto es igual al sueldo básico.

Specified by:

`getSueltoBruto` in class `Chofer`

Retorna: valor del **sueldo bruto** del chofer.

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
------------------	------------------	-------	-----------------	--------	--------------

1	dni	24532189	suldoBruto == sueldoBasico		
	nombre	Pedro			
	sueldoBasico	2000			

Método: public double getSueldoNeto()

Retorna: el valor del **sueldoNeto** del chofer, el cual corresponde al 86% del **sueldoBruto**, luego de realizadas las correspondientes retenciones.

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	setSueldoBasico(double sueldoBasico)	sueldoBasico = 300000.0	getSueldoNeto() == 0.86 * getSueldoBruto()		

Método: public String toString()

Overrides: toString in class Chofer

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1			Objeto de tipo String		

Clase: Pedido

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

Pre: 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**

Parámetros:

cliente: Objeto de tipo Cliente que indica el cliente que realiza el pedido

cantidadPasajeros: indica la cantidad de pasajeros solicitada para el pedido

mascota: indica si se solicita o no el traslado de mascotas

baul: indica si se requiere o no el uso de baul para equipajes

km: indica la cantidad de km a recorrer

zona: indica el tipo de zona a recorrer. Esta definido por alguna de las siguientes constantes: **Constantes.ZONA_PELIGROSA** , **Constantes.ZONA_SIN_ASFALTAR** o **Constantes.ZONA_STANDARD**

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
cliente	cliente == null	no	1
cantidadPasajeros	cantidadPasajeros <= 1	no	2
km	km < 0	no	3
zona	zona != Constantes.ZONA_PELIGROSA && zona != Constantes.ZONA_SIN_ASFALTAR && zona != Constantes.ZONA_STANDARD	no	4
cliente	cliente != null	si	5
cantidadPasajeros	cantidadPasajeros > 1	si	6
km	km >= 0	si	7
zona	zona == Constantes.ZONA_PELIGROSA zona == Constantes.ZONA_SIN_ASFALTAR zona == Constantes.ZONA_STANDARD	si	8
mascota	mascota == true	si	9
mascota	mascota == false	si	10
baul	baul == true	si	11
baul	baul == false	si	12

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1 (pedido1)	cliente	cliente1	Objeto de tipo Pedido (cliente == cliente1 cantidadPasajeros == 3 mascota == true baul == false km == 200 zona == Constantes.ZONA_SIN_ASFALTAR)	5, 6, 7, 8, 9 y 12	
	cantidadPasajeros	3			
	mascota	true			
	baul	false			
	km	200			
	zona	Constantes.ZONA_SIN_ASFALTAR			
2 (pedido2)	cliente	cliente1	Objeto de tipo Pedido (cliente == cliente1 cantidadPasajeros == 3 mascota == true baul == true km == 200 zona == Constantes.ZONA_SIN_ASFALTAR)	5, 6, 7, 8, 9 y 11	
	cantidadPasajeros	3			
	mascota	true			
	baul	true			
	km	200			
	zona	Constantes.ZONA_SIN_ASFALTAR			
2 (pedido2)	cliente	cliente1	Objeto de tipo Pedido (cliente == cliente1 cantidadPasajeros == 3 mascota == true baul == true km == 200 zona == Constantes.ZONA_SIN_ASFALTAR)	5, 6, 7, 8, 10 y 11	
	cantidadPasajeros	3			
	mascota	false			
	baul	true			
	km	200			
	zona	Constantes.ZONA_SIN_ASFALTAR			

Método: `public Cliente getCliente()`

Retorna: el atributo cliente.

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	pedido1		pedido1.getCliente() == pedido1.cliente1		

Método: public boolean isMascota()

Retorna: el atributo mascota

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	pedido1		pedido1.isMascota() == pedido1.mascota		

Método: public boolean isBaul()

Retorna: el atributo baul

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	pedido1		pedido1.isBaul() == pedido1.baul		

Método: public int getKm()

Retorna: el atributo km

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	pedido1		pedido1.getKm() == pedido1.km		

Método: public String getZona()

Retorna: el atributo zona

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	pedido1		pedido1.getZona() == pedido1.zona		

Método: public int getCantidadPasajeros()

Retorna: el atributo cantidadPasajeros

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	pedido1		pedido1.getCantidadPasajeros() == pedido1.cantidadPasajeros		

Método: public String toString()

Overrides:

toString in class [Object](#)

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1			Objeto de tipo String		

Clase: Viaje

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

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

Parameters:

pedido: Objeto de tipo Pedido a partir del cual se genera el viaje

chofer: Objeto de tipo Chofer que indica el chofer que realiza el viaje

vehículo: Objeto de tipo Vehiculo que indica el vehículo asignado al viaje

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
pedido	pedido == null	no	1
chofer	chofer == null	no	2
vehículo	vehiculo == null	no	3
pedido	pedido != null	si	4
chofer	chofer != null	si	5
vehiculo	vehiculo!= null	si	6

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1 (viaje1)	pedido	pedido1	Objeto de tipo Viaje y viaje1.getValorBase() == 1000	4, 5, 6	
	chofer	choferT1			
	vehiculo	auto1			

Metodo: public static double getValorBase()

Retorna: el atributo valorBasse.

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	setValorBase(1000)		setValorBase(1000) == getValorBase()		

Metodo: public static void setValorBase(double valorBase)

Pre: El parámetro **valorBase** es positivo.

Parámetros

valorBase: Corresponde al valor base para realizar el cálculo del costo del viaje.

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
valorBase	valorBase < 0	no	1
valorBase	valorBase > 0	si	2
valorBase	valorBase == 0	no	3

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	valorBase	2000	valorBase == 2000	2	

Metodo: public Pedido getPedido()

Retorna: el atributo pedido.

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	viaje1 = new Viaje(pedido1,chofer1,vehiculo1)		pedido1 == viaje1.getPedido()		

Metodo: public Chofer getChofer()

Returns:

Retorna el atributo chofer

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	<i>viaje1 = new Viaje(pedido1,chofer1,vehiculo1)</i>		chofer1 == viaje1.getChofer()		

Metodo: public Vehiculo getVehiculo()

Retorna: el atributo vehiculo

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	<i>viaje1 = new Viaje(pedido1,chofer1,vehiculo1)</i>		vehiculo1 == viaje1.getVehiculo()		

Metodo: public boolean isFinalizado()

Retorna: el atributo finalizado

Batería de pruebas

Número de	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
-----------	------------------	-------	-----------------	--------	--------------

prueba					
1	<i>viaje1.finalizarViaje(5)</i>		finalizado == true		
2	<i>viaje1 = new Viaje(pedido1,chofer1,vehiculo1)</i>		finalizado == false		

Metodo: `public void finalizarViaje(int calificacion)`

Pre: El parametro **calificacion** es mayor o igual que cero y menor o igual que 5. Setea el atributo **calificacion** de acuerdo al parametro calificacion. Setea el atributo **finalizado** en true.

Parametros:

calificacion: Indica la **calificacion** del viaje.

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
calificacion	calificacion < 0	no	1
calificacion	calificacion > 5	no	2
calificacion	calificacion >=0 && calificacion <=5	si	3

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	calificacion	2	calificacion == 2 finalizado == true	3	

Metodo: `public int getCalificacion()`

Retorna: el atributo calificacion

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1			0 <= calificacion <= 5		

Metodo: `public String toString()`

Overrides:

`toString` in class `Object`

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1			Objeto de tipo String		

Metodo: `public double getValor()`

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 baul** se incrementa un 10% por cada pasajero y 5% por cada km

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	pedido1	cliente: cliente1 cantidadPasajeros: 3 mascota: true baul: false km: 200 zona: Constantes.ZONA_SIN_ASFALTAR valorBase: 1000	getValor() = $1000 + 0.2 * 3 + 0.15 * 200 + 0.1 * 3 + 0.2 * 200 = 1070.9$		
2	pedido2	cliente: cliente1 cantidadPasajeros: 3 mascota: false baul: true km: 200 zona: Constantes.ZONA_SIN_ASFALTAR valorBase: 1000	getValor() = 1040.9		
3	pedido3	cliente: cliente1 cantidadPasajeros: 3 mascota: true baul: false km: 200 zona: Constantes.ZONA_STANDARD valorBase: 1000	getValor() = 1060.6		
4	pedido4	cliente: cliente1 cantidadPasajeros: 3 mascota: true baul: false km: 200 zona: Constantes.ZONA_PELIGROSA valorBase: 1000	getValor() = 1080.6		

Clase: *Vehiculo*

Método: *public Vehiculo(String patente, int cantidadPlazas, boolean mascota)*

Pre:

El parámetro **patente** es distinto de null.

El parámetro **cantidadPlazas** es positivo

Parámetros:

patente: String que representa la patente

cantidadPlazas: Cantidad de plazas de las que dispone el vehiculo

mascota: Indica si el vehiculo acepta traslado de mascota

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
patente	patente == null	no	1
cantidadPlazas	cantidadPlazas <= 0	no	2
patente	patente != null	si	3
cantidadPlazas	cantidadPlazas >0	si	4
mascota	false	si	5
mascota	true	si	6

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	patente	ABC123	Objeto de tipo Vehiculo	3,4,6	
	cantidadPlazas	2			
	mascota	true			
2	patente	ABC123	Objeto de tipo Vehículo	3,4,5	
	cantidadPlazas	2			
	mascota	false			

Metodo: getPatente()

Retorna:

Retorna el atributo patente

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	patente = ABC123		getPatente() == ABC123		

Metodo: getCantidadPlazas()

Retorna:

Retorna el atributo cantidadPlazas

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	cantidadPlazas=2		getCantidadPlazas() == 2		

Metodo: `public boolean isMascota()`

Retorna: el atributo mascota

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	Mascota=true		isMascota() == true		
2	Mascota=false		isMascota() == false		

Metodo: `public abstract Integer getPuntajePedido(Pedido pedido)`

Se analiza el método para los tres tipos de vehículos en sus respectivas clases.

Clase: `Auto`

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

Pre:

El parámetro **patente** es distinto al de null.

El parámetro **cantidadPlazas** es positivo y menor que 5

Parámetros:

patente: String que representa la patente

cantidadPlazas: Cantidad de plazas de las que dispone el vehiculo

mascota: Indica si el vehiculo acepta traslado de mascota

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
patente	patente ==null	no	1
cantidadPlazas	cantPlazas > 5	no	2
cantidadPlazas	cantPlazas < 0	no	3
patente	patente !=null	si	4
cantidadPlazas	0 < cantPlazas < 5	si	5
mascota	true	si	6
mascota	false	si	7

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1 (auto1)	patente	ABC123	Objeto de tipo Vehículo	4,5,6	
	cantidadPlazas	2			
	mascota	true			
2	patente	ABC123	Objeto de tipo Vehículo	4,5,7	
	cantidadPlazas	2			
	mascota	false			

Metodo: `public Integer getPuntajePedido(Pedido pedido)`

Pre:

El parámetro **pedido** es distinto de null

Specified by:

`getPuntajePedido` in class `Vehículo`

Parámetros:

pedido: Objeto de tipo Pedido que se deberá evaluar.

Retorna: el puntaje del vehículo en relación al pedido en cuestión, de acuerdo a la siguiente fórmula:

Si el vehículo puede satisfacer el pedido, solicitando el uso de baúl el valor es $40 * \text{cantPax}$. Sino el valor es $30 * \text{cantPax}$

Si el vehículo no puede satisfacer la necesidades del pedido, se retorna null.

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
pedido	pedido== null	no	1
pedido	pedido!=null	si	2

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	pedido(con cantPax > 0 y cantPax <= 4; con baúl)	cantPax==3	120	2	
2	pedido(con cantPax < 0 ó cantPax > 4)	cantPax==6	null	2	
3	pedido(con cantPax > 0 y cantPax <= 4; sin baúl)	canPax==3	90	2	

Metodo: `public String toString()`

Overrides:

`toString` in class `Object`

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
------------------	------------------	-------	-----------------	--------	--------------

1			Objeto de tipo String		
---	--	--	-----------------------	--	--

Clase: *Combi*

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

Pre:

El parámetro **patente** es distinto de null.

El parámetro **cantidadPlazas** es mayor que 4 y menor que 11.

Parámetros:

patente: String que representa la patente

cantidadPlazas: Cantidad de plazas de las que dispone el vehículo

mascota: Indica si el vehículo acepta traslado de mascota

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
patente	patente == null	no	1
cantidadPlazas	cantidadPlazas >= 11	no	2
cantidadPlazas	cantidadPlazas <= 4	no	3
patente	patente != null	si	4
cantidadPlazas	4 < cantidadPlazas < 11	si	5
mascota	false	si	6
mascota	true	si	7

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	patente	ABC123	Objeto de tipo Combi	4,5 y 6	
	cantidadPlazas	6			
	mascota	false			

2	patente	ABC123	Objeto de tipo Combi	4,5 y 7	
	cantidadPlazas	6			
	mascota	true			

Metodo: `public Integer getPuntajePedido(Pedido pedido)`

Pre: El parámetro **pedido** es distinto de null

Specified by:

`getPuntajePedido` in class `Vehiculo`

Parámetros:

pedido: Objeto de tipo `Pedido` que se deberá evaluar

Retorna: el puntaje del vehículo en relación al pedido en cuestion de acuerdo a la siguiente formula:
Si el vehículo puede satisfacer el pedido, el valor es $10 * \text{cantPax}$. Si el pedido solicita uso de baúl se suma 100 puntos.

Si el vehículo no puede satisfacer la necesidades del pedido, se retorna null.

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
pedido	pedido == null	no	1
pedido	pedido != null	si	2

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	pedido(con cantPax > 4 y cantPax < 11; con baúl)	cantPax == 6	160	2	

2	pedido(con cantPax < 5 ó cantPax > 10)	cantPax==20	null	2	
3	pedido(con cantPax > 4 y cantPax < 11; sin baúl)	canPax==6	60	2	

Metodo: `public String toString()`

Overrides:

`toString` in class `Object`

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1			Objeto de tipo String		

Clase: `Moto`

Metodo: `public Moto(String patente)`

Pre: El parámetro **patente** es distinto de null.

El atributo **cantidadPlazas** se inicializa en 1

El atributo **mascota** se inicializa en false

Parámetros:

patente: String que representa la patente

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
patente	patente == null	no	1
patente	patente != null	si	2

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	patente	ABC123	Objeto de tipo Moto	2	

Metodo: `public Integer getPuntajePedido(Pedido pedido)`

Pre: El parámetro **pedido** es distinto de null

Retorna el puntaje del vehiculo en relación al pedido

Si el vehiculo no puede satisfacer la necesidades del pedido, se retorna null.

Specified by:

`getPuntajePedido` in class `Vehiculo`

Parámetros:

pedido: Objeto de tipo Pedido que se deberá evaluar

Retorna:

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

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
pedido	pedido== null	no	1
pedido	pedido!=null	si	2

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
------------------	------------------	-------	-----------------	--------	--------------

1	pedido(con cantPax == 1, sin baul, sin mascota)	cantPax==1	1000	2	
2	pedido(con cantPax > 1, sin baul y sin mascota)	cantPax==10	null	2	
3	pedido(con cantPax == 1, con baul y sin mascota)	cantPax==1	null	2	
4	pedido(con cantPax == 1, sin baul y con mascota)	cantPax==1	null	2	

Método: `public String toString()`

Overrides:

`toString` in class `Object`

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1			Objeto de tipo String		

Clase: `Empresa`

Invariante de clase Los siguientes atributos son todos diferentes de null :

`HashMap[String, Cliente]` clientes

`HashMap[String, Chofer]` choferes

`HashMap[String, Vehiculo]` vehiculos

`ArrayList[Chofer]` choferesDesocupados

`ArrayList[Vehiculo]` vehiculosDesocupados

`HashMap[Cliente, Pedido]` pedidos

`HashMap[Cliente, Viaje]` viajesIniciados

`ArrayList[Viaje]` viajesTerminados

Método: `public static Empresa getInstance()`

Retorna:

Instancia de clase Empresa, aplica el patrón Singleton

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	Empresa1 y Empresa2	Empresa1= Empresa.getInstance() y Empresa2= Empresa.getInstance()	Objeto de tipo Empresa sea Único. Empresa1 == Empresa2 Empresa1.getClientes() != null Empresa1.getChoferes() != null Empresa1.getVehiculos() != null Empresa1.getVehiculosDesocupados() != null Empresa1.getChoferesDesocupados() != null Empresa1.getPedidos() != null Empresa1.getViajesIniciados() != null Empresa1.getViajesTerminados() != null		

Método: `public HashMap <String, Cliente> getClientes()`

Retorna:

Atributo clientes, HashMap [String, Cliente], donde la clave es el nombre de usuario de cada Cliente

El atributo clientes conserva la totalidad de los clientes registrados.

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	<code>clientes1 = new HashMap("Juan95", cliente1)</code>	<code>setClientes(clientes1)</code>	<code>Empresa.getClientes() == clientes1</code>		

Método: `public void setClientes(HashMap<String, Cliente> clientes)`

Parámetros

clientes: Setter del atributo clientes

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
clientes	clientes != null	si	1
clientes	clientes == null	no	2

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	<code>clientes1 = new HashMap("Juan95", cliente1)</code>	<code>setClientes(clientes1)</code>	<code>Empresa.getClientes() == clientes1</code>	1	

Método: `public ArrayList<Chofer> getChoferesDesocupados()`

Retorna: atributo **choferesDesocupados**. Representa los choferes desocupados, es decir, que no están ocupados realizando un viaje

Batería de pruebas

Número de pruebas	Datos de entrada	valor	salida esperada	clases	valor límite
-------------------	------------------	-------	-----------------	--------	--------------

1	choferesDesocupados1 = new ArrayList(choferP1)	setChoferesDesocupados(choferesDesocupados1)	empresa.getChoferesDesocupados() == choferesDesocupados1		
---	--	--	--	--	--

Método: public void setChoferesDesocupados(ArrayList<Chofer> choferesDesocupados)

Parámetros

choferesDesocupados: Setter del atributo choferesDesocupados

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
choferesDesocupados	choferesDesocupados != null	si	1
choferesDesocupados	choferesDesocupados == null	no	2

Batería de pruebas

número de pruebas	datos de entrada	valor	salida esperada	clases	valor límite
1	choferesDesocupados1 = new ArrayList(choferP1)	setChoferesDesocupados(choferesDesocupados1)	empresa.getChoferesDesocupados() == choferesDesocupados1	1	

Método: public HashMap<String, Chofer> getChoferes()

Retorna: atributo choferes, HashMap [String, Choferes], donde la clave es el dni de cada Chofer
el atributo choferes representa la totalidad de los choferes registrados

Batería de pruebas

número de pruebas	datos de entrada	valor	salida esperada	clases	valor límite
-------------------	------------------	-------	-----------------	--------	--------------

1	choferes1=new HashMap(dni:"24 532189",chofer P1)	setChoferes(ch oferes1)	empresa.getChof eres() == choferes1		
---	---	----------------------------	---	--	--

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

Parámetros

choferes: Setter del atributo choferes

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
choferes	choferes != null	si	1
choferes	choferes == null	no	2

Batería de pruebas

número de pruebas	datos de entrada	valor	salida esperada	clases	valor límite
1	choferes1=new HashMap(dni:"24 532189",chofer P1)	setChoferes(ch oferes1)	empresa.getCh oferes() == choferes1	1	

Método: public HashMap<String, Vehiculo> getVehiculos()

Retorna: atributo vehiculos, HashMap [String,Vehiculo], donde la clave es la patente de cada Vehiculo
el atributo vehiculos representa la totalidad de los vehiculos registrados

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
------------------	------------------	-------	-----------------	--------	--------------

1	<code>vehiculos1 = new HashMap("ABC123", auto1)</code>	<code>setVehiculos(vehiculos1)</code>	<code>Empresa.getVehiculos() == vehiculos1</code>		
---	--	---------------------------------------	---	--	--

Metodo: `public void setVehiculos(HashMap<String, Vehiculo> vehiculos)`

Parámetros

vehiculos: Setter del atributo vehiculos

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
vehiculos	vehiculos != null	si	1
vehiculos	vehiculos == null	no	2

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	<code>vehiculos1 = new HashMap("ABC123", auto1)</code>	<code>setVehiculos(vehiculos1)</code>	<code>Empresa.getVehiculos() == vehiculos1</code>	1	

Método: `public ArrayList<Vehiculo> getVehiculosDesocupados()`

Retorna: atributo vehiculosDesocupados. Representa los vehiculos desocupados, es decir, que no están ocupados realizando un viaje.

Batería de pruebas

Número de pruebas	Datos de entrada	valor	salida esperada	clases	valor límite
1	<code>vehiculosDesocupados1 = new ArrayList(auto1)</code>	<code>setVehiculosDesocupados(vehiculosDesocupados1)</code>	<code>empresa.getVehiculosDesocupados() == vehiculosDesocupados1</code>		

Método: `public void setVehiculosDesocupados(ArrayList<Vehiculo> vehiculosDesocupados)`

Parámetros

vehiculosDesocupados: Setter del atributo vehiculosDesocupados

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
vehiculosDesocupados	vehiculosDesocupados != null	si	1
vehiculosDesocupados	vehiculosDesocupados == null	no	2

Batería de pruebas

número de pruebas	datos de entrada	valor	salida esperada	clases	valor límite
1	vehiculosDesocupados1 = new ArrayList(auto1)	setVehiculosDesocupados(vehiculosDesocupados1)	empresa.getVehiculosDesocupados() == vehiculosDesocupados1	1	

Método: `public HashMap<Cliente, Pedido> getPedidos()`

Retorna: atributo pedidos, HashMap [Cliente, Pedido], donde la clave es el Cliente que realizo el Pedido
El atributo pedidos representa los pedidos solicitados por los clientes.

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	pedidos1 = new HashMap(cliente1, pedido1)	setPedidos(pedidos1)	Empresa.getPedidos() == pedidos1		

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

Parámetros

pedidos: Setter del atributo pedidos

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
pedidos	pedidos != null	si	1
pedidos	pedidos == null	no	2

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	pedidos1= new HashMap<cliente1, pedido1>	setPedidos(pedidos1)	Empresa.getPedidos() == pedidos1	1	

Método: public HashMap<Cliente, Viaje> getViajesIniciados()

Retorna: atributo viajesIniciados, HashMap [Cliente, Viaje], donde la clave es el Cliente que solicito el Viaje
El atributo viajesIniciados representa los viajes iniciados pero no terminados.

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	viajesIniciados1 = new HashMap<cliente1, viaje1>	setViajesIniciados(viajesIniciados1)	Empresa.getViajesIniciados() == viajesIniciados1		

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

Parámetros

viajesIniciados: Setter del atributo viajesIniciados

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
------------------	---	------------------------------	---------------------------

viajesIniciados	viajesIniciados != null	si	1
viajesIniciados	viajesIniciados == null	no	2

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	viajesIniciados1 = new HashMap (cliente1, viaje1)	setViajesIniciados(viajesIniciados1)	Empresa.getViajesIniciados() == viajesIniciados1	1	

Método: public ArrayList<Viaje> getViajesTerminados()

Retorna: atributo viajesTerminados. Representa los viajes ya finalizados. Ya han sido pagados y calificados.

Batería de pruebas

Número de pruebas	Datos de entrada	valor	salida esperada	clases	valor límite
1	viajesTerminados1 = new ArrayList(viaje1)	setViajesTerminados(viajesTerminados1)	empresa.getViajesTerminados() == viajesTerminados1		

Método: public void setViajesTerminados(ArrayList<Viaje> viajesTerminados)

Parámetros

viajesTerminados: Setter del atributo viajesTerminados

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
viajesTerminados	viajesTerminados != null	si	1
viajesTerminados	viajesTerminados == null	no	2

Batería de pruebas

número de pruebas	datos de entrada	valor	salida esperada	clases	valor límite
1	viajesTerminados1 = new ArrayList(viaje1)	setViajesTerminados(viajesTerminados1)	empresa.getViajesTerminados == viajesTerminados1	1	

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

Pre: El parámetro chofer es distinto de null

Agrega un chofer al HashMap de choferes si no está repetido

Parámetros

chofer: Objeto de tipo Chofer que se agrega.

Throws

ChoferRepetidoException: Se lanza en caso de que el dni del chofer pasado por parámetro coincida con el dni de un chofer previamente registrado.

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
chofer	== null	no	1
chofer	!= null	si	2

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	chofer	choferTemporalT1	Chofer agregado al HashMap	2	
2	chofer	choferTemporalT1	Lanza excepcion: ChoferRepetidoException con mensaje CHOFER_YA_REGISTRADO	2	
3	chofer	choferTemporalT1	Fail("Mensaje de la excepcion != CHOFER_YA_REGISTRADO")	2	

4	chofer	choferTemporalT1	Fail("No se lanzo la excepcion ChoferRepetidoException ")	2	
5			Fail("Se lanzo una excepcion que no deberia lanzarse")	2	

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

Pre: los parametros usuario, pass y nombreReal son diferentes de null y tienen al menos un caracter
Agrega un Cliente al HashMap de clientes si no esta repetido

Parametros

usuario: Nombre de usuario del Cliente

pass: Password del Cliente

nombreReal: Nombre real del Cliente

Throws

UsuarioYaExisteException: Se lanza en caso de que el parámetro usuario coincida con el nombre de usuario de algún Cliente previamente registrado.

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
usuario	usuario == null	no	1
pass	pass == null	no	2
nombreReal	nombreReal == null	no	3
usuario	usuario == ""	no	4
pass	pass == ""	no	5
nombreReal	nombreReal == ""	no	6
usuario	usuario != null && usuario != ""	si	7
pass	pass != null && pass != ""	si	8
nombreal	nombreReal != null && nombreRea != ""	si	9

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
------------------	------------------	-------	-----------------	--------	--------------

1	usuario	usuario1	Cliente agregado al hashmap Empresa.getClientes().size() == 1	7,8 y 9	
	pass	1234			
	nombreReal	Juan			
2	usuario	usuario1	Lanza excepción: UsuarioYaExisteException	7,8 y 9	
	pass	1234			
	nombreReal	Juan			
3	usuario	usuario1	Fail("No se lanzo la excepcion UsuarioYaExisteException")	7,8 y 9	
	pass	1234			
	nombreReal	Juan			
4	usuario	usuario1	Fail("Se lanzo una excepcion que no deberia lanzarse") o Fail("Mensaje de la excepcion != Mensajes.USUARIO_REPETIDO")	7,8 y 9	
	pass	1234			
	nombreReal	Juan			

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

Pre: el parámetro pedido es diferente de null

Parámetros

pedido: objeto de tipo Pedido que se agrega al HashMap pedidos

Throws

SinVehiculoParaPedidoException: Se lanza en caso de que la empresa no tenga registrado ningun vehiculo con las caracteristicas 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 parámetro pedido, no es un cliente registrado en la empresa

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
pedido	== null	no	1
pedido	!= null	si	2

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	pedido1=new Pedido(cliente1,...)	<p>Empresa.agregarCliente(usuario1,1234,juan)</p> <p>Vehiculo v1 = new Combi("0809ad",6,true)</p> <p>Empresa.agregarVehiculo(v1);</p> <p>v1 satisface al pedido1</p> <p>Empresa.AgregarPedido(pedido1)</p>	Empresa.getPedidos().size() - 1 == 0	2	
2	<p>pedido1=new Pedido(cliente1,...)</p> <p>pedido2=new Pedido(cliente1,...)</p>	<p>Empresa.agregarCliente(usuario1,1234,juan)</p> <p>Vehiculo v1 = new Combi("0809ad",6,true)</p> <p>Vehiculo v2 = new Combi("0206ar",6,true)</p> <p>Empresa.agregarVehiculo(v1);</p> <p>Empresa.agregarVehiculo(v2);</p>	lanza ClienteConPedidoPendienteExeption	2	

		<p>v1 satisface al pedido1</p> <p>v2 satisface al pedido2</p> <p>Empresa.AgregarPedido(pedido1)</p> <p>Empresa.AgregarPedido(pedido2)</p>			
3	<p>pedido1=new Pedido(cliente1,...)</p> <p>pedido2=new Pedido(cliente1,...)</p>	<p>Empresa.agregarCliente(usuario1,1234,juan)</p> <p>Vehiculo v1 = new Combi("0809ad",6,true)</p> <p>Vehiculo v2 = new Combi("0206ar",6,true)</p> <p>v1 satisface al pedido1</p> <p>v2 satisface al pedido2</p> <p>Empresa.agregarVehiculo(v1);</p> <p>Empresa.agregarVehiculo(v2);</p> <p>Empresa.AgregarPedido(pedido1)</p> <p>Empresa.AgregarPedido(pedido2)</p>	no lanza ClienteConPedidoPendienteException	2	
4	<p>pedido1=new pedido(...)</p>	<p>Empresa.agregarCliente(usuario1,1234,juan)</p> <p>Se debe cumplir que:</p>	no lanza SinVehiculoParaPedidoException	2	

		<p>Se tengan vehiculos en el HashMap[String, Vehiculo] vehiculos que NO satisfagan el pedido1</p> <p>o</p> <p>Empresa.getVehiculosDesocupados.size() () == 0</p> <p>Empresa.AgregarPedido(pedido1)</p>			
5	pedido1 =new Pedido(cliente1 ,...)	<p>Empresa.agregarCliente(usuario1,1234,juan)</p> <p>Se debe cumplir que:</p> <p>Se tengan vehiculos en el HashMap[String, Vehiculo] vehiculos vehiculos que NO satisfagan el pedido1</p> <p>o</p> <p>Empresa.getVehiculos.size() == 0</p> <p>Empresa.AgregarPedido(pedido1)</p>	lanza SinVehiculoParaPedidoException	2	
6	pedido1 =new Pedido(cliente1 ,...)	<p>Vehiculo v1 = new Combi("0809ad",6,true)</p> <p>Empresa.agregarVehiculo(v1);</p> <p>v1 satisface al pedido1</p>	lanza clienteNoExisteException	2	

		Empresa.AgregarPedido(pedido1)			
7	pedido1 =new Pedido(cliente1 ,...)	<p>Vehiculo v1 = new Combi("0809ad",6,true)</p> <p>Empresa.agregarVehiculo(v1);</p> <p>v1 satisface al pedido1</p> <p>Empresa.AgregarPedido(pedido1)</p>	no lanza clienteNoExisteException	2	
8	pedido1 =new Pedido(cliente1 ,...)	<p>Empresa.agregarCliente(usuario1,1234,j,uan)</p> <p>Vehiculo v1 = new Combi("0809ad",6,true)</p> <p>Vehiculo v2 = new Combi("0206ar",6,true)</p> <p>Empresa.agregarVehiculo(v1);</p> <p>Empresa.agregarVehiculo(v2);</p> <p>v1 satisface al pedido1</p> <p>v2 satisface al pedido1</p> <p>Empresa.AgregarPedido(pedido1)</p> <p>Empresa.crearViaje(pedido1,...)</p> <p>Empresa.AgregarPedido(pedido1)</p>	Lanza clienteConViajePendienteException	2	

9	pedido1=new Pedido(cliente1,...)	Empresa.agregarCliente(usuario1,1234,juan) Vehiculo v1 = new Combi("0809ad",6,true) Vehiculo v2 = new Combi("0206ar",6,true) Empresa.agregarVehiculo(v1); Empresa.agregarVehiculo(v2); v1 satisface al pedido1 v2 satisface al pedido1 Empresa.AgregarPedido(pedido1) Empresa.crearViaje(pedido1,...) Empresa.AgregarPedido(pedido1)	no Lanza clienteConViajePendienteException	2	
10	pedido1=new Pedido(cliente1,...)		Fail("Se lanzo una excepción que no debería lanzarse")	2	

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

Pre: El parámetro pedido es diferente de null.

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

Parámetros

pedido: Objeto de tipo pedido

Retorna: ArrayList de objetos de tipo Vehículo ordenados de forma descendente de acuerdo al puntaje de cada Vehículo para satisfacer el pedido

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
pedido	pedido == null	no	1
pedido	pedido != null	si	2

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	<p>moto1 = new Moto(...)</p> <p>moto2 = new Moto(...)</p> <p>pedido1=new Pedido(cliente1,...)</p>	<p>Empresa.agregarVehiculo(moto1)</p> <p>Empresa.agregarVehiculo(moto2)</p>	<p>ArrayList<vehículos> vehiculosOrdenadosDes = Empresa.vehiculosOrdenadosPorPedido(pedido1)</p> <p>vehiculosOrdenadosDes.get(0).getPuntajePedido(pedido1) >= vehiculosOrdenadosDes.get(1).getPuntajePedido(pedido1)</p>	2	

Método: public boolean validarPedido(Pedido pedido)

Pre: El parámetro pedido es diferente de null.

Indica si un pedido tiene al menos un vehiculo registrado con las características necesarias para satisfacer el pedido

Parámetros:

pedido: Objeto de tipo Pedido que se evalúa

Retorna: true si existe al menos un vehiculo que satisfaga el pedido, false en caso contrario.

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
pedido	== null	no	1
pedido	!= null	si	2

Batería de pruebas

Número	Datos de	Valor	Salida esperada	Clases	Valor
--------	----------	-------	-----------------	--------	-------

de prueba	entrada				límite
1	pedido	pedido = new Pedido(cliente1, ...)	ningún vehículo satisface el pedido Empresa.validarPedido(pedido) == FALSE	2	
2	pedido	moto1 = new Moto(...) Empresa.agregarVehiculo(moto1) pedido = new Pedido(cliente1, ...)	un vehículo satisface el pedido Empresa.validarPedido(pedido) == TRUE	2	

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

Pre: El atributo vehiculo es distinto de null

Agrega un vehiculo al HashMap vehiculos, si no esta repetido

Parámetros

vehiculo: Vehiculo que se agrega al HashMap

Throws

VehiculoRepetidoException: Se lanza si la patente del parámetro vehiculo coincide con la de algun vehiculo previamente registrado.

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
vehiculo	vehiculo == null	no	1
vehiculo	vehiculo != null	si	2

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	vehiculo	vehiculo=new Auto(...) Auto vehiculo2 = vehiculo	se lanza VehiculoRepetidoExeption	2	

		Empresa.agregarVehiculo(vehiculo) Empresa.agregarVehiculo(vehiculo2)			
2	vehiculo	vehiculo=new Auto(...) Auto vehiculo2 = vehiculo Empresa.agregarVehiculo(vehiculo) Empresa.agregarVehiculo(vehiculo2)	no lanza VehiculoRepetidoExeption	2	
3	vehiculo	vehiculo=new Auto(...) Empresa.agregarVehiculo(vehiculo)	Empresa.getVehiculos.size() == 1	2	

Método: public Iterator<Cliente> iteratorClientes()

Retorna: Iterator de objetos de tipo Cliente del HashMap de clientes

No hay forma de testear este método

Método: public Iterator<Pedido> iteratorPedidos()

Retorna: Iterator de objetos de tipo Pedido del HashMap de pedidos

No hay forma de testear este método

Método: public Iterator<Vehiculo> iteratorVehiculos()

Retorna: Iterator de objetos de tipo Vehiculo del HashMap de vehiculos

No hay forma de testear este método

Método: public Iterator<Chofer> iteratorChoferes()

Retorna: Iterator de objetos de tipo Chofer del HashMap de choferes

No hay forma de testear este método

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

Pre: los parámetros pedido, chofer y vehiculo son distintos de null

Parámetros

pedido: Objeto de tipo Pedido a partir del cual se genera el Viaje

chofer: Objeto de tipo Chofer que se asigna al Viaje

vehiculo: Objeto de tipo Vehiculo que se asigna al Viaje

Throws

PedidoInexistenteException: Se lanza si el pedido pasado como parámetro no pertenece 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 esta realizando un Viaje

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
vehiculo	== null	no	1
vehículo	!= null	si	2
pedido	== null	no	3
pedido	!= null	si	4
chofer	== null	no	5
chofer	!= null	si	6

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
------------------	------------------	-------	-----------------	--------	--------------

1	pedido	pedido1	Empresa.agregarCliente(usuario 1,1234,juan)	2,4,6	
	chofer	chofer1	ArrayList<Chofer> choferes = new ArrayList<Chofer>();		
	vehiculo	auto1	choferes debe tener a chofer1. Empresa.setChoferesDesocupados(choferes); ArrayList<Vehiculo> vehiculos = new ArrayList<Vehiculo>(); vehiculos debe tener a auto1. Empresa.setVehiculosDesocupados(vehiculos); Se debe cumplir que Empresa.validarPedido(pedido1) == true Empresa.crearViaje(pedido1,chofer1,auto1) pedido1 no agregado a la lista de pedidos lanza PedidoInexistenteExeption		
2	pedido	pedido1	Empresa.agregarCliente(usuario 1,1234,juan)	2,4,6	
	chofer	chofer1	ArrayList<Chofer> choferes = new ArrayList<Chofer>();		
	vehiculo	auto1	choferes debe tener a chofer1. Empresa.setChoferesDesocupados(choferes); ArrayList<Vehiculo> vehiculos = new ArrayList<Vehiculo>(); vehiculos debe tener a auto1. Empresa.setVehiculosDesocupados(vehiculos);		

			<p>Se debe cumplir que Empresa.validarPedido(pedido1) == true</p> <p>Empresa.crearViaje(pedido1,chofer1,auto1)</p> <p>pedido1 no agregado a la lista de pedidos</p> <p>no lanza PedidoInexistenteExeption</p>		
3	pedido	pedido1	<p>Empresa.agregarCliente(usuario1,1234,juan)</p>	2,4,6	
	chofer	chofer1	<p>HashMap<Cliente,Pedido> pedidos = new HashMap<Cliente,Pedido>();</p>		
	vehiculo	auto1	<p>pedidos debe tener el pedido1.</p> <p>Empresa.setPedidos(pedidos)</p> <p>ArrayList<Vehiculo> vehiculos = new ArrayList<Vehiculo>();</p> <p>vehiculos debe tener a auto1.</p> <p>Empresa.setVehiculosDesocupados(vehiculos);</p> <p>Se debe cumplir que Empresa.validarPedido(pedido1) == true</p> <p>Empresa.crearViaje(pedido1,chofer1,auto1)</p> <p>chofe1r no pertenece a la lista de choferesDisponibles</p> <p>Lanza ChoferNoDisponibleExeption</p>		
4	pedido	pedido1	<p>Empresa.agregarCliente(usuario1,1234,juan)</p>	2,4,6	
	chofer	chofer1			

	vehiculo	auto1	<p>HashMap<Cliente,Pedido> pedidos = new HashMap<Cliente,Pedido>(); pedidos debe tener el pedido1. Empresa.setPedidos(pedidos)</p> <p>ArrayList<Vehiculo> vehiculos = new ArrayList<Vehiculo>(); vehiculos debe tener a auto1. Empresa.setVehiculosDesocupa dos(vehiculos);</p> <p>Se debe cumplir que Empresa.validarPedido(pedido1) == true</p> <p>Empresa.crearViaje(pedido1,cho fer1,auto1)</p> <p>chofer1 no pertenece a la lista de choferesDisponibles</p> <p>no Lanza ChoferNoDisponibleExeption</p>		
5	pedido	pedido1	<p>Empresa.agregarCliente(usuario 1,1234,juan)</p> <p>HashMap<Cliente,Pedido> pedidos = new HashMap<Cliente,Pedido>(); pedidos debe tener el pedido1. Empresa.setPedidos(pedidos)</p> <p>ArrayList<Chofer> choferes = new ArrayList<Chofer>(); choferes debe tener a chofer1. Empresa.setChoferesDesocupa dos(choferes);</p>	2,4,6	
	chofer	chofer1			
	vehiculo	auto1			

			<p>Empresa.crearViaje(pedido1,chofer1,auto1)</p> <p>auto1 no pertenece a la lista de vehiculos disponibles</p> <p>lanza vehiculoNoDisponibleExeption</p>		
6	pedido	pedido1	Empresa.agregarCliente(usuario1 ,1234, juan)	2,4,6	
	chofer	chofer1	HashMap<Cliente,Pedido> pedidos = new HashMap<Cliente,Pedido>();		
	vehiculo	auto1	<p>pedidos debe tener el pedido1.</p> <p>Empresa.setPedidos(pedidos)</p> <p>ArrayList<Chofer> choferes = new ArrayList<Chofer>();</p> <p>choferes debe tener a chofer1.</p> <p>Empresa.setChoferesDesocupados(choferes);</p> <p>Empresa.crearViaje(pedido1,chofer1,auto1)</p> <p>auto1 no pertenece a la lista de vehiculos disponibles</p> <p>no lanza vehiculoNoDisponibleExeption</p>		
7	pedido	pedido1	Empresa.agregarCliente(usuario1 ,1234, juan)	2,4,6	
	chofer	chofer1	HashMap<Cliente,Pedido> pedidos = new HashMap<Cliente,Pedido>();		
	vehiculo	auto1	pedidos debe tener el pedido1 .		

			<p>Empresa.setPedidos(pedidos)</p> <p>ArrayList<Chofer> choferes = new ArrayList<Chofer>();</p> <p>choferes debe tener a chofer1.</p> <p>Empresa.setChoferesDesocupados(choferes);</p> <p>ArrayList<Vehiculo> vehiculos = new ArrayList<Vehiculo>();</p> <p>vehiculos debe tener a auto1.</p> <p>Empresa.setVehiculosDesocupados(vehiculos);</p> <p>Se debe cumplir que Empresa.validarPedido(pedido1) == false (auto1 NO satisface a pedido1)</p> <p>Empresa.crearViaje(pedido1,chofer1,auto1)</p> <p>lanza vehiculoNoValidoExeption</p>		
8	pedido	pedido1	<p>Empresa.agregarCliente(usuario1,1234,juan)</p> <p>HashMap<Cliente,Pedido> pedidos = new HashMap<Cliente,Pedido>();</p> <p>pedidos debe tener el pedido1.</p> <p>Empresa.setPedidos(pedidos)</p> <p>ArrayList<Chofer> choferes = new ArrayList<Chofer>();</p> <p>choferes debe tener a chofer1.</p> <p>Empresa.setChoferesDesocupados(choferes);</p>	2,4,6	
	chofer	chofer1			
	vehiculo	auto1			

			<p>ArrayList<Vehiculo> vehiculos = new ArrayList<Vehiculo>();</p> <p>vehiculos debe tener a auto1.</p> <p>Empresa.setVehiculosDesocupados(vehiculos);</p> <p>Se debe cumplir que Empresa.validarPedido(pedido1) == false (auto1 NO satisface a pedido1)</p> <p>Empresa.crearViaje(pedido1, chofer1, auto1)</p> <p>no lanza vehiculoNoValidoException</p>		
9	pedido	pedido1	Empresa.agregarCliente(usuario1 , 1234, juan)	2,4,6	
	chofer	chofer1	HashMap<Cliente, Pedido> pedidos = new HashMap<Cliente, Pedido>();		
	vehiculo	auto1	<p>pedidos debe tener a pedido1 y pedido2.</p> <p>pedido1 y pedido2 tienen el mismo cliente</p> <p>Empresa.setPedidos(pedidos)</p> <p>ArrayList<Chofer> choferes = new ArrayList<Chofer>();</p> <p>choferes debe tener a chofer1 y chofer2.</p> <p>Empresa.setChoferesDesocupados(choferes);</p>		

			<p>ArrayList<Vehiculo> vehiculos = new ArrayList<Vehiculo>();</p> <p>vehiculos debe tener a auto1 y auto2.</p> <p>Empresa.setVehiculosDesocupados(vehiculos);</p> <p>Se debe cumplir que Empresa.validarPedido(pedido1) == true</p> <p>Empresa.crearViaje(pedido1, chofer1, auto1)</p> <p>Se debe cumplir que Empresa.validarPedido(pedido2) == true</p> <p>Empresa.crearViaje(pedido2, chofer2, auto2)</p> <p>no lanza ClienteConViajePendienteException</p>		
10	pedido	pedido1	Empresa.agregarCliente(usuario1 , 1234 , juan)	2,4,6	
	chofer	chofer1	HashMap<Cliente, Pedido> pedidos = new HashMap<Cliente, Pedido>();		
	vehiculo	auto1	<p>pedidos debe tener a pedido1 y pedido2.</p> <p>pedido1 y pedido2 tienen el mismo cliente</p> <p>Empresa.setPedidos(pedidos)</p> <p>ArrayList<Chofer> choferes = new ArrayList<Chofer>();</p> <p>choferes debe tener a chofer1 y chofer2.</p> <p>Empresa.setChoferesDesocupados(choferes);</p>		

			<p>ArrayList<Vehiculo> vehiculos = new ArrayList<Vehiculo>();</p> <p>vehiculos debe tener a auto1 y auto2.</p> <p>Empresa.setVehiculosDesocupados(vehiculos);</p> <p>Se debe cumplir que Empresa.validarPedido(pedido1) == true</p> <p>Empresa.crearViaje(pedido1, chofer1, auto1)</p> <p>Se debe cumplir que Empresa.validarPedido(pedido2) == true</p> <p>Empresa.crearViaje(pedido2, chofer2, auto2)</p> <p>lanza ClienteConViajePendienteException</p>		
11	pedido	pedido1	Empresa.agregarCliente(usuario1 , 1234 , juan)	2,4,6	
	chofer	chofer1	HashMap<Cliente, Pedido> pedidos = new HashMap<Cliente, Pedido>();		
	vehiculo	auto1	<p>pedidos debe tener el pedido1.</p> <p>Empresa.setPedidos(pedidos)</p> <p>ArrayList<Chofer> choferes = new ArrayList<Chofer>();</p> <p>choferes debe tener a chofer1.</p> <p>Empresa.setChoferesDesocupados(choferes);</p>		

			<p>ArrayList<Vehiculo> vehiculos = new ArrayList<Vehiculo>();</p> <p>vehiculos debe tener a auto1.</p> <p>Empresa.setVehiculosDesocupados(vehiculos);</p> <p>Se debe cumplir que Empresa.validarPedido(pedido1) == true</p> <p>Empresa.crearViaje(pedido1, chofer1, auto1)</p> <p>se lanza una excepción no contemplada en los casos de prueba</p>		
12	pedido	pedido1	Empresa.agregarCliente(usuario1 , 1234 , juan)	2,4,6	
	chofer	chofer1	HashMap<Cliente, Pedido> pedidos = new HashMap<Cliente, Pedido>();		
	vehiculo	auto1	<p>pedidos debe tener el pedido1.</p> <p>Empresa.setPedidos(pedidos)</p> <p>ArrayList<Chofer> choferes = new ArrayList<Chofer>();</p> <p>choferes debe tener a chofer1.</p> <p>Empresa.setChoferesDesocupados(choferes);</p> <p>ArrayList<Vehiculo> vehiculos = new ArrayList<Vehiculo>();</p> <p>vehiculos debe tener a auto1.</p> <p>Empresa.setVehiculosDesocupados(vehiculos);</p> <p>Se debe cumplir que Empresa.validarPedido(pedido1) == true</p>		

			Empresa.crearViaje(pedido1 , ch ofer1 , auto1) Se crea el viaje y se almacena en HashMap[Ciudad, Viaje] viajesIniciados ; entonces, Empresa.getViajesIniciados().siz e == 1		
--	--	--	--	--	--

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

Califica y termina un Viaje Pendiente del Cliente Logueado

Pre: Hay un usuario de tipo Cliente logueado en la Empresa, calificacion está comprendido entre 0 y 5 inclusive

Parámetros

calificacion: parámetro de tipo int para calificar el viaje pendiente del cliente logueado

Throws

`ClienteSinViajePendienteException`: Se lanza si el Cliente no esta realizando un viaje.

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
calificacion	calificacion<0 calificacion>5	no	1
calificacion	0<=calificacion && calificacion<=5	si	2

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	calificacion	3	Empresa.agregarCliente(usuario1 , 1234 , juan) HashMap<Cliente,Pedido> pedidos = new HashMap<Cliente,Pedido>(); pedidos debe tener el pedido1 . Empresa.setPedidos(pedidos) ArrayList<Chofer> choferes = new ArrayList<Chofer>();	2	

			<p>choferes debe tener a chofer1.</p> <p><code>Empresa.setChoferesDesocupados(choferes);</code></p> <p><code>ArrayList<Vehiculo> vehiculos = new ArrayList<Vehiculo>();</code></p> <p>vehiculos debe tener a auto1.</p> <p><code>Empresa.setVehiculosDesocupados(vehiculos);</code></p> <p>Se debe cumplir que <code>Empresa.validarPedido(pedido1) == true</code></p> <p><code>Empresa.crearViaje(pedido1,chofer1,auto1)</code></p> <p><code>Empresa.pagarYfinalizarViaje(calificacion)</code></p> <p>Se finaliza y califica un viaje pendiente y se almacena en <code>ArrayList<Viaje> viajesTerminados</code>; entonces, <code>Empresa.getViajesIniciados().size == 0</code> <code>Empresa.getViajesTerminados().get(0) = viaje(pedido1,chofer1,auto1)</code> verificando que <code>Empresa.getViajesTerminados().get(0).getCalificacion() == viaje.getCalificacion()</code></p>		
2	calificacion	3	<p><code>Empresa.agregarCliente(usuario1,1234,juan)</code></p> <p><code>HashMap<Cliente,Pedido> pedidos = new HashMap<Cliente,Pedido>();</code></p> <p>pedidos debe tener el pedido1.</p> <p><code>Empresa.setPedidos(pedidos)</code></p> <p><code>ArrayList<Chofer> choferes = new ArrayList<Chofer>();</code></p> <p>choferes debe tener a chofer1.</p>	2	

			<p>Empresa.setChoferesDesocupados(choferes);</p> <p>ArrayList<Vehiculo> vehiculos = new ArrayList<Vehiculo>();</p> <p>vehiculos debe tener a auto1.</p> <p>Empresa.setVehiculosDesocupados(vehiculos);</p> <p>Se debe cumplir que Empresa.validarPedido(pedido1) == true</p> <p>Empresa.pagarYfinalizarViaje(calificacion)</p> <p>lanza ClientesinViajesPendienteExeption</p>		
3	calificacion	3	<p>Empresa.agregarCliente(usuario1,1234,juan)</p> <p>HashMap<Cliente,Pedido> pedidos = new HashMap<Cliente,Pedido>();</p> <p>pedidos debe tener el pedido1.</p> <p>Empresa.setPedidos(pedidos)</p> <p>ArrayList<Chofer> choferes = new ArrayList<Chofer>();</p> <p>choferes debe tener a chofer1.</p> <p>Empresa.setChoferesDesocupados(choferes);</p> <p>ArrayList<Vehiculo> vehiculos = new ArrayList<Vehiculo>();</p> <p>vehiculos debe tener a auto1.</p> <p>Empresa.setVehiculosDesocupados(vehiculos);</p> <p>Se debe cumplir que Empresa.validarPedido(pedido1) == true</p>	2	

			Empresa.pagarYfinalizarViaje(calificacion)		
			No lanza ClienteSinViajesPendienteExeption		
4	calificacion	3	se lanza una excepción no contemplada en los casos de prueba, ej: ClienteNoExisteException	2	

Método: `public Iterator<Viaje> iteratorViajesIniciados()`

No hay forma de testear este método

Método: `public ArrayList<Viaje> iteratorViajesTerminados()`

Retorna: Iterator de objetos de tipo Viaje del ArrayList de viajesTerminados

No hay forma de testear este método

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

Pre: los parámetros usserName y pass son distintos de null y tienen al menos un caracter

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

Parámetros

usserName: Nombre de usuario para logearse

pass: Password del usuario para logearse

Retorna: Objeto de tipo Usuario que se ha logeado al sistema

Throws

UsuarioNoExisteException: Se lanza si no existe ningun Usuario (Administrador o Cliente) registrado con ese nombre de usuario

PasswordErroneaException: Se lanza si el password del usuario es incorrecto

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
userName y pass	== null	no	1
userName y pass	!= null	si	2
userName	userName != ""	si	3
pass	pass != ""	si	4
pass	pass == ""	no	5
userName	userName == ""	no	6

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	userName	"usuario1"	Empresa.agregarCliente(usuario1,1234,juan) Cliente u = Empresa.login(userName,pass) u.getNombreUsuario() == usuario1 u.getPass() == 1234 u.getNombreReal() == juan	2,3,4	
	pass	"1234"			
2	userName	"admin"	Administrador admin = (Administrador)Empresa.login(userName,pass) se debe cumplir que: Empresa.isAdmin() == true Administrador.getInstance() == admin	2,3,4	
	pass	"admin"			
3	userName	"usuario1"	se debe cumplir que Empresa.isAdmin() == false y que no haya clientes logueados usuario u = Empresa.login(userName,pass) No lanza usuarioNoExistenteExeption	2,3,4	
	pass	"1234"			
4	userName	"usuario1"	se debe cumplir que Empresa.isAdmin() == false y que no haya clientes logueados usuario u = Empresa.login(userName,pass)	2,3,4	
	pass	"1234"			

			lanza UsuarioNoExistenteExeption		
5	userName	"usuario1"	Empresa.agregarCliente(usuario1, 1234, juan)	2,3,4	
	pass	"45"	Cliente u = Empresa.login(userName,pass) no Lanza PasswordIncorrectaExeption		
6	userName	"usuario1"	Empresa.agregarCliente(usuario1, 1234, juan)	2,3,4	
	pass	"45"	Cliente u = Empresa.login(userName,pass) Lanza PasswordIncorrectaExeption		
7	userName	"admin"	Empresa.setUsuarioLogeado(Administrador.getInstance())	2,3,4	
	pass	"45"	se debe cumplir que Empresa.isAdmin() == true Administrador ad = Empresa.login(userName,pass) Lanza PassWordIncorrectaExeption		
8	userName	"admin"	Empresa.setUsuarioLogeado(Administrador.getInstance())	2,3,4	
	pass	"45"	se debe cumplir que Empresa.isAdmin() == true Administrador ad = Empresa.login(userName,pass) no Lanza PasswordIncorrectaExeption		
9	userName	"usuario1"	se lanza una excepción no contemplada en los casos de prueba	2,3,4	
	pass	"45"			

Método: `public double getTotalSalarios()`

Retorna: un 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	Valor límite
1	<code>empresa1 = Empresa.getInstance()</code> <code>choferT1=new choferTemporario();</code>	<code>chofer1.setSueldoBasico(450.0)</code> <code>empresa1.AgregarChofer(chofer1)</code>	<code>empresa1.getTotalSalarios() == 450.0</code>		

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

Pre: el parámetro cliente es distinto de null

Parámetros

cliente: Objeto de tipo Cliente

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

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
cliente	<code>== null</code>	no	1
cliente	<code>!= null</code>	si	2

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	<code>empresa1 = Empresa.getInstance()</code>		<code>historialViajeCliente1 = empresa1.getHistorialViajeCliente()</code> <code>historialViajeCliente1.size == 0</code>	2	

	cliente6=new Cliente(...);				
2	empresa1 = Empresa.getInstance() cliente5=new Cliente(...); pedido5=new Pedido(cliente 5.. viaje5=new Viaje(pedido5 ...)		empresa1.getHistorialViajeCliente () == arrayList<viajes>=viajesTerminado s1[cliente5]	2	

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

Pre: el parámetro chofer es distinto de null

Parámetros

chofer: Objeto de tipo Chofer

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

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
chofer	== null	no	1
chofer	!= null	si	2

Batería de pruebas

Número	Datos de entrada	Valor	Salida esperada	Clase	Valor
--------	------------------	-------	-----------------	-------	-------

de prueba				s	límite
1	<pre> empresa2 = Empresa.getInstance() chofer7=new choferTemporario() cliente5=new Cliente(...); pedido5=new Pedido(cliente5..) viaje5=new Viaje(pedido5...) </pre>	<pre> empresa2.crearViaje (pedido5,chofer7) </pre>	<pre> empresa2.getHistorialDeViajeCh ofer(chofer7) == arrayList<viajes>=viaje1[chofer7] </pre>	2	
2	<pre> chofer6=new choferTemporario() chofer8=new choferTemporario() </pre>		<pre> historialViajeChofer1 = empresa2.getHistorialViajeChof er() historialViajeChofer1.size == 0 </pre>	2	

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

Pre: El parámetro chofer es distinto de null

Parámetros

chofer: objeto de tipo Chofer

Retorna: el promedio de las calificaciones de los viajes realizados por el chofer en cuestion

Throws:

SinViajesException: se Lanza si el chofer no tiene ningun viaje realizado.

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
chofer	== null	no	1
chofer	!= null	si	2

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	chofer1 viaje1=new viaje(chofer1)	chofer1 != null	0.0<=empresa.calificacionDeChofer(chofer1)<=5.0	2	
2	chofer2	chofer2 != null	no lanza SinviajesExeption	2	
3	chofer2	chofer2 != null	SinviajesExeption	2	

Método: public Pedido getPedidoDeCliente(Cliente cliente)

Pre: El parámetro cliente es distinto de null

Parameters

cliente: Objeto de tipo Cliente

Retorna: El Pedido realizado por el cliente. Si el cliente no tiene ningun pedido pendiente se retorna null

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
cliente	== null	no	1
cliente	!= null	si	2

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
------------------	------------------	-------	-----------------	--------	--------------

1	cliente1 Pedido1=new Pedido(cliente1...)	cliente1 != null Empresa.AgregarPedido(Pedido1)	empresa.getPedidoDeCliente(cliente1) == Pedido1	2	
2	cliente1	cliente1 != null	empresa.getPedidoDeCliente(cliente1) == null	2	

Método: public Viaje getViajeDeCliente(Cliente cliente)

Pre: El Parámetro cliente es distinto de null

Parámetros

cliente: Objeto de tipo Cliente

Retorna: El Viaje no terminado que esta realizando el cliente. Si el cliente no esta realizando un viaje se retorna null

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
cliente	== null	no	1
cliente	!= null	si	2

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	cliente1 != null cliente = cliente1 Pedido1=new Pedido(cliente1) chofer1=new choferPermanente(...)	empresa.crearViajes(Pedido1,chofer1,...)	empresa.getViajeDeCliente(cliente1) != null	2	

2	cliente1 != null	!= null	GetViajesDeClientes(cliente) == null	2	
---	------------------	---------	--------------------------------------	---	--

Método: public Usuario getUsuarioLogeado()

Retorna: Objeto de tipo Usuario (Administrador o Cliente) logueado al sistema. Si no hay un usuario logeado se retorna null

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	Administrador admin.getInstance()	empresa.setUsuarioLogeado(admin)	empresa.getUsuarioLogeado() == admin		
2	Cliente cliente1= new Cliente(Juan95,1245,juan)	empresa.setUsuarioLogeado(cliente1)	empresa.getUsuarioLogeado() == cliente1		
3			empresa.getUsuarioLogeado() == null		

Método: public void logout()

Cierra la sesión del usuario actual

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1		Administrador admin = (Administrador) Empresa.getInstance().login("admin","admin")	empresa.logout() empresa.isAdmin() == false		

2		<i>Cliente cliente1 = (Cliente)Empresa.getInstance().login("Juan95", "1245")</i>	<i>empresa.logout()</i> <i>empresa.getUsuarioLogeado() == null</i>		
---	--	--	---	--	--

Método: *public boolean isAdmin()*

Retorna: true si el Administrador está logueado, false en caso contrario.

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	Administrador <i>admin.getInstance()</i>	<i>empresa.setUsuarioLogeado(admin)</i>	<i>empresa.isAdmin() == true</i>		
2	Administrador <i>admin.getInstance()</i>		<i>empresa.isAdmin() == false</i>		

Método: *public void setUsuarioLogeado(Usuario usuarioLogeado)*

Parámetros

usuarioLogeado: Setter del atributo usuarioLogeado

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
usuarioLogeado	== null	si	1

usuarioLogeado	!= null	si	2
----------------	---------	----	---

Batería de pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases	Valor límite
1	Admin usuarioLogeado	<i>usuarioLogeado</i> .getInstance()	<i>empresa.getUsuarioLogeado() == usuarioLogeado</i>		
2	Cliente usuarioLogeado	<i>usuarioLogeado</i> = new Cliente(Juan95, 1245,juan)	<i>empresa.getUsuarioLogeado() == usuarioLogeado</i>		
3	usuarioLogeado = null		<i>empresa.getUsuarioLogeado() == null</i>		

Test de integración

Clase: Controlador

Controla el Modelo y la Vista. Recibe las acciones de un objeto de tipo IVista interactuando con la Clase Empresa (Singleton). Solicita a la IVista la actualización de los datos.

Implementa la interfaz ActionListener

Invariante de Clase

El atributo **vista** (de tipo IVista) es siempre diferente de null

El atributo **persistencia** (de tipo IPersistencia) es siempre diferente de null

El atributo **fileName** (de tipo String) es distinto de null

Casos de uso:

Método: public Controlador()

Crea una instancia de Controlador.

Por defecto inicializa el atributo vista (de tipo IVista) con un objeto de tipo Ventana y la hace visible

Por defecto inicializa el atributo persistencia (de tipo IPersistencia) con un objeto de tipo PersistenciaBIN

Por defecto inicializa el atributo fileName con el nombre "empresa.bin"

Casos de prueba:

Entrada	Condiciones de entrada (estados de las clases invocadas)	Salida esperada	Condiciones de salida
	Controlador cont = new Controlador()	cont.getVista() != null cont.getPersistencia() != null cont.getFileName() == "empresa.bin"	

Método: public void leer()

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

Si la lectura no se puede realizar por cualquier motivo, entonces se delega en el atributo **vista** mostrar un mensaje correspondiente a la excepción lanzada y se intenta escribir un archivo nuevo.

Casos de prueba:

Entrada	Condiciones de entrada (estados de las clases invocadas)	Salida esperada	Condiciones de salida
	El archivo indicado por el atributo fileName == "empresa.bin" NO existe; y Clase Ventana presente.	File arch = new File("empresa.bin"); if(!arch.exists()) fail()	

	<pre>File arch = new File("empresa.bin"); if(arch.exists()) arch.delete(); Controlador cont = new controlador() cont.leer()</pre>		
	El archivo indicado por el atributo fileName == "empresa.bin" NO existe; y Clase Ventana ausente.	Falla de ejecución.	
	Clase PersistenciaBIN ausente y Clase Ventana ausente.	Falla de ejecución.	
	<p>El archivo indicado por el atributo fileName == "empresa.bin" existe, Clase Ventana presente y Clase PersistenciaBIN presente.</p> <p>Controlador cont = new controlador();</p> <p>Cliente cli1 = null; Cliente cli2 = null;</p> <p>Empresa.getInstance().agregarCliente("usuario1", "1234", "juan");</p> <p>cli2 = (Cliente) Empresa.getInstance().login("usuario1", "1234");</p> <p>controlador.escribir(); controlador.leer();</p> <p>cli1 = (Cliente) Empresa.getInstance().login("usuario1", "1234");</p>	<p>cli1 == cli2</p> <p>Recupera los datos de la clase Empresa (singleton), conservados en el archivo indicado por el atributo fileName delegando la implementación en el atributo persistencia</p>	

Método: `public void escribir()`

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

Si la escritura no se puede realizar por cualquier motivo, entonces se delega en el atributo **vista** mostrar el mensaje correspondiente a la excepción lanzada

Casos de prueba:

Entrada	Condiciones de entrada (estados de las clases invocadas)	Salida esperada	Condiciones de salida
	<p>Clase Ventana activa y Clase PersistenciaBIN presente. () (El caso de que el archivo "empresa.bin" exista o no, se contempla en este caso)</p> <p>Controlador cont = new controlador();</p> <p>Cliente cli1 = null; Cliente cli2 = null;</p> <p>Empresa.getInstance().agregarCliente("usuario1", "1234", "juan");</p> <p>cli2 = (Cliente) Empresa.getInstance().login("usuario1", "1234");</p> <p>controlador.escribir(); controlador.leer();</p> <p>cli1 = (Cliente) Empresa.getInstance().login("usuario1", "1234");</p>	<p>(Puede realizar la persistencia del archivo "empresa.bin")</p> <p>cli1 == cli2</p>	

	Clase Ventana desactiva Clase PersistenciaBIN presente	Falla de ejecución	
	Clase vista Activa Clase PersistenciaBIN Ausente	Falla de ejecución	

Método: public IVista getView()

Retorna: atributo vista

Entrada	Condiciones de entrada (estados de las clases invocadas)	Salida esperada	Condiciones de salida
	Clase Ventana desactiva	Controlador cont= new Controlador() Falla de ejecución	
	Clase Ventana activa IVista vista = new IVista();	Controlador cont; cont.setVista(vista); cont.getView() == vista	

Método: public void setVista(IVista vista)

Pre: El parámetro es diferente de null

Parámetros

vista: de tipo IVista.

Casos de prueba:

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
vista	!= null	si	1

vista	== null	no	2
-------	---------	----	---

Entrada	Condiciones de entrada (estados de las clases invocadas)	Salida esperada	Condiciones de salida
vista	vista != null (1) IVista vista = new IVista() ()	Controlador cont; cont.setVista(vista); cont.getVista() == vista	

Método: `public IPersistencia getPersistencia()`

Retorna: atributo persistencia

Casos de prueba:

Entrada	Condiciones de entrada (estados de las clases invocadas)	Salida esperada	Condiciones de salida
	Clase PersistenciaBIN ausente	Controlador cont = new Controlador() Falla de ejecución.	
	Clase PersistenciaBIN presente	Controlador cont; cont.setPersistencia(persistencia); cont.getPersistencia() == persistencia;	

Método: `public void setPersistencia(IPersistencia persistencia)`

Pre: El parámetro es diferente de null

Parametros

persistencia: de tipo IPersistencia.

Casos de prueba:

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
persistencia	persistencia != null	si	1
persistencia	persistencia == null	no	2

Entrada	Condiciones de entrada (estados de las clases invocadas)	Salida esperada	Condiciones de salida
---------	--	-----------------	-----------------------

IPersistencia persistencia = new IPersistencia()		Controlador cont; cont.setPersistencia(persistencia); cont.getPersistencia() == persistencia;	
--	--	--	--

Método: `public String getFileName()`

Retorna: atributo fileName

Casos de prueba:

Entrada	Condiciones de entrada (estados de las clases invocadas)	Salida esperada	Condiciones de salida
		Controlador cont = new Controlador() cont.getFileName() == "empresa.bin"	

Metodo: public void setFileName(String fileName)

Pre: El parámetro es diferente de null

Parámetros:

fileName: de tipo String

Casos de prueba:

Tabla de particiones en clases de equivalencia

Datos de entrada	Descripción de la clase de equivalencia	Aplica? (cumple el contrato)	Identificador de clase de
fileName	fileName != null	si	1
fileName	fileName == null	no	2

Entrada	Condiciones de entrada (estados de las clases invocadas)	Salida esperada	Condiciones de salida
String fileName = "empresa.bin"		Controlador cont = new Controlador() cont.getFileName() == fileName;	

Método: public void actionPerformed(ActionEvent e)

Método de la interfaz ActionListener. Dependiendo el valor del actionCommand del parámetro de tipo ActionEvent se invoca a los siguientes metodos:

Constantes.CERRAR_SESION_CLIENTE or Constantes.CERRAR_SESION_ADMIN se invoca a logout()

Constantes.LOGIN se invoca a login()

Constantes.REG_BUTTON_REGISTRAR se invoca a registrar()

Constantes.NUEVO_PEDIDO se invoca a nuevoPedido()

Constantes.CALIFICAR_PAGAR se invoca a calificarPagar()

Constantes.NUEVO_CHOFER se invoca a nuevoChofer()

Constantes.NUEVO_VEHICULO se invoca a nuevoVehiculo()

Constantes.NUEVO_VIAJE se invoca a nuevoViaje()

Cualquier otro valor es ignorado.

Specified by:

actionPerformed in interface [ActionListener](#)

Casos de prueba:

Entrada	Condiciones de entrada	Salida esperada	Condiciones de salida
ActionEvent e = new ActionEvent(null, 0, Constantes.CERRAR_SESION_CLIENTE)	Clase Cliente y Empresa presente, con: <i>Empresa.getInstance().agregarCliente("Juan95", "1245", "juan");</i> Cliente cliente1 = (Cliente) <i>Empresa.getInstance().login("Juan95", "1245")</i> File arch = new File("empresa.bin"); if(arch.exists()) arch.delete(); Controlador cont = new	<i>empresa.getUsuarioLogeado() == null</i> arch.exists() == true cont.leer() Cliente c2 = (Cliente) <i>Empresa.getInstance().login("Juan95", "1245")</i> <i>cliente1 == c2</i>	

	controlador() cont.actionPerformed(e);		
ActionEvent e = new ActionEvent(null, 0, Constantes.CERRAR_SESION_ADMIN)	Clase Administrador y Empresa presente, con: <i>Empresa.getInstance().agregarCliente("Juan95", "1245", "juan");</i> Cliente cliente1 = (Cliente) <i>Empresa.getInstance().login("Juan95", "1245")</i> File arch = new File("empresa.bin"); if(arch.exists()) arch.delete(); Controlador cont = new controlador() cont.actionPerformed(e);	<i>empresa.getUsuarioLogeado() == null</i> arch.exists() == true cont.leer() Cliente c2 = (Cliente) <i>Empresa.getInstance().login("Juan95", "1245")</i> <i>cliente1 == c2</i>	
ActionEvent e = new ActionEvent(null, 0, Constantes.LOGIN)	Clase Ventana presente, Clase Empresa presente, existe vista.getUssetName() (Cliente) y vista.getPassword() es correcta. Cliente cli = new Cliente("usuario1", "1234", "juan"); Empresa.agregarCliente(usuario1, 1234, juan) IVista ventana = mock(IVista.class); Controlador cont = new Controlador();	Realiza el logeo de un Usuario (Cliente) al sistema. cli == Empresa.getInstance().getUsuarioLogeado();	

	<pre> cont.setVista(ventana); when(ventana.getUsser Name()).thenReturn("us uario1"); when(ventana.getPass word()).thenReturn("12 34"); cont.actionPerformed(e); </pre>		
ActionEvent e = new ActionEvent(null, 0, Constantes.REG_BUT TON_REGISTRAR)	<p>Clase Ventana presente y "pass" y "confirm" coinciden; pero NO existe el usuario a registrar:</p> <pre> IVista ventana = mock(IVista.class); Controlador cont = new Controlador(); cont.setVista(ventana); when(ventana.getRegN ombreReal()).thenRetur n("Juan"); when(ventana.getRegU sserName()).thenRetur n("Juan95"); when(ventana.getRegP assword()).thenReturn(" 45"); when(ventana.getRegC onfirmPassword()).then Return("45"); cont.actionPerformed(e); </pre>	<pre> Empresa.getInstance().get Clientes().size() == 1 </pre>	
ActionEvent e = new ActionEvent(null, 0, Constantes.NUEVO_P EDIDO)	<p>Clase Ventana presente y Clase Empresa presente.</p>	<pre> Empresa.getPedidos().size () == 1 Se actualiza la empresa </pre>	

	<p>(Se puede agregar un nuevo pedido a la empresa)</p> <pre> Cliente cli = new Cliente (usuario1,1234,juan); Empresa.setUsuarioLo geado(cli); Vehiculo v1 = new Combi("0809ad",6,true) Empresa.agregarVehic ulo(v1); v1 satisface al pedido generado por la vista IVista ventana = mock(IVista.class); Controlador cont = new Controlador(); cont.setVista(ventana); when(ventana.getCanti dadPax()).thenReturn(3); when(ventana.isPedido ConMascota()).thenRet urn(true); when(ventana.getPlaza s()).thenReturn(8); when(ventana.isPedido ConBaul()).thenReturn(false); when(ventana.getCant Km()).thenReturn(100); when(ventana.getTipoZ ona()).thenReturn(Cons tantes.ZONA_PELIGR </pre>	con la información nueva.	
--	--	---------------------------	--

	OSA); cont.actionPerformed(e);		
ActionEvent e = new ActionEvent(null, 0, Constantes.CALIFICA R_PAGAR)	<p>Clase Ventana presente y Clase Empresa presente. (El Cliente ESTA realizando un viaje)</p> <p>IVista ventana = mock(IVista.class);</p> <p>Controlador cont = new Controlador(); cont.setVista(ventana);</p> <p>when(ventana.getCalifi cacion()).thenReturn(4);</p> <p>cont.actionPerformed(e);</p>	<p>Se finaliza y califica un viaje pendiente y se almacena en ArrayList[Viaje] viajesTerminados;</p> <p>entonces,</p> <p>Empresa.getViajesIniciado s().size == 0 Empresa.getViajesTermina dos().get(0) == viaje(pedido1,chofer1,auto 1) verificando que Empresa.getViajesTermina dos().get(0).getCalificacion () == viaje.getCalificacion()</p> <p>Se actualiza la empresa con la información nueva..</p>	
ActionEvent e = new ActionEvent(null, 0, Constantes.NUEVO_C HOFER)	<p>Clase Ventana presente y Clase Empresa presente. (La empresa NO tiene ningún chofer registrado y agrega un chofer PERMANENTE)</p> <p>IVista ventana = mock(IVista.class);</p> <p>Controlador cont = new Controlador(); cont.setVista(ventana);</p> <p>when(ventana.getTipoC hofer()).thenReturn(Co nstantes.PERMANENT</p>	<p>Chofer agregado al HashMap</p> <p>Empresa.getInstance().get Choferes().size() == 1</p> <p>Se actualiza la empresa con la información nueva.</p>	

	<pre> E); when(ventana.getDNIChofer()).thenReturn("43666918"); when(ventana.getNombreChofer()).thenReturn("Jose"); when(ventana.getAnioChofer()).thenReturn(2000); when(ventana.getHijosChofer()).thenReturn(2); cont.actionPerformed(e); </pre>		
<pre> ActionEvent e = new ActionEvent(null, 0, Constantes.Constante s.NUEVO_VEHICULO) </pre>	<pre> Clase Ventana presente y Clase Empresa presente. (La empresa NO tiene vehiculos agregados y agrega un AUTO) IVista ventana = mock(IVista.class); Controlador cont = new Controlador(); cont.setVista(ventana); when(ventana.getTipoVehiculo()).thenReturn(Constantes.AUTO); when(ventana.getPatente()).thenReturn("532"); when(ventana.getPlazas()).thenReturn(3); when(ventana.isVehicul </pre>	<pre> Empresa.getVehiculos.size() == 1 Se actualiza la empresa con la información nueva. </pre>	

	<pre>oAptoMascota()).thenReturn(true); cont.nuevoViaje(); cont.actionPerformed(e);</pre>		
ActionEvent e = new ActionEvent(null, 0, Constantes.NUEVO_VIAJE)	<p>Clase Ventana presente y Clase Empresa presente. (Se deben dar las condiciones para generar un viaje con los 3 datos recibidos por la ventana) IVista ventana = mock(IVista.class);</p> <p>Controlador cont = new Controlador(); cont.setVista(ventana);</p> <p>Pedido p1 = new Pedido(...)</p> <p>Chofer chP1 = new ChoferPermanente(...)</p> <p>Vehiculo a1 = new Auto(...)</p> <p>when(ventana.getPedidoSeleccionado()).thenReturn(p1);</p> <p>when(ventana.getChoferDisponiblesSeleccionado()).thenReturn(chP1);</p> <p>when(ventana.getVehiculosDisponiblesSeleccionado()).thenReturn(a1);</p> <p>cont.actionPerformed(e);</p>	<p>Se crea el viaje y se almacena en HashMap[Cliente, Viaje] viajesIniciados ; entonces, Empresa.getViajesIniciados().size == 1</p> <p>Se actualiza la empresa con la información nueva.</p>	

Método: public void nuevoViaje()

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

```
Pedido pedido = this.vista.getPedidoSeleccionado()
```

```
Chofer chofer = this.vista.getChoferDisponibleSeleccionado()
```

```
Vehiculo vehiculo = this.vista.getVehiculoDisponibleSeleccionado()
```

```
Empresa.getInstance().crearViaje(pedido, chofer, vehiculo)
```

Luego se actualiza la vista

Si la acción no se puede realizar por cualquier motivo, entonces se delega en el atributo **vista** mostrar el mensaje correspondiente a la excepcion lanzada

Casos de prueba:

Entrada	Condiciones de entrada	Salida esperada	Condiciones de salida
	<p>Clase Ventana presente y Clase Empresa presente.</p> <p>(Se deben dar las condiciones para generar un viaje con los 3 datos recibidos por la ventana)</p> <p>IVista ventana = mock(IVista.class);</p> <p>Controlador cont = new Controlador();</p> <p>cont.setVista(ventana);</p> <p>Pedido p1 = new Pedido(...)</p> <p>Chofer chP1 = new</p>	<p>Se actualiza la empresa con la información nueva.</p> <p>Se crea el viaje y se almacena en HashMap[Cliente, Viaje] viajesIniciados ; entonces, Empresa.getViajesIniciados().size == 1</p>	

	<p>ChoferPermanente(...)</p> <p>Vehiculo a1 = new Auto(...)</p> <p>when(ventana.getPedidoSeleccionado()).thenReturn(p1);</p> <p>when(ventana.getChoferDisponibleSeleccionado()).thenReturn(chP1);</p> <p>when(ventana.getVehiculoDisponibleSeleccionado()).thenReturn(a1);</p> <p>cont.nuevoViaje();</p>		
	Clase Ventana ausente y Clase Empresa presente.	falla de ejecución	
	Clase Ventana presente y Clase Empresa ausente.	falla ejecución	
	<p>Clase Ventana presente y Clase Empresa presente. (chofer es null)</p> <p>IVista vista = mock(IVista.class);</p> <p>Controlador cont = new Controlador();</p> <p>Controlador.setVista(vista);</p> <p>Pedido p1 = new Pedido(...)</p> <p>Vehiculo a1 = new Auto(...)</p>	<p>ventana.getOptionPane().getMensaje() == Mensajes.PARAMETROS_NULOS.getValor().</p>	

	<p>when(ventana.getPedidoSeleccionado()).thenReturn(p1);</p> <p>when(ventana.getChoferDisponibleSeleccionado()).thenReturn(null);</p> <p>when(ventana.getVehiculoDisponibleSeleccionado()).thenReturn(a1);</p> <p>cont.nuevoViaje();</p>		
	<p>Clase Ventana presente y Clase Empresa presente. (pedido es null)</p> <p>IVista vista = mock(IVista.class);</p> <p>Controlador cont = new Controlador();</p> <p>Controlador.setVista(vista);</p> <p>Chofer chP1 = new ChoferPermanente(...)</p> <p>Vehiculo a1 = new Auto(...)</p> <p>when(ventana.getPedidoSeleccionado()).thenReturn(null);</p> <p>when(ventana.getChoferDisponibleSeleccionado()).thenReturn(chP1);</p> <p>when(ventana.getVehiculoDisponibleSeleccionado()).thenReturn(a1);</p>	<p>ventana.getOptionPane().getMensaje() == Mensajes.PARAMETROS_NULOS.getValor().</p>	

	cont.nuevoViaje();		
	<p>Clase Ventana presente y Clase Empresa presente. (vehiculo es null)</p> <p>IVista vista = mock(IVista.class);</p> <p>Controlador cont = new Controlador();</p> <p>Controlador.setVista(vista);</p> <p>Pedido p1 = new Pedido(...)</p> <p>Chofer chP1 = new ChoferPermanente(...)</p> <p>when(ventana.getPedidoSeleccionado()).thenReturn(p1);</p> <p>when(ventana.getChoferDisponibleSeleccionado()).thenReturn(chP1);</p> <p>when(ventana.getVehiculoDisponibleSeleccionado()).thenReturn(null);</p> <p>cont.nuevoViaje();</p>	<p>ventana.getOptionPane().getMensaje() == <i>Mensajes.PARAMETROS_NULOS</i>.getValor().</p>	
	<p>Clase Ventana presente y Clase Empresa presente. (El vehiculo "a1" no pertenece al ArrayList de vehiculosDisponibles)</p>	<p>ventana.getOptionPane().getMensaje() == <i>Mensajes.VEHICULO_NO_DISPONIBLE</i>.getValor();</p>	

	<pre> IVista ventana = mock(IVista.class); Controlador cont = new Controlador(); cont.setVista(ventana); Pedido p1 = new Pedido(...) Chofer chP1 = new ChoferPermanente(...) Vehiculo a1 = new Auto(...) when(ventana.getPedidoSeleccionado()).thenReturn(p1); when(ventana.getChoferDisponibleSeleccionado()).thenReturn(chP1); when(ventana.getVehiculoDisponibleSeleccionado()).thenReturn(a1); cont.nuevoViaje(); </pre>		
	<p>Clase Ventana presente y Clase Empresa presente. (El el chofer “chP1” no pertenece al ArrayList de choferesDisponibles)</p> <pre> IVista ventana = mock(IVista.class); Controlador cont = new Controlador(); </pre>	<pre> ventana.getOptionPane(). getMensaje() == Mensajes.CHOFER_NO_DISPONIBLE.getValor(); </pre>	

	<pre> cont.setVista(ventana); Pedido p1 = new Pedido(...) Chofer chP1 = new ChoferPermanente(...) Vehiculo a1 = new Auto(...) when(ventana.getPedidoSeleccionado()).then Return(p1); when(ventana.getChoferDisponibleSeleccionado()).thenReturn(chP1); when(ventana.getVehiculoDisponibleSeleccionado()).thenReturn(a1); cont.nuevoViaje(); </pre>		
	<p>Clase Ventana presente y Clase Empresa presente. (El pedido “p1” no pertenece al HashMap de pedidos)</p> <pre> IVista ventana = mock(IVista.class); Controlador cont = new Controlador(); cont.setVista(ventana); Pedido p1 = new Pedido(...) Chofer chP1 = new ChoferPermanente(...) Vehiculo a1 = new </pre>	<pre> ventana.getOptionPane(). getMensaje() == Mensajes.PEDIDO_INEXISTENTE.getValor(); </pre>	

	<pre> Auto(...) when(ventana.getPedidoSeleccionado()).then Return(p1); when(ventana.getChoferDisponibleSeleccionado()).thenReturn(chP1); when(ventana.getVehiculoDisponibleSeleccionado()).thenReturn(a1); cont.nuevoViaje(); </pre>		
	<p>Clase Ventana presente y Clase Empresa presente. (El el vehiculo “a1” no puede satisfacer el pedido “p1”)</p> <pre> IVista ventana = mock(IVista.class); Controlador cont = new Controlador(); cont.setVista(ventana); Pedido p1 = new Pedido(...) Chofer chP1 = new ChoferPermanente(...) Vehiculo a1 = new Auto(...) when(ventana.getPedidoSeleccionado()).then Return(p1); when(ventana.getChoferDisponibleSeleccionado()).thenReturn(chP </pre>	<pre> ventana.getOptionPane(). geMensaje() == Mensajes.VEHICULO_NO_VALIDO.getValor(); </pre>	

	<pre> 1); when(ventana.getVehiculoDisponibleSeleccionado()).thenReturn(a1); cont.nuevoViaje(); </pre>		
	<p>Clase Ventana presente y Clase Empresa presente. (el Cliente, que anteriormente hizo un pedido, está realizando un Viaje)</p> <pre> IVista ventana = mock(IVista.class); Controlador cont = new Controlador(); cont.setVista(ventana); Pedido p1 = new Pedido(...) Chofer chP1 = new ChoferPermanente(...) Vehiculo a1 = new Auto(...) when(ventana.getPedidoSeleccionado()).thenReturn(p1); when(ventana.getChoferDisponibleSeleccionado()).thenReturn(chP1); when(ventana.getVehiculoDisponibleSeleccionado()).thenReturn(a1); cont.nuevoViaje(); </pre>	<pre> ventana.getOptionPane().getMensaje() == Mensajes.CLIENTE_CON_VIAJE_PENDIENTE.getValor(); </pre>	

Método: public void nuevoVehiculo()

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

```
String tipo = this.vista.getTipoVehiculo()
```

```
String patente = this.vista.getPatente()
```

Si tipo == Constantes.MOTO se agrega una moto con el parámetro "patente"

Si tipo == Constantes.AUTO o tipo== Constantes.COMBI

```
int plazas = this.vista.getPlazas()
```

```
boolean mascota = this.vista.isVehiculoAptoMascota()
```

se crea una combi o un auto segun corresponda con los parametros "patente", "plazas", "mascota"

Luego se actualiza la vista

Si la accion no se puede realizar por cualquier motivo, entonces se delega en el atributo **vista** mostrar el mensaje correspondiente a la excepción lanzada

Casos de prueba:

Entrada	Condiciones de entrada	Salida esperada	Condiciones de salida
	Clase Ventana ausente y Clase Empresa presente.	falla de ejecución	
	Clase Ventana presente y Clase Empresa ausente.	falla ejecución	
	Clase Ventana presente y Clase Empresa presente. (la patente del parámetro vehiculo coincide con la de algun vehiculo previamente registrado)	ventana.getOptionPane().getMessage() == Mensajes.VEHICULO_YA_REGISTRADO.getValue();	

	<p>El Vehiculo moto1 = new Moto("532") ya debe estar agregado en la Empresa</p> <p>IVista ventana = mock(IVista.class);</p> <p>Controlador cont = new Controlador(); cont.setVista(ventana);</p> <p>when(ventana.getTipo Vehiculo()).thenReturn(Constantes.MOTO);</p> <p>when(ventana.getPate nte()).thenReturn("532");</p> <p>cont.nuevoViaje();</p>		
	<p>Clase Ventana presente y Clase Empresa presente. (La empresa NO tiene vehiculos agregados y agrega una MOTO)</p> <p>IVista ventana = mock(IVista.class);</p> <p>Controlador cont = new Controlador(); cont.setVista(ventana);</p> <p>when(ventana.getTipo Vehiculo()).thenReturn(Constantes.MOTO);</p>	<p>Empresa.getVehiculos.size() == 1</p> <p>Se actualiza la empresa con la información nueva.</p>	

	when(ventana.getPate nte()).thenReturn("532"); cont.nuevoViaje();		
	Clase Ventana presente y Clase Empresa presente. (La empresa NO tiene vehiculos agregados y agrega una COMBI) IVista ventana = mock(IVista.class); Controlador cont = new Controlador(); cont.setVista(ventana); when(ventana.getTipo Vehiculo()).thenReturn(Constantes.COMBI); when(ventana.getPate nte()).thenReturn("532"); when(ventana.getPlaz as()).thenReturn(8); when(ventana.isVehicu loAptoMascota()).then Return(true); cont.nuevoViaje();	Empresa.getVehiculos.size() == 1 Se actualiza la empresa con la información nueva.	
	Clase Ventana presente y Clase Empresa presente. (La empresa NO tiene vehiculos agregados y agrega un AUTO)	Empresa.getVehiculos.size() == 1 Se actualiza la empresa con la información nueva.	

	<pre> IVista ventana = mock(IVista.class); Controlador cont = new Controlador(); cont.setVista(ventana); when(ventana.getTipo Vehiculo()).thenReturn(Constantes.AUTO); when(ventana.getPate nte()).thenReturn("532"); when(ventana.getPlaz as()).thenReturn(3); when(ventana.isVehicu loAptoMascota()).then Return(true); cont.nuevoViaje(); </pre>		
--	---	--	--

Método: public void nuevoChofer()

Se invoca al *metodo agregarChofer(Chofer chofer)* de la clase Empresa con los parametros obtenidos del atributo **vista**:

```
String tipo = this.vista.getTipoChofer()
```

```
String nombre = this.vista.getNombreChofer()
```

```
String dni = this.vista.getDNIChofer();
```

Si tipo== Constantes.TEMPORARIO se agrega un chofer temporario con los parámetros "dni" y "nombre"

Si tipo== Constantes.PERMANENTE

```
int anio = this.vista.getAnioChofer()
```

```
int hijos = this.vista.getHijosChofer()
```

se agrega un chofer permanente con los parametros "dni", "nombre", "anio" e "hijos"

Luego se actualiza la vista

Si la accion no se puede realizar porque el dni esta repetido entonces se delega en el atributo vista mostrar el mensaje correspondiente a la excepcion ChoferRepetidoException lanzada

Casos de prueba:

Entrada	Condiciones de entrada	Salida esperada	Condiciones de salida
	Clase Ventana ausente y Clase Empresa presente.	falla de ejecución	
	Clase Ventana presente y Clase Empresa ausente.	falla ejecución	
	Clase Ventana presente y Clase Empresa presente. (La empresa NO tiene ningún chofer registrado y agrega un chofer PERMANENTE) IVista ventana = mock(IVista.class); Controlador cont = new Controlador(); cont.setVista(ventana); when(ventana.getTipoChofer()).thenReturn(Constants.PERMANENTE);	Chofer agregado al HashMap Empresa.getInstance().getChofers().size() == 1 Se actualiza la empresa con la información nueva.	

	<pre> when(ventana.getDNIChofer()).thenReturn("43666918"); when(ventana.getNombreChofer()).thenReturn("Jose"); when(ventana.getAnioChofer()).thenReturn(2000); when(ventana.getHijosChofer()).thenReturn(2); cont.nuevoChofer(); </pre>		
	<p>Clase Ventana presente y Clase Empresa presente. (La empresa NO tiene ningún chofer registrado y agrega un chofer TEMPORARIO)</p> <pre> IVista ventana = mock(IVista.class); Controlador cont = new Controlador(); cont.setVista(ventana); when(ventana.getTipoChofer()).thenReturn(Constantes.TEMPORARIO); when(ventana.getNombreChofer()).thenReturn("Jose"); when(ventana.getDNIChofer()).thenReturn("43 </pre>	<p>Chofer agregado al HashMap</p> <p><code>Empresa.getInstance().getChoferes().size() == 1</code></p> <p>Se actualiza la empresa con la información nueva.</p>	

	666918"); cont.nuevoChofer();		
	<p>Clase Ventana presente y Clase Empresa presente. (el dni del chofer pasado por parametro coincide con el dni de un chofer previamente registrado)</p> <p>El Chofer chT1 = new ChoferTemporariol("43666918", "Jose") ya debe estar agregado en la Empresa</p> <p>IVista ventana = mock(IVista.class);</p> <p>Controlador cont = new Controlador(); cont.setVista(ventana);</p> <p>when(ventana.getTipoChofer()).thenReturn(Constants.TEMPORARIO);</p> <p>when(ventana.getDNIChofer()).thenReturn("43666918");</p> <p>when(ventana.getNombreChofer()).thenReturn("Jose");</p> <p>cont.nuevoChofer();</p>	ventana.getOptionPane().getMensaje() == Mensajes.CHOFER_YA_REGISTRADO.getValor();	

Método: public void calificarPagar()

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

```
int calificacion = this.vista.getCalificacion()
```

Luego se actualiza la vista

Si la acción no se puede realizar entonces se delega en el atributo vista mostrar el mensaje correspondiente a la excepcion ClienteSinViajePendienteException lanzada

Casos de prueba:

Entrada	Condiciones de entrada	Salida esperada	Condiciones de salida
	Clase Ventana ausente y Clase Empresa presente.	falla de ejecución	
	Clase Ventana presente y Clase Empresa ausente.	falla ejecución	
	Clase Ventana presente y Clase Empresa presente. (El Cliente ESTA realizando un viaje) IVista ventana = mock(IVista.class); Controlador cont = new Controlador(); cont.setVista(ventana);	Se finaliza y califica un viaje pendiente y se almacena en ArrayList[Viaje] viajesTerminados; entonces, Empresa.getViajesIniciados().size == 0 Empresa.getViajesTerminados().ge t(0)== viaje(ped,cht,v1) verificando que Empresa.getViajesTerminados().ge t(0).getCalificacion() == viaje.getCalificacion()	

	<pre>when(ventana.getCalificacion()).thenReturn(4);</pre> <pre>cont.calificarPagar();</pre>	Se actualiza la empresa con la información nueva.	
	<p>Clase Ventana presente y Clase Empresa presente. (el Cliente no esta realizando un viaje)</p> <pre>IVista ventana = mock(IVista.class);</pre> <pre>Controlador cont = new Controlador();</pre> <pre>cont.setVista(ventana);</pre> <pre>when(ventana.getCalificacion()).thenReturn(4);</pre> <pre>cont.calificarPagar();</pre>	<pre>ventana.getOptionPane().getMessage() == Mensajes.CLIENTE_SIN_VIAJE_PENDIENTE.getValor();</pre>	

Método: `public void nuevoPedido()`

Pre: Hay un usuario de tipo Cliente logeado en la Empresa

Invoca al *metodo agregarPedido(Pedido pedido)* de la clase Empresa

Para crear el pedido se utilizan los parámetros proporcionados por el atributo **vista**:

Cliente cliente = (Cliente) Empresa.getInstance().getUsuarioLogeado()

```
int cantidadPasajeros = this.vista.getCantidadPax()
```

```
boolean mascota = this.vista.isPedidoConMascota()
```

```
boolean baul=this.vista.isPedidoConBaul()
```

```
int km=this.vista.getCantKm()
```

```
String zona = this.vista.getTipoZona() (puede ser: Constantes.ZONA_STANDARD
```

```
Constantes.ZONA_PELIGROSA Constantes.ZONA_SIN_ASFALTAR)
```

luego se actualiza la vista

Si la acción no se puede realizar entonces se delega en el atributo vista mostrar el mensaje correspondiente a la excepción lanzada.

Casos de prueba:

Entrada	Condiciones de entrada	Salida esperada	Condiciones de salida
	Clase Ventana ausente y Clase Empresa presente.	falla de ejecución	
	Clase Ventana presente y Clase Empresa ausente.	falla ejecución	
	Clase Ventana presente y Clase Empresa presente. (Se puede agregar un nuevo pedido a la empresa) Cliente cli = new Cliente (usuario1,1234,juan); Empresa.setUsuarioLogeado(cli);	Empresa.getPedidos().size() -1 == 0 Se actualiza la empresa con la información nueva.	

	<p>Vehiculo v1 = new Combi("0809ad",6,true)</p> <p>Empresa.agregarVehiculo(v1);</p> <p>v1 satisface al pedido generado por la vista</p> <p>IVista ventana = mock(IVista.class);</p> <p>Controlador cont = new Controlador();</p> <p>cont.setVista(ventana);</p> <p>when(ventana.getCantidadPax()).thenReturn(2);</p> <p>when(ventana.isPedidoConMascota()).thenReturn(false);</p> <p>when(ventana.isPedidoConBaul()).thenReturn(false);</p> <p>when(ventana.getCantKm()).thenReturn(20);</p> <p>when(ventana.getTipoZona()).thenReturn(Constantes.ZONA_PELIGROSA);</p> <p>cont.nuevoPedido();</p>		
	<p>Clase Ventana presente y Clase Empresa presente.</p> <p>(La empresa no tiene registrado ningún vehículo con las características necesarias para satisfacer el pedido.)</p>	<p>ventana.getOptionPane().getMessage() == Mensajes.SIN_VEHICULO_PARA_PEDIDO.getValor();</p>	

	<p> Cliente cli = new Cliente (usuario1, 1234, juan); </p> <p> Empresa.setUsuarioLo geado(cli); </p> <p> Se debe cumplir que: Se tengan vehiculos en el HashMap[String, Vehiculo] vehiculos vehiculos que NO satisfagan el pedido generado por la vista </p> <p> IVista ventana = mock(IVista.class); </p> <p> Controlador cont = new Controlador(); cont.setVista(ventana); </p> <p> when(ventana.getCanti dadPax()).thenReturn(2); </p> <p> when(ventana.isPedido ConMascota()).thenRet urn(false); </p> <p> when(ventana.isPedido ConBaul()).thenReturn(false); </p> <p> when(ventana.getCant Km()).thenReturn(20); </p> <p> when(ventana.getTipo Zona()).thenReturn(Co nstantes.ZONA_PELIG ROSA); </p> <p> cont.nuevoPedido(); </p>		
	<p> Clase Ventana presente y </p>	<p> ventana.getOptionPane().getM ensaje() == </p>	

	<p>Clase Empresa presente. (La empresa no tiene registrado ningún vehículo con las características necesarias para satisfacer el pedido.)</p> <p>Cliente cli = new Cliente (usuario1, 1234, juan);</p> <p>Empresa.setUsuarioLo geado(cli);</p> <p>Empresa.getVehiculos. size() == 0</p> <p>IVista ventana = mock(IVista.class);</p> <p>Controlador cont = new Controlador(); cont.setVista(ventana);</p> <p>when(ventana.getCanti dadPax()).thenReturn(2);</p> <p>when(ventana.isPedido ConMascota()).thenRet urn(false);</p> <p>when(ventana.isPedido ConBaul()).thenReturn(false);</p> <p>when(ventana.getCant Km()).thenReturn(20);</p> <p>when(ventana.getTipo Zona()).thenReturn(Co nstantes.ZONA_PELIG ROSA);</p>	<p>Mensajes.SIN_VEHICULO_PA RA_PEDIDO.getValor();</p>	
--	--	--	--

	cont.nuevoPedido();		
	<p>Clase Ventana presente y Clase Empresa presente. (El Cliente tiene un viaje iniciado.)</p> <p>Cliente cli = new Cliente (usuario1,1234,juan);</p> <p>Empresa.getInstance().login("usuario1","1234");</p> <p>Auto v1 = new Auto("NOW678", 3, false);</p> <p>Auto v2 = new Auto("ETC567",3,true);</p> <p>ped=new Pedido(cli,...)</p> <p>Empresa.agregarVehiculo(v1);</p> <p>Empresa.agregarVehiculo(v2);</p> <p>v1 satisface al ped</p> <p>v2 satisface al pedido generado por la vista</p> <p>Empresa.AgregarPedido(pedido1)</p> <p>Empresa.crearViaje(pedido1,...)</p> <p>IVista ventana = mock(IVista.class);</p> <p>Controlador cont = new Controlador();</p>	<p>ventana.getOptionPane().getMensaje() == Mensajes.CLIENTE_CON_VIAJE_PENDIENTE.getValor();</p>	

	<pre> cont.setVista(ventana); when(ventana.getCantidadPax()).thenReturn(2); when(ventana.isPedidoConMascota()).thenReturn(false); when(ventana.isPedidoConBaul()).thenReturn(false); when(ventana.getCantKm()).thenReturn(20); when(ventana.getTipoZona()).thenReturn(Constants.ZONA_PELIGROSA); cont.nuevoPedido(); </pre>		
	<p>Clase Ventana presente y Clase Empresa presente. (El Cliente tiene un pedido iniciado.)</p> <pre> Cliente cli = new Cliente(usuario1, 1234, juan); Empresa.getInstance().login("usuario1", "1234"); Auto v1 = new Auto("NOW678", 3, false); Auto v2 = new Auto("ETC567", 3, true); ped=new Pedido(cli,...) Empresa.agregarVehiculo(v1); </pre>	<pre> ventana.getOptionPane().getMensaje() == Mensajes.CLIENTE_CON_PEDIDO_PENDIENTE.getValor(); </pre>	

	<p>Empresa.agregarVehiculo(v2);</p> <p>v1 satisface al pedido</p> <p>v2 satisface al pedido generado por la vista</p> <p>Empresa.AgregarPedido(pedido1)</p> <p>IVista ventana = mock(IVista.class);</p> <p>Controlador cont = new Controlador(); cont.setVista(ventana);</p> <p>when(ventana.getCantidadPax()).thenReturn(2);</p> <p>when(ventana.isPedidoConMascota()).thenReturn(false);</p> <p>when(ventana.isPedidoConBaul()).thenReturn(false);</p> <p>when(ventana.getCantKm()).thenReturn(20);</p> <p>when(ventana.getTipoZona()).thenReturn(Constants.ZONA_PELIGROSA);</p> <p>cont.nuevoPedido();</p>		
--	---	--	--

Método: public void registrar()

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

```
String nombreReal = this.vista.getRegNombreReal()
```

```
String nombreUsuario = this.vista.getRegUssetName()
```

```
String pass = this.vista.getRegPassword()
```

```
String confirm = this.vista.getRegConfirmPassword()
```

Si "pass" y "confirm" no coinciden, entonces se delega en el atributo **vista** mostrar el mensaje correspondiente a *Mensajes.PASS_NO_COINCIDE.getValor()*.

Si "pass" y "confirm" coinciden, se invoca al metodo *agregarCliente(nombreUsuario, pass, nombreReal)* de la clase Empresa.

Si la acción no se puede realizar entonces se delega en el atributo **vista** mostrar el mensaje correspondiente a la excepción lanzada.

Casos de prueba:

Entrada	Condiciones de entrada (estados de las clases invocadas)	Salida esperada	Condiciones de salida
	Clase Ventana ausente	Falla de ejecución.	
	<pre>Clase Ventana presente y "pass" y "confirm" no coinciden: IVista ventana = mock(IVista.class); Controlador cont = new Controlador(); cont.setVista(ventana); when(ventana.getRegNo mbreReal()).thenReturn("Juan"); when(ventana.getRegUs ername()).thenReturn("Juan95"); when(ventana.getRegPa ssword()).thenReturn("45");</pre>	<pre>Se delega en el atributo vista mostrar el mensaje correspondiente a Mensajes.PASS_NO_C OINCIDE.getValor().</pre>	

	when(ventana.getRegConfirmPassword()).thenReturn("61");		
	<p>Clase Ventana presente y "pass" y "confirm" coinciden; pero ya existe el usuario a registrar:</p> <pre> IVista ventana = mock(IVista.class); Controlador cont = new Controlador(); cont.setVista(ventana); when(ventana.getRegNombreReal()).thenReturn("Juan"); when(ventana.getRegUserName()).thenReturn("Juan95"); when(ventana.getRegPassword()).thenReturn("45"); when(ventana.getRegConfirmPassword()).thenReturn("45"); </pre>	<p>Se delega en el atributo vista mostrar el mensaje correspondiente a Mensajes.USUARIO_REGISTRO.getValor().</p>	
	<p>Clase Ventana presente y "pass" y "confirm" coinciden; pero NO existe el usuario a registrar:</p> <pre> IVista ventana = mock(IVista.class); Controlador cont = new Controlador(); cont.setVista(ventana); when(ventana.getRegNombreReal()).thenReturn("Juan"); when(ventana.getRegUs </pre>	<pre> cont.registrar() Empresa.getInstance().getClientes().size() == 1 </pre>	

	<pre> serName()).thenReturn(" Juan95"); when(ventana.getRegPa ssword()).thenReturn("45 "); when(ventana.getRegCo nfirmPassword()).thenRet urn("45"); </pre>		
--	---	--	--

Método: public void login()

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

```
String usserName = vista.getUsserName()
```

```
String pass = vista.getPassword()
```

Se invoca al metodo login(usserName, pass) de la clase Empresa. Si la acción no se puede realizar entonces se delega en el atributo **vista** mostrar el mensaje correspondiente a la excepción lanzada

Casos de prueba:

Entrada	Condiciones de entrada (estados de las clases invocadas)	Salida esperada	Condiciones de salida
	Clase Ventana ausente	Falla de ejecución.	
	Clase Ventana presente y Clase Empresa ausente	Falla de ejecución.	
	Clase Ventana presente, Clase Empresa presente; pero no existe ningún Usuario (Cliente) registrado con ese nombre de usuario.	<p>Se lanza la excepción UsuarioNoExisteExcepcion.</p> <p>vista.getOptioPane().getMensaje() == Mensajes.USUARIO_DESCONOCIDO</p>	

	<pre> IVista ventana = mock(IVista.class); Controlador cont = new Controlador(); cont.setVista(ventana); when(ventana.getUsserN ame()).thenReturn("juan9 5"); when(ventana.getPassw ord()).thenReturn("1234") ; cont.login(); </pre>		
	<p>Clase Ventana presente, Clase Empresa presente; pero el password del Usuario (Cliente) es incorrecto.</p> <p>Empresa.agregarCliente("usuario1", "1234", "juan")</p> <pre> IVista ventana = mock(IVista.class); Controlador cont = new Controlador(); cont.setVista(ventana); when(ventana.getUsserN ame()).thenReturn("usua rio1"); when(ventana.getPassw ord()).thenReturn("31"); cont.login(); </pre>	<p>Se lanza la excepción PasswordErro neaException.</p> <p>vista.getOptioPane().get Mensaje() == Mensajes.PASS_ERRONE O</p>	
	<p>Clase Ventana presente,</p>	<p>Se lanza la</p>	

	<p>Clase Empresa presente; pero el password del Usuario (Administrador) es incorrecto.</p> <pre> I Vista ventana = mock(I Vista.class); Controlador cont = new Controlador(); cont.setVista(ventana); when(ventana.getUsserName()).thenReturn("admin"); when(ventana.getPassword()).thenReturn("31"); cont.login(); </pre>	<p>excepción PasswordErrorException.</p> <pre> vista.getOptionPane().getMessage() == Mensajes.PASS_ERRONEO </pre>	
	<p>Clase Ventana presente, Clase Empresa presente, existe vista.getUserName() (Cliente) y vista.getPassword() es correcta.</p> <pre> Cliente cli = new Cliente("usuario1", "1234", "juan"); Empresa.agregarCliente("usuario1", "1234", "juan") I Vista ventana = mock(I Vista.class); Controlador cont = new Controlador(); cont.setVista(ventana); when(ventana.getUserName()).thenReturn("usua </pre>	<p>Realiza el logeo de un Usuario (Cliente) al sistema.</p> <pre> cli == Empresa.getInstance().getUsuarioLogeado(); </pre>	

	<pre> rio1"); when(ventana.getPassword()).thenReturn("1234"); cont.login(); </pre>		
	<p>Clase Ventana presente, Clase Empresa presente, existe vista.getUserN (Administrador) y vista.getPassword() es correcta.</p> <p>IVista ventana = mock(IVista.class);</p> <p>Controlador cont = new Controlador();</p> <p>cont.setVista(ventana);</p> <p>when(ventana.getUserN ame()).thenReturn("admi n");</p> <p>when(ventana.getPassw ord()).thenReturn("admin ");</p> <p>cont.login();</p>	<p>Realiza el logeo de un Usuario (Administrador) al sistema.</p> <p>Administrador.getInstanc e() == Empresa.getInstance().g etUsuarioLogeado();</p>	

Método: public void logout()

Se invoca al método logout de la clase empresa Luego se invoca al método this.escribir()

Casos de prueba:

Entrada	Condiciones de entrada (estados de las clases invocadas)	Salida esperada	Condiciones de salida
	Clase Empresa ausente	Fallo de ejecución.	
	<p>Clase Administrador y Empresa presente, con:</p> <pre>File arch = new File("empresa.bin"); if(arch.exists()) arch.delete();</pre> <p><i>Empresa.getInstance().login("admin","admin")</i></p> <p>Controlador cont = new controlador()</p> <p>cont.logout()</p>	<p><i>Empresa.getInstance(),is Admin() == false</i></p> <p>arch.exists() == true</p>	
	<p>Clase Cliente y Empresa presente, con:</p> <pre>File arch = new File("empresa.bin"); if(arch.exists()) arch.delete();</pre> <p>Empresa.getInstance().agregarCliente("usuario1", "1234", "juan");</p> <p><i>Empresa.getInstance().login("usuario1", "1234")</i></p> <p>Controlador cont = new controlador()</p> <p>cont.logout()</p>	<p><i>empresa.getUsuarioLogeado() == null</i></p> <p>arch.exists() == true</p>	

Resultados obtenidos

Los resultados obtenidos, por los test realizados en JUnit 4, se encuentran en el archivo “XML test.xml” dentro del repositorio de GuitHub.

Conclusiones

Cumplimiento de expectativas

Al comienzo del trabajo final las expectativas eran las de poder solidificar nuestros conocimientos a medida que íbamos avanzando en la materia y, en última instancia, generar un proyecto informático en base a aquellos conocimientos adquiridos. Por otro lado, se promovió un trabajo colaborativo activo entre los diferentes integrantes del grupo para facilitar la realización de dicho trabajo final por medio de la metodología anteriormente descrita.

Luego de la completa realización del trabajo se obtuvo mucha experiencia, tanto en el ámbito de la cooperación con otras personas en la realización de un proyecto en conjunto; como en el ámbito académico, por medio de la obtención de muchos recursos a la hora de abordar un tema del cual se tenía muy poco conocimiento como lo es el testing.

Por último, cabe resaltar el hecho de que a pesar de ser un trabajo de testeo de un sistema informático complejo, se pudieron cumplir con la mayoría de las expectativas presentes al comienzo del mismo; aunque es cierto que siempre se puede pulir un poco más el trabajo ya terminado.

Dificultades encontradas

En primer lugar, la principal dificultad encontrada fue cómo se desarrollaría la dinámica grupal para con la realización del trabajo; ya que el tiempo que dispone un integrante no es el mismo para todo el resto. Esto se pudo ir trabajando mediante la división de tareas hecha desde el principio de cada parte, esta metodología permitió que cada integrante pueda dedicarle el mayor tiempo posible a su respectiva tarea. Por último, la unificación de las diferentes partes desarrollada por cada integrante se realizó de manera grupal de forma tal que se puedan evacuar dudas y realizar modificaciones finales.

Aprendizaje

Para cerrar el informe de este trabajo final, se debe mencionar que todo trabajo no es perfecto y siempre se puede mejorar algo del mismo. Como aprendizaje de este trabajo, se debe rescatar el

hecho de que una comunicación fluida con todos los integrantes del grupo de trabajo repercute directamente en el resultado del mismo. Por otro lado, el haber dispuesto de más conocimiento y experiencia acerca de los temas vistos en clase; hubiera ocasionado una realización mucho más fluida y satisfactoria de este trabajo final.

Cabe resaltar que el testing aplicado en este trabajo final, representa una de las tantas formas de encontrar errores en los diversos programas desarrollados en un lenguajes de programación. Cada lenguaje en sí posee una cantidad exponencial de casos de análisis que no pueden ser testeados en su totalidad para garantizar un funcionamiento al 100% del sistema; por lo cual y dependiendo, principalmente, del dominio del problema y del lenguaje de programación es necesario realizar un recorte de los casos a testear en el sistema desarrollado.