Informe trabajo práctico final

Grupo 5

Integrantes

Bailon, Guido
Barriga, Nahuel
Hernandez, Julieta
Zamora Wendy

Repositorio GitHub

https://github.com/NahuelBarriga/Taller1

Introducción:

En el presente trabajo buscaremos explicar de forma abarcativa los procesos que fueron llevados a cabo al momento de realizar un test completo de un sistema de gestión de búsquedas laborales que nos fue provisto por la cátedra.
El marco teórico abordado en este estudio sirve como base conceptual para entender la lógica y el funcionamiento del sistema de gestión de búsquedas laborales. Se explorarán los fundamentos teóricos que respaldan la estructura y el diseño del sistema, proporcionando así un contexto sólido para evaluar su rendimiento.
Además, se prestará especial atención a la toma de decisiones durante el proceso de prueba. Se analizarán las elecciones estratégicas tomadas para diseñar y ejecutar los diferentes tipos de pruebas, considerando factores como la cobertura, la eficiencia y la relevancia para los objetivos del sistema.
A lo largo del informe se dará a entender en gran detalle el proceso realizado durante el testing, el marco teórico tomado en cuenta, la toma de decisiones, los problemas encontrados.
También se mencionan los errores encontrados durante la ejecución de los diversos test.
Se llevaron a cabo métodos de testeo de pruebas unitarias, de caja negra y blanca, de integración, persistencia y finalmente GUI. Durante la ejecución de las pruebas dinámicas, se identificarán y documentarán los problemas encontrados. Estos problemas pueden abarcar desde incongruencias en la interfaz de usuario hasta fallos en la lógica de negocio del sistema. Cada error será detalladamente registrado, proporcionando una visión completa de los desafíos enfrentados durante el proceso de prueba.

PreTesting:

Antes de poder comenzar el proceso de pruebas dentro del sistema debíamos tener acceso al mismo. Para lo mencionado utilizamos el archivo .jar que nos fue proveído por la cátedra y fue importado, a la ruta de ejecución de un nuevo *Java Project* creado por nosotros, como una librería. Esto nos garantizo acceso a todos los métodos públicos del sistema para realizar los test correspondientes.
El siguiente paso fue definir qué clases y métodos iban a ser testeados. Para definirlo contábamos con el *javaDoc* del sistema, garantizandonos acceso a todas las descripciones de las clases y métodos, los invariantes de clases y las pre y post condiciones de cada método. Dicho javaDoc nos brindó toda la información necesaria para saber que debíamos testear y de qué forma.
En una *hoja de cálculo de google* creamos diversas páginas, cada una conteniendo cada clase y sus métodos. Para decidir el orden de las clases a testear utilizamos el árbol de jerarquía, empezando desde las ramas hacia adentro.

Caja Negra:

El primer paso fue llevado a cabo utilizando el método de caja negra, esto quiere decir que no tenemos la posibilidad de ver el código, solamente acceso a las funciones públicas del sistema y al propio *JavaDoc.*
Para definir qué debemos testear y de qué forma, utilizamos el método de particiones de equivalencia. Esto quiere decir que debimos subdividir las entradas de los métodos en grupos llamados clases de equivalencia, cada clase de equivalencia deriva en casos de pruebas que fueron testeados posteriormente utilizando JUnit.
Como ya fue mencionado, para realizar los testeos empezamos desde lo más adentro del árbol, en este caso, EmpleadoPretenso y Empleador.

Empleador: (extiende cliente)

Constructor

Empleador(String userName, String password, String realName, String telefono, String rubro, String tipoPersona)

pre: los parámetros son diferentes de null, rubro y tipoPersona de los tipos esperados (contemplados en la clase Constantes)

Dato de Entrada	Clase de equivalencia	¿Aplica?	Motivo	Id. de clase de equivalencia:
userName	userName != null	Si	Cumple contrato	1
	userName == null	No	No cumple contrato	
password	password != null	Si	Cumple contrato	2
	password == null	No	No cumple contrato	
realName	realName alfabético	Si	Cumple contrato	3
	realName == null	No	No cumple contrato	
teléfono	teléfono numérico	Si	Cumple contrato	4
	teléfono == null	No	No cumple contrato	
rubro	rubro != null	No	No cumple contrato	
	rubro == "SALUD" o "COMERCIO_LOCAL" o "COMERCIO_INTERNACIONAL"	Si	Cumple contrato	5
	rubro == cualquier otro	No	No cumple contrato	
tipoPersona	tipoPersona == null	No	No cumple contrato	
	tipoPersona == "FISICA" o "JURIDICA"	Si	Cumple contrato	6
	tipoPersona == cualquier otro	No	No cumple contrato	

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	userName	"Juan123"	userName = "Juan123"	1
	password	"Juan123"	password = "Juan123"	3
	realName	"Juan"	realName = "Juan"	5
	teléfono	"2235698547"	teléfono = "2235698547"	7
	rubro	"salud"	rubro = "salud"	9
	tipoPersona	"fisica"	tipoPersona = "fisica"	12

Métodos de empleador

calculaComision(Ticket ticket)

Este método se utiliza para calcular la comisión que un usuario debe recibir según el tipo de puesto en un Ticket y su puntaje (ver detalles de cálculo de comisión).

Pre: El método requiere un objeto Ticket válido como entrada para realizar los cálculos de comisión.

Escenario 1	this.rubro == "Salud"			

Tabla de Particiones en Clases de Equivalencia

Dato de entrada	Clases de equivalencia	¿Aplica ?	Motivo	Identificador de clase de equivalencia
ticket	ticket != null	Si	Cumple contrato	1
	ticket == null	No	No cumple contrato	-

Batería de Pruebas

número de prueba	Datos de entrada	Valor	Salida Esperada	clases que abarca
	ticket.remuneracion	250.000,00 €	150.000,00 €	1
Escenario 2	this.rubro == "comercio_local"			

Tabla de Particiones en Clases de Equivalencia

Dato de entrada	Clases de equivalencia	Aplica ?	Motivo	Identificador de clase de

				equivalencia
ticket	ticket != null	Si	Cumple contrato	1
	ticket == null	No	No cumple contrato	-

Batería de Pruebas

número de prueba	Datos de entrada	Valor	Salida Esperada	clases que abarca
	ticket.remuneracion	250.000,00 €	175.000,00 €	1

Escenario 3	this.rubro == "comercio_internacional"

Tabla de Particiones en Clases de Equivalencia

Dato de entrada	Clases de equivalencia	Aplica ?	Motivo	Identificador de clase de equivalencia
ticket	ticket != null	Si	Cumple contrato	1
	ticket == null	No	No cumple contrato	-

Batería de Pruebas

número de prueba	Datos de entrada	Valor	Salida Esperada	clases que abarca
	ticket.remuneracion	250.000,00 €	200.000,00 €	1

setCandidato(Cliente candidato)

pre: candidato!= null, candidato es de un tipo de Usuario válido (EmpleadoPretenso para un Empleador y viceversa)

Escenario 1: El cliente es un Empleador

Tabla de Particiones en Clases de Equivalencia

Dato de entrada	Clases de equivalencia	¿Aplica ?	Motivo	Identificador de clase de equivalencia
candidato	candidato = NULL	No	No cumple contrato	-
	candidato es Empleador	No	No cumple contrato	-

	candidato es EmpleadoPretenso	Si	Cumple contrato	1

Batería de Pruebas

número de prueba	Datos de entrada	Valor	Salida Esperada	clases que abarca
1	candidato	EmpleadoPretenso	this.candidato = candidato	1

setListaDePostulantes(ArrayList<ClientePuntaje> listaDePostulantes)

Tabla de Particiones en Clases de Equivalencia

Dato de entrada	Clases de equivalencia	¿Aplica ?	Motivo	Identificador de clase de equivalencia
listaDePostulantes	listaDePostulantes = null	No	No cumple contrato	-
	listaDePostulantes contiene solo Empleador	No	No cumple contrato	-
	listaDePostulantes contiene uno o más EmpleadoPretenso	Si	Cumple contrato	1

Batería de Pruebas

número de prueba	Datos de entrada	Valor	Salida Esperada	clases que abarca
1	listaDePostulantes	Contiene solo EmpleadoPretenso	this.listaDePostulantes = listDePostulantes	1

EmpleadoPretenso: (extiende cliente)
Solo están presentes los atributos *apellido* y *edad* ya que los demás fueron testeados en la clase Empleador.

Constructor

Dato de Entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
apellido	apellido == null	No	No cumple contrato	5.1
	apellido alfabetico	Si	Cumple contrato	5.2
edad	edad < 0	No	No cumple contrato	6.1
	edad = 0	No	No cumple contrato	6.2

	edad > 0	Si	Cumple contrato	6.3

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	usserName	"Juan123"	usserName = "Juan123"	1.1
	password	"Juan123"	password = "Juan123"	2.1
	realName	"Juan"	realName = "Juan"	3.1
	telefono	"2235698547"	telefono = "2235698547"	4.1
	apellido	"Rodriguez"	apellido = "Rodriguez"	5.1
	edad	25	edad = 25	6.1

Métodos

Los getter y setter de los atributos: realName, usserName, password y telefono estan testeadas en la clase Usuario

setEdad(int edad)

Pre: Edad es un valor positivo

datos de entrada	clases de equivalencia	Aplica?	Motivo	identificador de clase
edad	edad > 0	Si		1
	edad < 0	No	No cumple contrato	-

Baterías de prueba

número de prueba	Datos de entrada	Valor	Salida Esperada	clases que abarca
1	edad	26	this.edad= 26	1

setCandidato(Cliente candidato)

pre: candidato!= null, candidato es de un tipo de Usuario válido (EmpleadoPretenso para un Empleador y viceversa)

Tabla de Particiones en Clases de Equivalencia

Dato de entrada	Clases de equivalencia	Aplica ?	Motivo	Identificador de clase de equivalencia
candidato	candidato = NULL	No	No cumple contrato	
	candidato es Empleador	Si		1
	candidato es EmpleadoPretenso	No	No cumple contrato	

Batería de Pruebas				
número de prueba	Datos de entrada	Valor	Salida Esperada	clases que abarca
1	candidato	Empleador	this.candidato = candidato	1

setListaDePostulantes(ArrayList<ClientePuntaje> listaDePostulantes)

pre: listaDePostulantes es diferente de null, los postulantes son de tipo valido (EmpleadorPretensos para Empleadores y viceversa)

Tabla de Particiones en Clases de Equivalencia

Dato de entrada	Clases de equivalencia	Aplica ?	Motivo	Identificador de clase de equivalencia
listaDePostulantes	listaDePostulantes = null	No	No cumple contrato	-
	listaDePostulantes contiene solo Empleador	Si	Cumple contrato	1
	listaDePostulantes contiene uno o mas EmpleadoPretenso	No	No cumple contato	-

Batería de Pruebas				
número de prueba	Datos de entrada	Valor	Salida Esperada	clases que abarca
1	listaDePostulantes	Contiene solo Empleador	this.listaDePostulantes = listDePostulantes	1

class Ticket extends Object

Constructores

Ticket(String locacion, int remuneracion, String jornada, String puesto, String experiencia, String estudios)

Dato de Entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
locacion	locacion == null	No	No cumple contrato	-
	locacion == "HOME_OFFICE"	Si	Cumple contrato	1.1.
	locacion == "PRESENCIAL"	Si	Cumple contrato	1.2
	locacion == INDIFERENTE	Si	Cumple contrato	1.3

remuneracion	Remuneracion < 0	No	No cumple contrato	-
	Remuneracion < V1	Si	Cumple contrato	2.1
	Remuneracion >V1 & Remuneracion < V2	Si	Cumple contrato	2.2
	Remuneracion > V2		Cumple contrato	2.3
jornada	jornada == null	No	No cumple contrato	-
	jornada == "JORNADA_COMPLETA"	Si	Cumple contrato	3.1
	jornada == "JORNADA_MEDIA"	Si	Cumple contrato	3.2
	jornada == "JORNADA_EXTENDIDA"	Si	Cumple contrato	3.3
	jornada == "" || jornada == cualquiera	No	No cumple contrato	-
puesto	puesto == null	No	No cumple contrato	-
	puesto == "JUNIOR"	Si	Cumple contrato	4.1
	puesto == "SENIOR"	Si	Cumple contrato	4.2
	puesto == "GERENCIAL"	Si	Cumple contrato	4.3
experiencia	experiencia == null	No	No cumple contrato	-
	experiencia == "EXP_NADA"	Si	Cumple contrato	5.1
	experiencia == "EXP_MEDIA"	Si	Cumple contrato	5.2
	experiencia = "EXP_MUCHA"	Si	Cumple contrato	5.3
	experiencia == "" || experiencia == cualquiera	No	No cumple contrato	-
estudios	estudios == null	No	No cumple contrato	-
	estudios == "PRIMARIOS"	Si	Cumple contrato	6.1
	estudios == "SECUNDARIOS"	Si	Cumple contrato	6.2
	estudios == "TERCIARIOS"	Si	Cumple contrato	6.3
	estudios == "" || estudios == "cualquiera"	No	No cumple contrato	-

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	locacion	"HOME_OFFICE"	locacion = "HOME_OFFICE"	1.1
	remuneracion	1.200,00 €	Remuneracion = 800	2.1
	jornada	"JORNADA_COMPLETA"	jornada = "JORNADA_COMPLETA"	3.1
	puesto	"JUNIOR"	puesto = "JUNIOR"	4.1

	experiencia	"EXP_NADA"	experiencia = "EXP_NADA"	5.1
	estudios	"PRIMARIOS"	estudios = "PRIMARIOS"	6.1
2	locacion	"PRESENCIAL"	locacion = "PRESENCIAL"	1.2
	remuneracion	20.000,00 €	Remuneracion = 1200	2.2
	jornada	"JORNADA_MEDIA"	jornada = "JORNADA_MEDIA"	3.2
	puesto	"SENIOR"	puesto = "SENIOR"	4.2
	experiencia	"EXP_MEDIA"	experiencia = "EXP_MEDIA"	5.2
	estudios	"SECUNDARIOS"	estudios = "SECUNDARIOS"	6.2
3	locacion	"INDIFERENTE"	locacion = "PRESENCIAL"	1.3
	remuneracion	58.000,00 €	Remuneracion = 3000	2.3
	jornada	"JORNADA_EXTENDIDA"	jornada == "JORNADA_EXTENDIDA"	3.3
	puesto	"MANAGMENT"	puesto = "GERENCIAL"	3.4
	experiencia	"EXP_MUCHA"	experiencia = "EXP_MUCHA"	4.4
	estudios	"TERCIARIOS"	estudios = "TERCIARIOS"	5.4

setRemuneracion(int remuneracion)				
Dato de Entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
remuneracion	remuneracion < 0	No	No cumple contrato	-
	remuneracion < V1	Si	Cumple contrato	1
	remuneracion < V2 && remuneracion > V1	Si	Cumple contrato	2
	remuneracion> V2	Si	Cumple contrato	3
Batería de Pruebas				
Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	remuneracion	20	this.remuneracion== 20	1
2	remuneracion	56	this.remuneracion== 56	2
3	remuneracion	520	this.remuneracion== 520	3

public class Contratación

Clase que representa una contratación entre un empleado y un empleador con su correspondiente fecha de creación Todos los setters tienen como precondición , que su parámetro es diferente de null

Invariante de clase: Los atributos empleado, empleador y fecha son diferentes de null

Constructores

Contratacion(Empleador empleador, EmpleadoPretenso empleado)

Pre: Los parametros son distintos de null.

Dato de Entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
empleado	empleado!= null	Si		1
	empleado== null	No	No cumple contrato	
empleador	empleador!= null	Si		2
	empleador== null	No	No cumple contrato	

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	clases que abarca
1	empleado	empleado	this.empleado= empleado	1
	empleador	empleador	this.empleador= empleador	2

public class clientePuntaje

Clase que representa un Cliente con su corespondiente puntaje obtenido en una busqueda laboral El objeto es Comparable en forma descendente por su puntaje obtenido. Todos los setters tienen como precondicion , que su parametro es diferente de null

Invariante de clase: cliente es distinto de null

Constructores

ClientePuntaje(Double puntaje, Cliente cliente)

Pre: Los parametros son distintos de null.

Dato de Entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
puntaje	puntaje == null	Si	Cumple contrato	1.1
	puntaje > 0	Si	Cumple contrato	1.2
	puntaje < 0	Si	Cumple contrato	1.3
cliente	cliente!= null	Si	Cumple contrato	2.1

	cliente== null	No	No cumple contrato	-
Batería de Pruebas				
Número de prueba	Datos de entrada	Valor	Salida Esperada	clases que abarca
1	cliente	cliente	this.cliente= cliente	2.1
	puntaje	null	this.puntaje = null	1.1
2	cliente	cliente	this.cliente = cliente	2.1
	puntaje	-5	this.puntaje = -5	1.3

Metodos

setCliente(Cliente cliente)				
Pre: Usuario es diferente de null				
Dato de Entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
cliente	cliente== null	No	No cumple contrato	-
	cliente != null	Si	Cumple contrato	1
Batería de Pruebas				
Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	cliente	cliente	this.cliente= cliente	1

Metodos de agencia:

getEstado()

Returns:

un String dependiendo del estado en que se encuentre la Agencia, estos pueden ser:

Mensajes.AGENCIA_EN_CONTRATACION.getValor()

Mensajes.AGENCIA_EN_BUSQUEDA.getValor()

Escenario 1: el estado de contratacion es true

Dato de entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
-	-	Si	-	-

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	-	-	Mensajes.AGENCIA_EN_CONTRATACION.getValor()	-

Escenario 2: el estado de contratacion es false

Dato de entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
-	-	Si	-	-

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	-	-	Mensajes.AGENCIA_EN_BUSQUEDA.getValor()	-

setLimitesRemuneracion(int limiteInferior, int limiteSuperior)

throws LimiteSuperiorRemuneracionInvalidaException, LimiteInferiorRemuneracionInvalidaException

Pameters:

limiteInferior - int

limiteSuperior - int

Returns:

void

Throws				
LimiteSuperiorRemuneracionInvalidaException - Se lanza la excepcion si limite superior es menor				

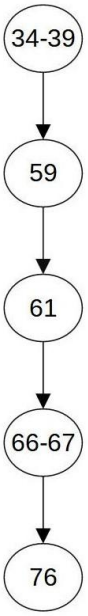
que limite superior				
LimiteInferiorRemuneracionInvalidaException - Se lanza la excepcion si limite inferior es menor a cero				
Dato de Entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
limiteInferior	limiteInferior <= 0	Si		1.1
	limiteInferior > 0	Si		1.2
limiteSuperior	limiteSuperior <=0	Si		2.1
	limiteSuperior >0	Si		2.2

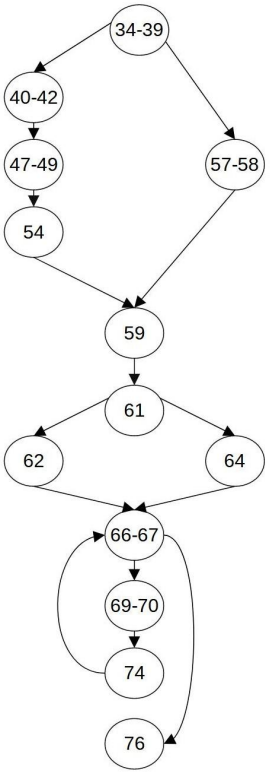
Batería de Pruebas

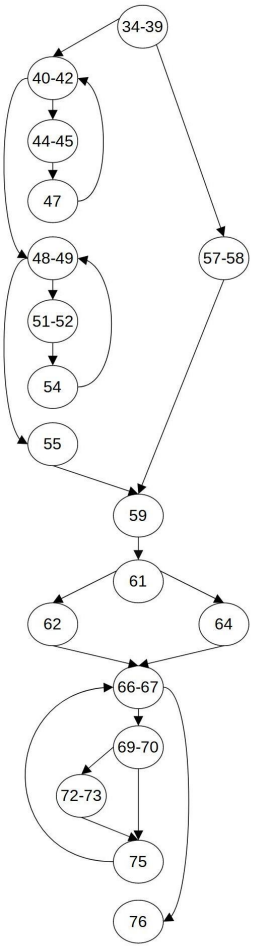
Número de prueba	Datos de entrada	Valor	Salida Esperada	clases que abarca
1	limiteInferior	-5	LimiteInferiorRemuneracionInvalidaException	1.1
	limiteSuperior	-2		2.1
2	limiteInferior	20	LimiteSuperiorRemuneracionInvalidaException	1.1
	limiteSuperior	6		2.2
3	limiteInferior	150	This.limiteInferior = 150	1.2
	limiteSuperior	180	This.limiteSuperior = 180	2.1

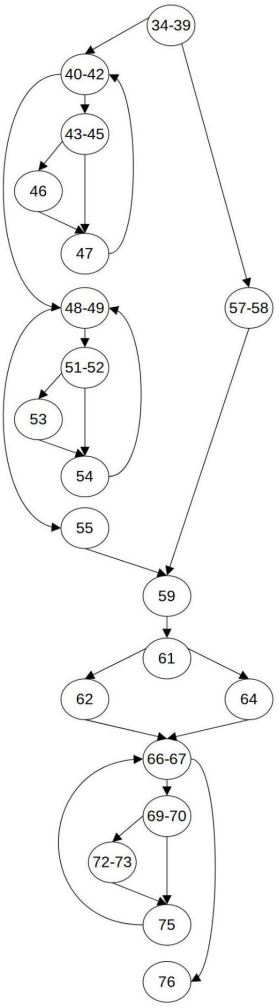
Caja Blanca

Luego de haber finalizado las pruebas de caja negra, fue solicitado que realizáramos un testeo de un método perteneciente a la clase “Agencia”. Brevemente, este método se encarga de aplicar una promoción a un empleado o un empleador en función de un valor boolean pasado como parámetro.
Al utilizar el método de caja blanca, los casos de pruebas se basan en información sobre cómo fue diseñado y codificado el programa. Se analiza el código con la intención de encontrar casos de entrada para luego hacer el test de unidad e intentar que se recorra la mayor cantidad de líneas posibles, ejecutando todos los caminos y bucles posibles dentro del código. Para lograr esto se busca transformar el código en un grafo y luego definir caminos de prueba, los cuales atraviesan como mínimo el interior de cada bucle que encuentra una vez.
Por cuestiones azarosas, nos fue instruido que el grafo ciclomático debía ser calculado de forma descendente y los caminos encontrados utilizando el método simplificado. Esto quiere decir que se comienza con un solo nodo y se recorre paso a paso expandiendo a varios nodos a medida que se van encontrando cambios de direcciones o ciclos, dándole prioridad a los ciclos. Luego al momento de definir los caminos básicos, inicialmente selecciona el camino más corto de principio a fin y luego si encuentra segmentos no recorridos y el número de camino es menor a V(G) se entra a ese camino y se busca volver al inicio de la forma más simple posible e incrementa la cantidad de caminos en 1. Este método tiende a recorrer primero los caminos de excepción, los errores, las salidas de emergencia.


34-39
59
61
66-67
76


34-39
40-42
47-49
57-58
54
59
61
62
64
66-67
69-70
74
76


34-39
40-42
44-45
47
48-49
57-58
51-52
54
55
59
61
62
64
66-67
69-70
72-73
75
76


34-39
40-42
43-45
46
47
48-49
57-58
51-52
53
54
55
59
61
62
64
66-67
69-70
72-73
75
76

Para la determinación de caminos se calcula la complejidad ciclomática de McCabe, tal que

V(G) = arcos - nodos +2

En este caso resulta en 9 caminos básicos.

Casos	Camino
1	34-39, 57-78, 59, 61,64,66-67,76
2	34-39, 40-42, 48-49, 55, 69, 61, 64, 66-67,76
3	34-39, 40-42, 43-45, 47, 40-42, 48-49, 55, 69, 61, 64, 66-67,76
4	34-39, 40-42, 43-45, 46, 47, 40-42, 48-49, 55, 69, 61, 64, 66-67,76
5	34-39, 40-42, 48-49, 51-52, 54, 48-49, 55, 69, 61, 64, 66-67,76
6	34-39, 40-42, 48-49, 51-52, 53, 54, 48-49, 55, 69, 61, 64, 66-67,76
7	34-39, 57-78, 59, 61,62,66-67,76
8	34-39, 57-78, 59, 61,64,66-67, 69-70, 75, 66-67, 76
9	34-39, 57-78, 59, 61,64,66-67, 69-70, 72-73, 75, 66-67, 76

Caminos de prueba		
Camino independiente	Parámetros	Salida esperada
1	promoPorListaDePostulantes =False Empleados = [] Empleadores = []	null
2	promoPorListaDePostulantes = True Empleados = [] Empleadores = []	null
3	promoPorListaDePostulantes = True Empleados = [] Empleadores = [empleador1]	null
4	promoPorListaDePostulantes = True Empleados = [] Empleadores = [empleador2]	?
5	promoPorListaDePostulantes = True Empleados = [empleado1] Empleadores = []	null
6	promoPorListaDePostulantes = True Empleados = [empleado2] Empleadores = []	?
7	promoPorListaDePostulantes = False	?

	Empleados = []	
	Empleadores = [empleador1]	
8	promoPorListaDePostulantes = False	null
	Empleados = [empleado1]	
	Empleadores = []	
9	promoPorListaDePostulantes = False	empleado2
	Empleados = [empleado1]	
	Empleadores = []	

Los caminos marcados en rojo resultaron imposibles de testear dado que por cuestiones propias del algoritmo, la entrada a ciertos ciclos condiciona la entrada a los próximos, causando que sea inaccesible acceder a ciertos valores. Esto genera que se obtenga una cobertura del 89%.

```java
package promo;

import java.util.HashMap;

public class UtilPromo
{
    /* El metodo aplicaPromo se utiliza para seleccionar y beneficiar a un cliente
    public Cliente aplicaPromo(boolean promoPorListaDePostulantes,
            HashMap<String, EmpleadoPretenso> empleados, HashMap<String, Empleador> empleadores)
    {
        Iterator clientes = null;
        int contadorEmpleador = 0;
        int contadorEmpleadoPretenso = 0;
        Cliente clienteBeneficiado = null;

        if (promoPorListaDePostulantes)
        {
            Iterator<Empleador> itEmpleadores = empleadores.values().iterator();
            while (itEmpleadores.hasNext())
            {
                Empleador empleador = itEmpleadores.next();
                if (empleador.getListaDePostulantes() != null)
                    contadorEmpleador += empleador.getListaDePostulantes().size();
            }
            Iterator<EmpleadoPretenso> itEmpleados = empleados.values().iterator();
            while (itEmpleados.hasNext())
            {
                EmpleadoPretenso empleadoPretenso = itEmpleados.next();
                if (empleadoPretenso.getListaDePostulantes() != null)
                    contadorEmpleadoPretenso += empleadoPretenso.getListaDePostulantes().size();
            }
        } else
        {
            contadorEmpleador = empleadores.size();
            contadorEmpleadoPretenso = empleados.size();
        }

        if (contadorEmpleador > contadorEmpleadoPretenso)
            clientes = empleadores.values().iterator();
        else
            clientes = empleados.values().iterator();

        int puntajeMaximo = Integer.MIN_VALUE;
        while (clientes.hasNext())
        {
            Cliente cl = (Cliente) clientes.next();
            if (cl.getPuntaje() > puntajeMaximo)
            {
                puntajeMaximo = cl.getPuntaje();
                clienteBeneficiado = cl;
            }
        }
        return clienteBeneficiado;
    }

}
```

Test de Integración:

Para testear la interacción entre los distintos componentes de un sistema se utilizan los test de integración. Se parte de la base (ya testeada) de que sus componentes más básicos funcionan acordemente. Las herramientas para llevar esto a cabo son los casos de uso y los casos de prueba. Los primeros cuentan la historia de cómo una persona interactúa con un sistema de soft, las rutas que el usuario puede seguir. En el segundo caso, los casos de prueba cubren el soft mas a fondo y con más detalle que un caso de uso, incluyen todas las funciones que el soft es capaz de realizar, todos los requisitos deberán ser cubiertos por estos casos. A partir del escenario de caos de uso se crearán los casos de pruebas.
Los testeos de excepciones fueron contemplados en las pruebas de integración. sus casos de prueba y baterías son explayadas a continuación en aquellos métodos que pueden arrojar excepciones.

(corte)

Métodos				
registroEmpleado(String nombreUsuario, String pass, String nombreReal, String apellido, String telefono, int edad)				
throws NewRegisterException, ImposibleCrearEmpleadoException				
Se encarga de registrar un nuevo empleado				
Parameters:				
nombreUsuario - String con el nombre de usuario del empleado				
pass - String password generada para el empleado				
nombreReal - String Nombre real del empleado que se esta registrando				
apellido - String con el apellido del empleado que esta registrando				
telefono - String con el telefono del empleado				
edad - int con la edad del empleado que se esta registrando				
Returns:				
objeto Cliente que es el empleado registrado				
Throws:				
NewRegisterException - se lanza para indicar que el nombre de usuario ya esta en uso.				
ImposibleCrearEmpleadoException - Se lanza en caso de que nombreUsuario, pass, nombreReal, apellido, telefono sean nulos				
Escenario 1: Todas las listas están vacias o el empleado a registrar no está repetido				
Dato de Entrada	**Clase de equivalencia**	**Aplica?**	**Motivo**	**Id. de clase de equivalencia:**

nombreUsuario	nombreUsuario != null	Si	Cumple contrato	1.1
	nombreUsuario == null	No	No cumple contrato	1.2
pass	pass != null	Si	Cumple contrato	2.1
	pass == null	No	No cumple contrato	2.2
nombreReal	nombreReal != null	Si	Cumple contrato	3.1
	nombreReal == null	No	No cumple contrato	3.2
apellido	apellido != null	Si	Cumple contrato	4.1
	apellido == null	No	No cumple contrato	4.2
telefono	telefono != null	Si	Cumple contrato	5.1
	telefono == null	No	No cumple contrato	5.2
edad	edad> 0	Si	Cumple contrato	6.1
	edad <= 0	No	No cumple contrato	6.2

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	clases que abarca
1	nombreUsuario	"Juan123"	nueva instancia de EmpleadoPretenso	1.1
	pass	"Juan123"		2.1
	nombreReal	"Juan"		3.1
	apellido	"Rodriguez"		4.1
	telefono	"2235698547"		5.1
	edad	25		6.1
2	nombreUsuario	null	ImposibleCrearEmpleadoException	1.2
	pass	null		2.2
	nombreReal	null		3.2
	apellido	null		4.2
	telefono	null		5.2
	edad	-24		6.2

Escenario 2: Hay elementos en la lista de empleados y el nombre de usuario del empleado a registrar ya existe

Dato de Entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
nombreUsuario	nombreUsuario != null	Si	Cumple contrato	1.1
	nombreUsuario == null	No	No cumple contrato	1.2
pass	pass != null	Si	Cumple contrato	2.1

	pass == null	No	No cumple contrato	2.2
nombreReal	nombreReal != null	Si	Cumple contrato	3.1
	nombreReal == null	No	No cumple contrato	3.2
apellido	apellido != null	Si	Cumple contrato	4.1
	apellido == null	No	No cumple contrato	4.2
telefono	telefono != null	Si	Cumple contrato	5.1
	telefono == null	No	No cumple contrato	5.2
edad	edad> 0	Si	Cumple contrato	6.1
	edad <= 0	No	No cumple contrato	6.2

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	clases que abarca
1	nombreUsuario	"Juan123"	NewRegisterException	1.1
	pass	"Juan123"		2.1
	nombreReal	"Juan"		3.1
	apellido	"Dominguez"		4.1
	telefono	"2235698547"		5.1
	edad	25		6.1

registroEmpleador(String nombreUsuario, String pass, String nombreReal, String telefono, String tipoPersona, String rubro)

throws NewRegisterException, ImposibleCrearEmpleadorException

Se encarga de registrar un nuevo empleado

Parameters:

nombreUsuario - String nombre de usuario para el empleador que se quiere registrar

pass - String password para el empleador que se esta registrando

nombreReal - String nombre real del empleador que se esta registrando

telefono - String telefono del empleador que se esta registrando

tipoPersona - String tipo de persona: Fisica o Juridica para el empleador que se esta registrando

rubro - String rubro del empleador que se esta registrando

Returns:

objeto Cliente que es el empleador registrado

Throws:

NewRegisterException - se lanza para indicar que el nombre de usuario ya esta en uso.

ImposibleCrearEmpleadorException - lanza una excepcion ImposibleCrearEmpleadorException en caso de que nombreUsuario, pass, nombreReal, telefono, tipoPersona, o rubro sean nulos o en caso de que el tipo de persona no sea "JURIDICA" ni "FISICA"o en caso de que rubro no sea "SALUD" "COMERCIO LOCAL" "COMERCIO INTERNACIONAL" (ver clase Constantes)

Escenario 1: Todas las listas están vacias o el empleador a registrar no está repetido				
Dato de Entrada	**Clase de equivalencia**	**Aplica?**	**Motivo**	**Id. de clase de equivalencia:**
nombreUsuario	nombreUsuario != null	Si		1.1
	nombreUsuario == null	No	No cumple contrato	1.2
pass	pass != null	Si		2.1
	pass == null	No	No cumple contrato	2.2
nombreReal	nombreReal != null	Si		3.1
	nombreReal == null	No	No cumple contrato	3.2
telefono	telefono != null	Si		4.1
	telefono == null	No	No cumple contrato	4.2
tipoPersona	tipoPersona = "JURIDICA" o tipoPersona = "FISICA"	Si		5.1
	tipoPersona = null o cualquier otro string	No	No cumple contrato	5.2
rubro	rubro = "SALUD" o "COMERCIO_INTERNACIONAL" o "COMERCIO LOCAL"	Si		6.1
	rubro = null o cualquier otro string	No	No cumple contrato	6.2

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	clases que abarca
1	nombreUsuario	"Juan123"	nueva instancia de Empleador	1.1
	pass	"Juan123"		2.1
	nombreReal	"Juan"		3.1
	telefono	"2235698547"		4.1
	tipoPersona	"FISICA"		5.1
	rubro	"SALUD"		6.1
2	nombreUsuario	null	ImposibleCrearEmpleadorException	1.2
	pass	null		2.2

	nombreReal	null		3.2
	telefono	null		4.2
	TipoPersona	null		5.2
	rubro	"POLIRUBRO"		6.2

Escenario 2: Todas las listas contienen elementos y el nombre de usuario del empleador a registrar ya existe

Dato de Entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
nombreUsuario	nombreUsuario != null	Si		1.1
	nombreUsuario == null	No	No cumple contrato	1.2
pass	pass != null	Si		2.1
	pass == null	No	No cumple contrato	2.2
nombreReal	nombreReal != null	Si		3.1
	nombreReal == null	No	No cumple contrato	3.2
telefono	telefono != null	Si		4.1
	telefono == null	No	No cumple contrato	4.2
tipoPersona	tipoPersona = "JURIDICA" o tipoPersona = "FISICA"	Si		5.1
	tipoPersona = null o cualquier otro string	No	No cumple contrato	5.2
rubro	rubro = "SALUD" o "COMERCIO_INTERNACIONAL" o "COMERCIO LOCAL"	Si		6.1
	rubro = null o cualquier otro string	No	No cumple contrato	6.2

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	clases que abarca
1	nombreUsuario	"Juan123"	NewRegisterException	1.1
	pass	"Juan123"		2.1
	nombreReal	"Juan"		3.1
	telefono	"2235698547"		4.1
	tipoPersona	"FISICA"		5.1
	rubro	"SALUD"		6.1

login(String nombreUsuario, String pass)				
throws ContraException, NombreUsuarioException				
Se encarga de realizar el inicio de sesion de un usuario en la aplicacion, ya sean empleados, empleadores o administradores Si el inicio de sesion es exitoso, el usuario quedara logeado en la aplicacion y el atributo tipo de usuario sera seteado: 0 para Empleado, 1 para Empleador, 2 para Administrador				
Parameters:				
nombreUsuario - String: nombre de usuario de quien se quiere loguear				
pass - String: contrasena del usuario que se quiere loguear				
Returns:				
Se retorna un objeto Usuario con el objeto usuario				
Throws:				
ContraException - si la contrasena no es correcta se lanza una excepcion del tipo ContraException				
NombreUsuarioException - si no se encontro un empleado, un empleador o un administrador con el nombreUsuario proporcionado, se lanza una excepcion NombreUsuarioException				
Pre: nombre de usuario y contrasena distintos de null				
Escenario 1:				
Hay un EmpleadoPretenso cargado con usserName "Juan123" y pass "Juan123"				

Dato de Entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
nombreUsuario	nombreUsuario existente en las listas	Si	Cumple contrato	1.1
	nombreUsuario inexistente en las listas	Si	Cumple contrato	1.2
	nombreUsuario == null	No	No cumple contrato	~~1.3~~
pass	pass correcta	Si	Cumple contrato	2.1
	pass incorrecta	Si	Cumple contrato	2.2
	pass == null	No	No cumple contrato	~~2.3~~

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	clases que abarca
1	nombreUsuario	"Juan123"	agencia.getTipoUsuario = 0 y devuelve el Usuario corr.	1.1
	pass	"Juan123"		2.1
2	nombreUsuario	"Juan123"	ContraException	1.1

	pass	"qwerty123"		2.2
3	nombreUsuario	"Alejandro"	NombreUsuarioException	1.2
	pass	"Juan123"		2.1

Escenario 2:				
Hay un Empleador en las listas con usserName ""Juan123" y pass "Juan123"				
Dato de Entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
nombreUsuario	nombreUsuario existente en las listas	Si	Cumpe contrato	1.1
	nombreUsuario inexistente en las listas	Si	Cumpe contrato	1.2
	nombreUsuario == null	No	No cumple contrato	~~1.3~~
pass	pass correcta	Si	Cumple contrato	2.1
	pass incorrecta	Si	Cumple contrato	2.2
	pass == null	No	No cumple contrato	~~2.3~~

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	clases que abarca
1	nombreUsuario	"HPC"	agencia.getTipoUsuario = 1 y devuelve el Usuario corr.	1.1
	pass	"privadoComun"		2.1
2	nombreUsuario	"HPC"	ContraException	1.1
	pass	"qwerty123"		2.2
3	nombreUsuario	"Alejandro"	NombreUsuarioException	1.2
	pass	"privadoComun"		2.1

setLimitesRemuneracion(int limiteInferior, int limiteSuperior)

throws LimiteSuperiorRemuneracionInvalidaException, LimiteInferiorRemuneracionInvalidaException

Pameters:

limiteInferior - int

limiteSuperior - int

Returns:

void

Throws				
LimiteSuperiorRemuneracionInvalidaException - Se lanza la excepcion si limite superior es menor que limite superior				
LimiteInferiorRemuneracionInvalidaException - Se lanza la excepcion si limite inferior es menor a cero				
Dato de Entrada	**Clase de equivalencia**	**Aplica?**	**Motivo**	**Id. de clase de equivalencia:**
limiteInferior	**limiteInferior <= 0**	**Si**		**1.1**
	limiteInferior > 0	**Si**		**1.2**
limiteSuperior	**limiteSuperior <=0**	**Si**		**2.1**
	limiteSuperior >0	**Si**		**2.2**

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida Esperada	clases que abarca
1	limiteInferior	-5	LimiteInferiorRemuneracionInvalidaException	1.1
	limiteSuperior	-2		2.1
2	limiteInferior	20	LimiteSuperiorRemuneracionInvalidaException	1.1
	limiteSuperior	6		2.2
3	limiteInferior	150	This.limiteInferior = 150	1.2
	limiteSuperior	180	This.limiteSuperior = 180	2.1

eliminarTicket(int limiteInferior, int limiteSuperior)

Elimina el ticlet del usuario logueado en

pre:

Hay un cliente logueado en la aplicacion				
Dato de Entrada	**Clase de equivalencia**	**Aplica?**	**Motivo**	**Id. de clase de equivalencia:**
-	**estado de contratacion**	**Si**		**1.1**
	estado de contratacion	**Si**		**1.2**

CrearTicketEmpleado(String locacion, int remuneracion, String jornada, String puesto, String experiencia, String estudios, Cliente cliente)

Se encarga de crear un nuevo ticket para un empleado con los datos enviados por parametros. Si el Empleado tiene un ticket activo, este sera eliminado.
Pre:
El cliente debe ser un empleado
Parameters:
locacion - String con la locacion del empleado
remuneracion - int con la remuneracion pretendida por el empleado
jornada - String con la jornada de preferencia del empleado
puesto - String tipo de puesto al que aspira el empleado
experiencia - String experiencia del empleado
estudios - String estudios que posee el empleado
cliente - objeto cliente del ticket
Throws:
ImposibleModificarTicketsException - lanza una excepcion ImposibleModificarTicketsException si el estado de contratacion no esta en un estado valido

Escenario 1: El estadoContratacion es false y el empleado pretenso es				
usserName = "Juan123", pass="Juan123", realName="Juan123", apellido="Rodriguez", telefono="2235698547", edad = 25				
Dato de entrada	**Clase de equivalencia**	**Aplica?**	**Motivo**	**Id. de clase de equivalencia:**
locacion	**locacion == null**	**No**	**No cumple contrato**	
	locacion == "HOME_OFFICE" o "PRESENCIAL" o "INDIFERENTE"	**Si**	**Cumple contrato**	**1.1**
remuneracion	**Remuneracion < 0**	**No**	**No cumple contrato**	
	Remuneracion < V1	**Si**	**Cumple contrato**	**2.1**
	Remuneracion >V1 & Remuneracion < V2	**Si**	**Cumple contrato**	**2.2**
	Remuneracion > V2	**Si**	**Cumple contrato**	**2.3**
jornada	**jornada == null**	**No**	**No cumple contrato**	
	jornada == "JORNADA_COMPLETA" o "JORNADA_MEDIA" o "JORNADA_EXTENDIDA"	**Si**	**Cumple contrato**	**3.1**
	jornada == "" || jornada == cualquiera	**No**	**No cumple contrato**	

puesto	puesto == null	No	No cumple contrato	
	puesto == "JUNIOR", 'SENIOR' O "GERENCIAL"	Si	Cumple contrato	4.1
experiencia	experiencia == null	No	No cumple contrato	
	experiencia == "EXP_NADA" o "EXP_MEDIA" o "EXP_MUCHA"	Si	Cumple contrato	5.1
	experiencia == "" o experiencia == cualquier otro	No	No cumple contrato	
estudios	estudios == null	No	No cumple contrato	
	estudios == "PRIMARIOS", "SECUNDARIOS" O "TERCIARIOS"	Si	Cumple contrato	6.1
	estudios == "" || estudios == "cualquiera"	No	No cumple contrato	

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	locacion	"HOME_OFFICE"	El cliente tiene un ticket nuevo	1.1
	remuneracion	1.200,00 €		
	jornada	"JORNADA_COMPLETA"		3.1
	puesto	"JUNIOR"		4.1
	experiencia	"EXP_NADA"		5.1
	estudios	"PRIMARIOS"		6.1
	cliente	EmpleadoPretenso establecido		-
Escenario 2: El estadoContratacion es true y el cliente es				
usserName = "Juan123", pass="Juan123", realName="Juan123", apellido="Rodriguez", telefono="2235698547", edad = 25				

Dato de entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
locacion	locacion == null	No	No cumple contrato	

	locacion == "HOME_OFFICE" o "PRESENCIAL" o "INDIFERENTE"	Si	Cumple contrato	1.1
remuneracion	Remuneracion < 0	No	No cumple contrato	
	Remuneracion < V1	Si	Cumple contrato	2.1
	Remuneracion >V1 & Remuneracion < V2	Si	Cumple contrato	2.2
	Remuneracion > V2	Si	Cumple contrato	2.3
jornada	jornada == null	No	No cumple contrato	
	jornada == "JORNADA_COMPLETA" o "JORNADA_MEDIA" o "JORNADA_EXTENDIDA"	Si	Cumple contrato	3.1
	jornada == "" || jornada == cualquiera	No	No cumple contrato	
puesto	puesto == null	No	No cumple contrato	
	puesto == "JUNIOR", 'SENIOR' O "GERENCIAL"	Si	Cumple contrato	4.1
experiencia	experiencia == null	No	No cumple contrato	
	experiencia == "EXP_NADA" o "EXP_MEDIA" o "EXP_MUCHA"	Si	Cumple contrato	5.1
	experiencia == "" o experiencia == cualquier otro	No	No cumple contrato	
estudios	estudios == null	No	No cumple contrato	
	estudios == "PRIMARIOS", "SECUNDARIOS" O "TERCIARIOS"	Si	Cumple contrato	6.1
	estudios == "" || estudios == "cualquiera"	No	No cumple contrato	

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca

1	locacion	"HOME_OFFICE"	ImposibleModificarTicketsException	1.1
	remuneracion	1.200,00 €		
	jornada	"JORNADA_COMPLETA"		3.1
	puesto	"JUNIOR"		4.1
	experiencia	"EXP_NADA"		5.1
	estudios	"PRIMARIOS"		6.1
	cliente	EmpleadoPretenso establecido		-

CrearTicketEmpleador(String locacion, int remuneracion, String jornada, String puesto, String experiencia, String estudios, Cliente cliente)

Se encarga de crear un nuevo ticket para un empleado con los datos enviados por parametros. Si el Empleado tiene un ticket activo, este sera eliminado.

Pre:

El cliente debe ser un empleado

Parameters:

locacion - String con la locacion del empleado

remuneracion - int con la remuneracion pretendida por el empleado

jornada - String con la jornada de preferencia del empleado

puesto - String tipo de puesto al que aspira el empleado

experiencia - String experiencia del empleado

estudios - String estudios que posee el empleado

cliente - objeto cliente del ticket

Throws:

ImposibleModificarTicketsException - lanza una excepcion ImposibleModificarTicketsException si el estado de contratacion no esta en un estado valido

Escenario 1: El estadoContratacion es false y el empleado pretenso es

usserName = "Juan123", pass="Juan123", realName="Juan", telefono="2235698547", rubro="SALUD", tipoPersona="FISICA"

Dato de entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
locacion	locacion == null	No	No cumple contrato	
	locacion == "HOME_OFFICE" o	Si	Cumple contrato	1.1

	"PRESENCIAL" o "INDIFERENTE"			
remuneracion	Remuneracion < 0	No	No cumple contrato	
	Remuneracion < V1	Si	Cumple contrato	2.1
	Remuneracion >V1 & Remuneracion < V2	Si	Cumple contrato	2.2
	Remuneracion > V2	Si	Cumple contrato	2.3
jornada	jornada == null	No	No cumple contrato	
	jornada == "JORNADA_COMPLETA" o "JORNADA_MEDIA" o "JORNADA_EXTENDIDA"	Si	Cumple contrato	3.1
	jornada == "" || jornada == cualquiera	No	No cumple contrato	
puesto	puesto == null	No	No cumple contrato	
	puesto == "JUNIOR", 'SENIOR' O "GERENCIAL"	Si	Cumple contrato	4.1
experiencia	experiencia == null	No	No cumple contrato	
	experiencia == "EXP_NADA" o "EXP_MEDIA" o "EXP_MUCHA"	Si	Cumple contrato	5.1
	experiencia == "" o experiencia == cualquier otro	No	No cumple contrato	
estudios	estudios == null	No	No cumple contrato	
	estudios == "PRIMARIOS", "SECUNDARIOS" O "TERCIARIOS"	Si	Cumple contrato	6.1
	estudios == "" || estudios == "cualquiera"	No	No cumple contrato	

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	locacion	"HOME_OFFICE"	El cliente tiene un ticket nuevo	1.1
	remuneracion	1.200,00 €		

	jornada	"JORNADA_COMPLETA"		3.1
	puesto	"JUNIOR"		4.1
	experiencia	"EXP_NADA"		5.1
	estudios	"PRIMARIOS"		6.1
	cliente	EmpleadoPretenso establecido		-

getContratacionEmpleadoPretenso(EmpleadoPretenso ep)

Busca la contratacion asociada a un EmpleadoPretenso especifico en la lista de contrataciones de la agencia.

Parameters

ep - el Objeto EmpleadoPretenso de quien se quiere obtener la contratacion

Returns:

objeto Cliente con el Empleador encontrado si no se encontro ninguna contratacion que involucre al EmpleadoPretenso especificado.

Pre: empleado distinto de null y registrado en el sistema

Escenario 1: El empleado pretenso esta cargado en el sistema, y tiene contratacion con un empleador

Dato de entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
ep	ep != null			1
	ep == null	No	No cumple contrato	

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	ep	-	Empleador asociado en Contratacion	1

getContratacionEmpleador(Empleador empleador)

Busca la contratacion asociada a un Empleador especifico en la lista de contrataciones de la agencia.

Parameters

empleador - el Objeto Empleador de quien se quiere obtener la contratacion

Returns:

objeto Usuario con la contratacion.

Pre: empleador distinto de null y registrado en el sistema

Escenario 1: El empleador esta cargado en el sistema y tiene una contratacion con un empleado

Dato de entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
empleador	empleador != null			1
	empleador == null	No	No cumple contrato	

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	empleador	-	Empleado asociado con el empleador	1

calculaPremiosCastigosAsignaciones()

Este metodo otorga premios y castigos a los clientes que participan en el proceso de seleccion.

Pre: Las listas de postulantes estan ordenadas

Dato de entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
-	-	-	-	-

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	-	-	Puntajes de empleados y empleadores actualizados	

match(Empleador empleador, EmpleadoPretenso empleado)

Esta relacionado con la ocurrencia de un "match" entre un empleador y un empleado pretenso, lo que implica que un empleado potencial ha sido seleccionado por un empleador. Realiza ajustes en los puntajes y las comisiones, y marca a ambos clientes como no disponibles para futuros matches o contrataciones.

Pre: Empleado y empleador deben ser validos y estar registrados en el sistema

Crea una nueva instancia de Contratacion que representa la contratacion o emparejamiento entre el empleador y el empleado. Incrementa el puntaje del empleado en 10 puntos Incrementa el puntaje del empleador en 50 puntos Calcula las comisiones para el empleado y el empleador en funcion del Ticket asociado al empleador. Empleado y empleador ya no estan disponibles para futuros emparejamientos o contrataciones.

Parameters:

empleador - : Objeto empleador con quien el empleado pretenso hace match

empleado - : Objeto empleado con quien el empleador hace match

Post:

Los tickets del Empleado y del emepleador se eliminan sin generar penalizaciones.

Escenario 1: Existe un solo empleador y empleado en la agencia para asegurar que haya un match

Dato de entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
empleador	empleador==null o no registrado	No	No cumple contrato	
	empleador != null y registrado	Si		1
empleado	empleado==null o no registrado	No	No cumple contrato	
	empleado != null y registrado	Si		2

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	empleador	-	Creada nueva instancia de Contratacion y agregada a la lista de contrataciones	1
	empleado	-		2

generaPostulantes()

Este metodo se encarga de calcular y asignar listas de postulantes a los Empleadores y EmpleadoPretenso en funcion de sus Ticket y las comparaciones de puntajes entre ellos.

Pre: Listas de empleados y empleadores con elementos validos

Establece la lista de EmpleadoPretenso ordenados de mayor a menor como la lista de postulantes para el Empleador Establece una lista de Empleadores ordenada por puntaje de mayor a menor de compatibilidad y la establece como la lista de postulantes para ese EmpleadoPretenso.

Escenario 1: El admin esta logueado y existe un solo empleador y empleado en la agencia para forzar la creación

Dato de entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
-	-	Si	-	-

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca

1	-	-	Lista de postulantes generada para el empleador y el empleado	

cerrarSesion()

Se utiliza para cerrar la sesion de un usuario en la aplicacion. Despues de ejecutar el metodo, no habra un usuario logeado en la aplicacion

Escenario 1: Hay un usuario logueado, agencia.getTipoUsuario != -1

Dato de entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
-	-	Si	-	-

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	-	-	agencia.getTipoUsuario = -1	-

getEstado()

Returns:

un String dependiendo del estado en que se encuentre la Agencia, estos pueden ser:

Mensajes.AGENCIA_EN_CONTRATACION.getValor()

Mensajes.AGENCIA_EN_BUSQUEDA.getValor()

Escenario 1: el estado de contratacion es true

Dato de entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
-	-	Si	-	-

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	-	-	Mensajes.AGENCIA_EN_CONTRATACION.getValor()	-

Escenario 2: el estado de contratacion es false

Dato de entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
-	-	Si	-	-

Batería de Pruebas				
Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	-	-	Mensajes.AGENCIA_EN_BUSQUEDA.getValor()	-

Class Ticket:

getComparacionLocacion(Ticket otro)				
Pre: el parametro otro es diferente de null				
Escenario 1: (this.locacion == "PRESENCIAL")				
Dato de Entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
otro	otro == null	No	No cumple contrato	-
	otro == "PRESENCIAL"	Si	Cumple contrato	1
	otro == "HOME_OFFICE"	Si	Cumple contrato	2
	otro == "INDIFERENTE"	Si	Cumple contrato	3
Batería de Pruebas				
Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	otro.locacion	"PRESENCIAL"	1	1
2	otro.locacion	"HOME_OFFICE"	-1	2
3	otro.locacion	"INDIFERENTE"	-1	3
Escenario 2: (this.locacion == "HOME_OFFICE")				
Dato de Entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
otro	otro == null	No	No cumple contrato	-
	otro == "PRESENCIAL"	Si	Cumple contrato	1
	otro == "HOME_OFFICE"	Si	Cumple contrato	2
	otro == "INDIFERENTE"	Si	Cumple contrato	3
Batería de Pruebas				
Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	otro.locacion	"PRESENCIAL"	-1	1
2	otro.locacion	"HOME_OFFICE"	1	2

3	otro.locacion	"INDIFERENTE"	1	3

Escenario 3: (this.locacion == "INDIFERENTE")				
Dato de Entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
otro	otro == null	No	No cumple contrato	-
	otro == "PRESENCIAL"	Si	Cumple contrato	1
	otro == "HOME_OFFICE"	Si	Cumple contrato	2
	otro == "INDIFERENTE"	Si	Cumple contrato	3

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	otro.locacion	"PRESENCIAL"	-1	1
2	otro.locacion	"HOME_OFFICE"	1	2
3	otro.locacion	"INDIFERENTE"	1	3

getComparacionJornada(Ticket otro)

Pre: el parametro otro es diferente de null

Escenario 1: (this.jornada== "MEDIA")				
Dato de Entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
otro	otro == null	No	No cumple contrato	-
	otro.jornada == media	Si	Cumple contrato	1
	otro.jornada == completa	Si	Cumple contrato	2
	otro.jornada == extendida	Si	Cumple contrato	3

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	otro.jornada	"MEDIA"	1	1
2	otro.jornada	"COMPLETO"	-0,5	2
3	otro.jornada	"EXTENDIDA"	-1	3

Escenario 2: (this.jornada== "COMPLETA")				
Dato de Entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
otro	otro == null	No	No cumple contrato	-
	otro.jornada == media	Si	Cumple contrato	1

	otro.jornada == completa	Si	Cumple contrato	2
	otro.jornada == extendida	Si	Cumple contrato	3

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	otro.jornada	"MEDIA"	-0,5	1
2	otro.jornada	"COMPLETO"	1	2
3	otro.jornada	"EXTENDIDA"	-0,5	3

Escenario 3: (this.jornada== "EXTENDIDA")

Dato de Entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
otro	otro == null	No	No cumple contrato	-
	otro.jornada == media	Si	Cumple contrato	1
	otro.jornada == completa	Si	Cumple contrato	2
	otro.jornada == extendida	Si	Cumple contrato	3

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	otro.jornada	"MEDIA"	-1	1
2	otro.jornada	"COMPLETO"	1	2
3	otro.jornada	"EXTENDIDA"	1	3

getComparacionPuesto(Ticket otro)

Pre: el parametro otro es diferente de null

Escenario 1: (this.puesto== "JUNIOR")

Dato de Entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
otro	otro == null	No	No cumple contrato	-
	otro.puesto== junior	Si	Cumple contrato	1
	otro.puesto== senior	Si	Cumple contrato	2
	otro.puesto== managment	Si	Cumple contrato	3

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	otro.puesto	"JUNIOR"	1	1
2	otro.puesto	"SENIOR"	-0,5	2
3	otro.puesto	"MANAGMENT"	-1	3

Escenario 2: (this.puesto== "SENIOR")				
Dato de Entrada	**Clase de equivalencia**	**Aplica?**	**Motivo**	**Id. de clase de equivalencia:**
otro	**otro == null**	**No**	**No cumple contrato**	**-**
	otro.puesto== junior	**Si**	**Cumple contrato**	**1**
	otro.puesto== senior	**Si**	**Cumple contrato**	**2**
	otro.puesto== managment	**Si**	**Cumple contrato**	**3**

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	**otro.puesto**	**"JUNIOR"**	**-0,5**	**1**
2	**otro.puesto**	**"SENIOR"**	**1**	**2**
3	**otro.puesto**	**"MANAGMENT"**	**-0,5**	**3**

Escenario 3: (this.puesto== "MANAGMENT")				
Dato de Entrada	**Clase de equivalencia**	**Aplica?**	**Motivo**	**Id. de clase de equivalencia:**
otro	**otro == null**	**No**	**No cumple contrato**	**-**
	otro.puesto== junior	**Si**	**Cumple contrato**	**1**
	otro.puesto== senior	**Si**	**Cumple contrato**	**2**
	otro.puesto== managment	**Si**	**Cumple contrato**	**3**

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	**otro.puesto**	**"JUNIOR"**	**-1**	**1**
2	**otro.puesto**	**"SENIOR"**	**1**	**2**
3	**otro.puesto**	**"MANAGMENT"**	**1**	**3**

getComparacionEstudios(Ticket otro)

Pre: el parametro otro es diferente de null

Escenario 1: (this.puesto== "PRIMARIO")				
Dato de Entrada	**Clase de equivalencia**	**Aplica?**	**Motivo**	**Id. de clase de equivalencia:**
otro	**otro == null**	**No**	**No cumple contrato**	**-**
	otro == primario	**Si**	**Cumple contrato**	**1**
	otro == secundario	**Si**	**Cumple contrato**	**2**
	otro == terciario	**Si**	**Cumple contrato**	**3**

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	otro.estudios	"PRIMARIO"	1	1
2	otro.estudios	"SECUNDARIO"	-0,5	2
3	otro.estudios	"TERCIARIO"	-2	3

Escenario 2: (this.puesto== "SECUNDARIO")

Dato de Entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
otro	otro == null	No	No cumple contrato	-
	otro == primario	Si	Cumple contrato	1
	otro == secundario	Si	Cumple contrato	2
	otro == terciario	Si	Cumple contrato	3

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	otro.estudios	"PRIMARIO"	1,5	1
2	otro.estudios	"SECUNDARIO"	1	2
3	otro.estudios	"TERCIARIO"	-1,5	3

Escenario 3: (this.puesto== "TERCIARIO")

Dato de Entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
otro	otro == null	No	No cumple contrato	-
	otro == primario	Si	Cumple contrato	1
	otro == secundario	Si	Cumple contrato	2
	otro == terciario	Si	Cumple contrato	3

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	otro.estudios	"PRIMARIO"	2	1
2	otro.estudios	"SECUNDARIO"	1,5	2
3	otro.estudios	"TERCIARIO"	1	3

getComparacionExperiencia(Ticket otro)

Pre: el parametro otro es diferente de null

Escenario 1: (this.experiencia== "NADA")

Dato de Entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
otro	otro == null	No	No cumple contrato	-
	otro == nada	Si	Cumple contrato	1
	otro == media	Si	Cumple contrato	2
	otro == mucha	Si	Cumple contrato	3

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	otro.experiencia	"NADA"	1	1
2	otro.experiencia	"MEDIA"	-0,5	2
3	otro.experiencia	"MUCHA"	-2	3

Escenario 2: (this.experiencia== "MEDIA")

Dato de Entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
otro	otro == null	No	No cumple contrato	-
	otro == nada	Si	Cumple contrato	1
	otro == media	Si	Cumple contrato	2
	otro == mucha	Si	Cumple contrato	3

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	otro.experiencia	"NADA"	1,5	1
2	otro.experiencia	"MEDIA"	1	2
3	otro.experiencia	"MUCHA"	-1,5	3

Escenario 3: (this.experiencia== "MUCHA")

Dato de Entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
otro	otro == null	No	No cumple contrato	-
	otro == nada	Si	Cumple contrato	1
	otro == media	Si	Cumple contrato	2
	otro == mucha	Si	Cumple contrato	3

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	otro.experiencia	"NADA"	2	1
2	otro.experiencia	"MEDIA"	1,5	2
3	otro.experiencia	"MUCHA"	1	3

getComparacionRemuneracion(Ticket otro)

Pre: el parametro otro es diferente de null

Escenario 1: (this.remuneracion < V1)

Dato de Entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
otro	otro < 0	No	No cumple contrato	-
	otro.remuneracion < V1	Si	Cumple contrato	1
	otro.remuneracion ∈[V1,V2]	Si	Cumple contrato	2
	otro.remuneracion > V2	Si	Cumple contrato	3

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	otro.remuneracion	<V1	1	1
2	otro.remuneracion	∈[V1,V2]	1	2
3	otro.remuneracion	>V2	1	3

Escenario 2: (this.remuneracion ∈[V1,V2])

Dato de Entrada	Clase de equivalencia	Aplica?	Motivo	Id. de clase de equivalencia:
otro	otro < 0	No	No cumple contrato	-
	otro.remuneracion < V1	Si	Cumple contrato	1
	otro.remuneracion ∈[V1,V2]	Si	Cumple contrato	2
	otro.remuneracion > V2	Si	Cumple contrato	3

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	otro.remuneracion	<V1	-0,5	1
2	otro.remuneracion	∈[V1,V2]	1	2
3	otro.remuneracion	>V2	1	3

Escenario 3: (this.remuneracion > V2)				
Dato de Entrada	**Clase de equivalencia**	**Aplica?**	**Motivo**	**Id. de clase de equivalencia:**
otro	**otro < 0**	**No**	**No cumple contrato**	**-**
	otro.remuneracion < V1	**Si**	**Cumple contrato**	**1**
	otro.remuneracion ∈[V1,V2]	**Si**	**Cumple contrato**	**2**
	otro.remuneracion > V2	**Si**	**Cumple contrato**	**3**

Batería de Pruebas

Número de prueba	Datos de entrada	Valor	Salida esperada	Clases que abarca
1	**otro.remuneracion**	**<V1**	**-1**	**1**
2	**otro.remuneracion**	**∈[V1,V2]**	**-0,5**	**2**
3	**otro.remuneracion**	**>V2**	**1**	**3**

Empleado pretenso:

calculaComision(Ticket ticket)

Este metodo se utiliza para calcular la comision que un usuario debe recibir segun el tipo de puesto en un Ticket y su puntaje (ver detalles de calculo de comision).

Pre: El metodo requiere un objeto Ticket valido como entrada para realizar los calculos de comision.

Escenario 1	this.rubro == "Salud"			

Tabla de Particiones en Clases de Equivalencia

Dato de entrada	Clases de equivalencia	Aplica ?	Motivo	Identificador de clase de equivalencia
ticket	ticket != null	Si	Cumple contrato	1
	ticket == null	No	No cumple contrato	-

Batería de Pruebas

número de prueba	Datos de entrada	Valor	Salida Esperada	clases que abarca
	ticket.remuneracion	250.000,00 €	150.000,00 €	1

Escenario 2	this.rubro == "comercio_local"			

Tabla de Particiones en Clases de Equivalencia

Dato de entrada	Clases de equivalencia	Aplica ?	Motivo	Identificador de clase de equivalencia

ticket	ticket != null	Si	Cumple contrato	1
	ticket == null	No	No cumple contrato	-

Batería de Pruebas

número de prueba	Datos de entrada	Valor	Salida Esperada	clases que abarca
	ticket.remuneracion	250.000,00 €	175.000,00 €	1

Escenario 3	this.rubro == "comercio_internacional"		

Tabla de Particiones en Clases de Equivalencia

Dato de entrada	Clases de equivalencia	Aplica ?	Motivo	Identificador de clase de equivalencia
ticket	ticket != null	Si	Cumple contrato	1
	ticket == null	No	No cumple contrato	-

Batería de Pruebas

número de prueba	Datos de entrada	Valor	Salida Esperada	clases que abarca
	ticket.remuneracion	250.000,00 €	200.000,00 €	1

Casos de uso

- **Registrar Usuario**
  Descripción:
  Un usuario quiere registrarse en el sistema usando sus datos personales.
  Actores:
  Un usuario (empleador o empleado pretenso).
  Flujo normal:
  1) El usuario accede al sistema.
  2) El usuario presiona el botón “Registrar”.
  3) El usuario elige su nombre de usuario, contraseña y proporciona su nombre real y teléfono.
  4) El usuario indica si es un empleador o un empleado pretenso.
  5) El usuario ingresa sus datos específicos según la elección anterior y presiona el botón “Registrar”.

  Excepciones:
  1) NewRegisterExpection: El nombre de usuario ya está en uso.
  2) ImposibleCrear..Exception: Alguno de los valores introducidos fue nulo.

  Post-condiciones
  1) El usuario queda registrado en el sistema.

- **Login**

  Descripción

  Un usuario quiere loguearse al sistema, para ello usa su nombre de usuario y contraseña.

  Actores

  Un usuario (empleador o empleado pretenso) a través de la interfaz de usuario.

  Pre-condiciones

  El usuario debe estar registrado en el sistema.

  Flujo normal:

  1) El usuario ingresa al sistema.
  2) El usuario ingresa su nombre de usuario y contraseña.
  3) El usuario presiona el botón “Login”.

  Excepciones:

  1) ContraException: La contraseña introducida fue la incorrecta
  2) NombreUsuarioException: El nombre de usuario no se encuentra en el sistema.

  Post-condiciones:

  1) El usuario queda logueado en el sistema.

- **Modificar límites de remuneración:**

  Descripción:

  Un usuario tipo administrador establece nuevos límites de remuneración.

  Actores:

  Un usuario tipo Admin

  Pre-condiciones

  El usuario administrador debe estar logueado en el sistema.

  Flujo normal:

  1) El administrador accede al sistema.
  2) El administrador llena los campos de límite inferior y superior
  3) El administrador presiona el botón “Cambiar”

  Excepciones:

  1) LimiteSuperiorRemuneracionInvalidaException: El límite superior es menor al límite inferior.
  2) LimiteInferiorRemuneracionInvalidaException: El límite inferior es menor a cero.

  Post-condiciones:

  1) El límite inferior y superior fue cambiado a los valores establecidos por el administrador.

- **Gatillar ronda**

  Descripción:

  Un usuario tipo Admin comienza el proceso de ronda de contrataciones.

  Actores:

  Administrador, empleado pretenso y empleador

  Pre-condiciones:

  El usuario administrador debe estar logueado en el sistema.

  Flujo normal:

  1) El administrador accede al sistema.
  2) El administrador presiona el botón “Gatillar ronda”

  Post-condiciones:

1) Se producen *matcheos* entre los empleados y empleadores registrados en el sistema.
2) Se penaliza el puntaje de los empleadores que no tuvieron ninguna contratación (en caso que haya).
3) Se generan nuevos postulantes para cada empleado y empleador.
4) El sistema está preparado para la ronda de contrataciones.
5) Se invierte el estado de contratación del sistema.

- **Nuevo ticket**
  Descripción:
  Un usuario tipo empleador o empleado pretenso selecciona la opción de agregar ticket
  Actores:
  Empleado pretenso o empleador
  Pre-condiciones:
  El usuario debe estar logueado en el sistema.
  Flujo normal:
  1) El usuario accede al sistema.
  2) El usuario presiona el botón “nuevo ticket”
  3) El usuario completa los datos de la generación de ticket
  4) El usuario presiona el botón “aceptar”

  Excepciones:
  1) ImposibleModificarTicketsException: El estado del sistema no permite agregar tickets en ese momento.

  Post-condiciones:
  1) Un nuevo ticket es creado y asignado al usuario.

- **Eliminar ticket**
  Descripción:
  Un usuario tipo empleador o empleado pretenso selecciona la opción de eliminar ticket
  Actores:
  Empleado pretenso o empleador
  Pre-condiciones:
  El usuario debe estar logueado en el sistema, el ticket debe existir.
  Flujo normal:
  1. El usuario accede al sistema.
  2. El usuario presiona el botón “eliminar ticket”

  Excepciones:
  1. ImposibleModificarTicketsException: El estado del sistema no permite agregar tickets en ese momento.

  Post-condiciones:
  El ticket existente es eliminado.

- **Seleccionar candidato**
  Descripción:
  Un usuario tipo empleador o empleado pretenso selecciona un candidato.
  Actores:
  Empleado pretenso o empleador
  Pre-condiciones:

El usuario debe estar logueado en el sistema, otro usuario de otro tipo debe existir.
Flujo normal:

1) El usuario accede al sistema.
2) El usuario presiona el botón “eliminar ticket”

Post-condiciones:
Usuario es asociado con usuario de otro tipo.

- **Cerrar sesión**
  Descripción:
  Un usuario cierra la sesión en el sistema
  Actores:
  Un usuario: puede ser empleador, empleado o administrador.
  Pre-condiciones:
  Un usuario debe estar logueado en el sistema
  Flujo normal:
  1) Un usuario logueado en el sistema, presiona el botón “Cerrar sesión”
  Post-condiciones:
  No hay un usuario logueado en el sistema.

Reporte de errores:

- Clase Empleador:
  - En el constructor, no se almacena correctamente el atributo de telefono y realName, intercambiándolas.
- Ticket:
  - Método “GetComparacionJornada”, invocado por un ticket cuyo atributo en “jornada” es “extendida” y se pasa por parámetro otro ticket cuyo atributo es también “extendida”, Se esperaba una salida == 1, sin embargo se obtuvo -0.5
  - Método “GetComparacionPuesto”, invocado por un ticket cuyo atributo en “puesto” es “managment” y se pasa por parámetro otro ticket cuyo atributo es también “managment”, Se esperaba una salida == 1, sin embargo se obtuvo -0.5
  - Método “GetComparacionRemuneracion”, invocado por un ticket cuyo atributo en “remuneracion” es 800 y se pasa por parámetro otro ticket cuyo atributo es 1200. Siendo v1 == 1000 y v2 == 3000, se esperaba una salida de -0.5, sin embargo se obtuvo 1.
  - Método “GetComparacionRemuneracion”, invocado por un ticket cuyo atributo en “remuneracion” es 800 y se pasa por parámetro otro ticket cuyo atributo es 4000. Siendo v1 == 1000 y v2 == 3000, se esperaba una salida de -1, sin embargo se obtuvo 1.
  - Método “GetComparacionRemuneracion”, invocado por un ticket cuyo atributo en “remuneracion” es 1200y se pasa por parámetro otro ticket cuyo atributo es 4000. Siendo v1 == 1000 y v2 == 3000, se esperaba una salida de -0.5, sin embargo se obtuvo 1.
  - Método “GetComparacionTotal”, invocado por un ticket igual al pasado por parámetro, se espera que todos sus métodos devuelven 1 y su suma corresponda a 6, sin embargo se obtiene 5.
- Agencia:

- Método “calculaPremiosCastigosAsignaciones”, se esperaba que un empleado pretenso que se posiciona último disminuye su puntaje en 5 puntos. En cambio se aumentó su puntaje en 30.
- Método “calculaPremiosCastigosAsignaciones”, se esperaba que un empleador que se posiciona primero en la lista de empleadores fuera premiado aumentando su puntaje en 10 puntos por cada lista.
- Método “gatillarRonda”, se esperaba que los empleadores que no realizaron ninguna contratación, disminuyan su puntaje en 20 puntos. En cambio se aumentó en 20 puntos.
- Método “registroEmpleador”, se esperaba que se lance una excepción de tipo NewRegisterException ya que se intentó registrar un empleador con un nombre de usuario ya existente. La excepción no se lanzó.

Test de Gui

Para realizar los test de GUI en lugar de utilizar el enfoque de acceso directo donde al acceder a los componentes se modifican sus atributos utilizando sus eventos, se decidió utilizar la clase Robot para poder simular los caminos, modificando sus componentes. La clase Robot interactúa con la interfaz gráfica como lo haría un usuario real, por lo que las pruebas pueden seguir funcionando si hay modificaciones en las interfaz, en nuestro caso son los cambios de paneles.

Utilizamos la clase TestUtil brindada por la cátedra para una mayor facilidad a la hora de implementar el robot.

Test de GUI para el panel LogIn

Test Enabled Disabled

En este panel, el botón “Login” puede estar enabled o disabled, mientras que el botón “Registrarse” permanece enabled todo el tiempo. Por esta razón, se testeo solo el comportamiento del primero mencionado.

El comportamiento esperado del botón “LogIn” estará habilitado si solo si los campos “Nombre de Usuario” y “Contraseña” no están vacíos. Se realizaron diferentes test para para abarcar las diferentes posibles combinaciones de orden en que estos campos fueron llenados. Dado que este número es chico, se testearon todos los posibles casos.

Escenarios posibles con su resultado esperado:

1. “Nombre de usuario” completo y “Contraseña” vacío → botón logIn deshabilitado
2. “Nombre de usuario” vacío y “Contraseña” completo → botón logIn deshabilitado
3. “Nombre de usuario” completo y “Contraseña” completo → botón logIn habilitado

Los resultados obtenidos fueron los esperados para todos los escenarios, por lo que se concluye que no hay errores en la habilitación y deshabilitación de botones en este panel.

Test de cambio de paneles

Desde el panel LogIn se puede acceder al panel Registro, al panel Cliente y al panel Admin. Para el primer caso, solo se requiere presionar el botón “Registrarse”. Para el segundo caso, es necesario que el usuario a loguear ya haya sido registrado en la agencia. Y por último, para el tercer caso, el “Nombre de usuario” y “Contraseña” deben ser “admin”.

Escenarios posibles con su resultado esperado:

1. Login con un empleado pretenso → cambia al panel Cliente de ese empleado

2. Login con un empleador → cambia al panel Cliente de ese empleador

3. Login de un admin → cambia al panel Admin

4. Registrarse → cambia al panel Registro

Para verificar que el cambio de panel fue realizado con éxito, se utilizó una referencia a un botón del panel esperado y se verificó, mediante un assert, que este esté habilitado.

Los resultados obtenidos fueron los esperados para todos los escenarios, por lo que se concluye que no hay errores en el cambio de paneles desde este panel.

Test conjunto con datos

Estos test hacen referencia a lo que debe suceder cuando un usuario al loguearse escribe mal su usuario o su contraseña, el estado en el que debe estar el panel luego de cerrar sesión desde el panel Cliente o Admin, y si registra correctamente a los usuarios.

Escenarios posibles con su resultado esperado:

1. Se espera que si un usuario escribe mal su nombre de usuario, salte un cartel con el siguiente mensaje “Usuario desconocido”.

2. Se espera que si el usuario escribe mal su contraseña, salte un cartel con el siguiente mensaje “Contraseña Incorrecta”.

3. Se espera que si un cliente de tipo Empleado Pretenso cierra sesión, al volver al panel login, este se encuentre sin datos escritos en los campos.

4. Se espera que si un cliente de tipo Empleador cierra sesión, al volver al panel login, este se encuentre sin datos escritos en los campos.

5. Se espera que si se registraron x cantidad de empleados e y cantidad de empleadores, estos se hayan guardado bien en la agencia.

Los test realizados para los escenarios 1, 2 y 5 dieron los resultados esperados. Sin embargo, los resultados de los test 3 y 4 dieron resultados distintos a los esperados, es decir, que cuando un cliente de tipo Empleado Pretenso o Empleador cierra sesión, al volver al panel login, este se encuentra con los campos “Nombre de usuario” y “Contraseña” completos con los que el usuario se logueo.

Test de GUI para el panel Registrar

Test Enabled Disabled

En este panel, según la persona que se quiera registrar, puede ser un Empleado o un empleador, el botón “Registrar” puede estar enabled o disabled según sean completados algunos o todos los campos de datos que le correspondan a cada uno.

El comportamiento esperado para el botón “Registrar” es que se encuentre deshabilitado mientras no estén completos todos los campos y esté habilitado cuando estén todos los campos completados

Para un empleador se decidió hacer un corte en los casos a testear.
Se testea solamente los casos donde son cargados todos los campos menos uno o cuando se carga solo un campo y el resto quede vacío, ya que si se tuvieran que testear todos los casos considerando cada combinación lleno/vacío tendríamos $2^7 = 128$ casos sin considerar el orden en el que los campos fueron llenados.

Escenarios posibles con su resultado esperado:

Para un Empleador:

Todos los campos completos

1. “Nombre de usuario”, “Contraseña”, “Repetir contraseña”, “Nombre Real”, “Telefono” completos → botón Registrar habilitado.

- Sin completar un campo

2. “Nombre de usuario” vacío y “Contraseña”, “Repetir contraseña”, “Nombre Real”, “Telefono” completos → botón Registrar deshabilitado.

3. “Contraseña” vacío y “Nombre de usuario” , “Repetir contraseña”, “Nombre Real”, “Telefono” completos → botón Registrar deshabilitado.

4. “Repetir contraseña” vacío y “Nombre de usuario”, “Contraseña”, “Nombre Real”, “Telefono” completos → botón Registrar deshabilitado.

5. “Nombre Real” vacío y “Nombre de usuario”, “Contraseña”, “Repetir contraseña”, “Teléfono” completos → botón Registrar deshabilitado.

6. “Telefono” vacío y “Nombre de usuario”, “Contraseña”, “Repetir contraseña”, “Nombre Real” completos → botón Registrar deshabilitado.

- Solo con un campo completo

7. “Nombre de usuario” completo y “Contraseña”, “Repetir contraseña”, “Nombre Real”, “Teléfono” vacíos → botón Registrar deshabilitado.

8. “Contraseña” completo y “Nombre de usuario”, “Repetir contraseña”, “Nombre Real”, “Teléfono” vacíos → botón Registrar deshabilitado.

9. “Repetir contraseña” completo y “Nombre de usuario”, “Contraseña”, “Nombre Real”, “Teléfono” vacíos → botón Registrar deshabilitado.

10. “Nombre Real” completo y “Nombre de usuario”, “Contraseña”, “Repetir contraseña”, “Teléfono” vacíos→ botón Registrar deshabilitado.

11. “Telefono” completo y “Nombre de usuario”, “Contraseña”, “Repetir contraseña”, “Nombre Real” vacíos → botón Registrar deshabilitado.

Para un Empleado:

- Todos los campos completos

1. “Nombre de usuario”, “Contraseña”, “Repetir contraseña”, “Nombre Real”, “Telefono”, “Apellido”, “Edad” completos → botón Registrar habilitado.

- Sin completar un campo

2. “Apellido” vacío y “Nombre de usuario”, “Contraseña”, “Repetir contraseña”, “Nombre Real”, “Telefono” “Edad”, completos → botón Registrar deshabilitado.

3. “Edad” vacío y “Nombre de usuario”, “Contraseña”, “Repetir contraseña”, “Nombre Real”, “Telefono”, “Apellido” completos → botón Registrar deshabilitado.

- Solo un campo completo

4. “Apellido” completo y “Nombre de usuario”, “Contraseña”, “Repetir contraseña”, “Nombre Real”, “Telefono”, “Edad” vacíos → botón Registrar deshabilitado.

5. “Edad” completo y “Nombre de usuario”, “Contraseña”, “Repetir contraseña”, “Nombre Real”, “Telefono”, “Apellido” vacíos → botón Registrar deshabilitado.

Para un Empleado no se testean los casos de los campos que se encuentran en el lado izquierdo del panel Registro, ya que estos test se consideran en lso escenarios del 2 al 11 de la parte empleador.

Los resultados obtenidos son los esperados para todos los escenarios tanto para los del empleador como empleado, por lo que se concluye que no hay errores en la habilitación y deshabilitación de botones en este panel.

Test de cambio de paneles

Desde este panel con el botón cancelar se vuelve al panel de Login y con el botón de Registrar después de cargar los campos correspondientes a empleado o empleador se puede cambiar de panel al panel de Empleado o empleador según corresponda.

El comportamiento esperado estando en el panel de Registro es que se vuelve al panel Login si se presiona el botón cancelar. Si se carga el formulario y se presiona el botón

Registrar se cambia al panel que corresponda según se este registrando un empleado o empleador.

Escenarios posibles con su resultado esperado:

1. Presionar botón cancelar → cambia a panel login

2. Presionar botón registrar → Cambia al panel empleado o empleador según corresponda

Los resultados obtenidos son los esperados para todos los escenarios, por lo que se puede concluir que no hay errores en el cambio de panel desde este panel.

Test conjunto con datos

En estos test se probara que sucede si se carga un usuario repetido o si la contraseña no coincide con la confirmación de contraseña si aparecen los mensajes de advertencia correspondientes.

El comportamiento esperado cuando se registra una persona ya sea empleado o empleador al momento de ingresar la contraseña y su contraseña es que aparezca un mensaje advirtiendo que no coinciden las contraseñas. También debería aparecer un mensaje de advertencia cuando una persona intenta volver a registrarse, este mensaje debería decir que el usuario se encuentra repetido.

Escenarios posibles con su resultado esperado:

1. Se registra un usuario ya registrado → mensaje: usuario repetido

2. La contraseña y su confirmación no son iguales → mensaje: Contraseña no coinciden

Los resultados obtenidos para el escenario 1 fueron los esperados, en cambio para el escenario 2 no se obtuvieron los valores esperados, ya que no se advierte de que las contraseña y su confirmación son diferentes y se procede a realizar el registro. Se concluye que no hay errores en la notificación cuando se quiere registrar un usuario ya registrado pero si cuando se ingresa una confirmación de contraseña diferente.

Test GUI panel Cliente

Test Enabled Disabled

Este es el panel donde el cliente, ya sea Empleado pretenso o Empleador, puede realizar las acciones por las cuales paga el servicio. Aquí es donde se crean tickets, donde se eliminan tickets y donde el usuario puede visualizar la lista de candidatos y realizar los match.

Se realizaron diferentes test para para abarcar las diferentes posibles combinaciones para cada uno de los botones en el panel. Dado que este número es chico, se testearon todos los posibles casos.

Escenarios posibles y resultado esperado:

- Botón “Aceptar”
    1. Campo de “Remuneración” con un número negativo → Botón “Aceptar” deshabilitado

    2. Campo de “Remuneración” vacío → Botón “Aceptar” deshabilitado

    3. Campo de “Remuneración” con un número entero positivo → Botón “Aceptar” habilitado

- Botón “Nuevo Ticket”

    4. Luego de presionar el botón “Nuevo ticket” → Botón “Nuevo ticket” deshabilitado

    5. Luego de presionar el botón “Nuevo ticket”, seguido de completar el campo “Remuneración” con un entero positivo” y finalmente presionar el botón “Aceptar” → Botón “Nuevo ticket” habilitado

- RadioButtons

    6. En el estado inicial del panel → RadioButtons deshabilitados

    7. Luego de presionar “Nuevo ticket” → RadioButtons habilitados

- Botón “Eliminar Ticket”

    8. Si no hay un ticket creado → Botón “Eliminar ticket” deshabilitado

    9. Si hay ticket creado → Botón “Eliminar ticket” habilitado

    10. Si no hay ticket creado → “Text Area Ticket” debería decir “Sin tickets creados”

- Botón Seleccionar Candidato

    11. En cualquier momento estando en el panel Cliente → Botón “Seleccionar candidato” debe estar habilitado

Los resultados obtenidos para todos los test fueron los esperados. Por lo que se concluye que no hay errores en este panel.

Test GUI panel Admin

Test Enabled Disabled

En este panel, el único botón que puede estar habilitado o deshabilitado es el botón “Cambiar”, que se encarga de modificar los valores de los límites inferior y superior. Este estará habilitado si solo si ambos campos son número enteros positivos y el límite superior es mayor que el límite inferior. En el caso que se ingrese un límite negativo, debería saltar un cartel con el mensaje “Límite de remuneración negativo” y en el caso que se ingresen número invalido, debería saltar un cartel con el mensaje “Límite de remuneración inválido”.

El botón “Gatillar” debe estar siempre habilitado. Luego de presionarlo debería salir el mensaje “Agencia en estado de contratación, se generaron listas de postulantes” y, luego de presionarlo por segunda vez, debería salir el mensaje “Agencia en estado de búsqueda laboral, se pueden crear y eliminar tickets ”.

Se realizaron diferentes test para abarcar las diferentes posibles combinaciones para que cada botón se habilite o se deshabilite y para verificar la salida correcta de los mensajes. Dado que este número es chico, se testearon todos los posibles casos.

Escenarios posibles y resultado esperado:

- Botón “Cambiar”

1. límite inferior positivo, límite superior vacío → Botón “Cambiar” deshabilitado
2. límite inferior vacío, límite superior positivo → Botón “Cambiar” deshabilitado
3. límite inferior negativo, límite superior positivo → Botón “Cambiar” deshabilitado
4. límite inferior positivo, límite superior negativo → Botón “Cambiar” deshabilitado
5. límite inferior negativo, límite superior negativo → Botón “Cambiar” deshabilitado
6. límite superior positivo menor que límite inferior positivo → Botón “Cambiar” deshabilitado
7. límite superior positivo mayor que límite inferior positivo → Botón “Cambiar” habilitado

- Mensaje “Límite de remuneración inválido” y Mensaje “Límite de remuneración negativo”.

8. límite inferior positivo, límite superior vacío → Mensaje “Límite de remuneración inválido”.

9. límite inferior vacío, límite superior positivo → Mensaje “Límite de remuneración inválido”.

10. límite inferior negativo, límite superior positivo →Mensaje “Límite de remuneración negativo”.

11. límite inferior positivo, límite superior negativo → Mensaje “Límite de remuneración negativo”.

12. límite inferior negativo, límite superior negativo → Mensaje “Límite de remuneración negativo”.

13. Límite superior positivo menor que límite inferior positivo → Mensaje “Límite de remuneración inválido”.

- Botón “Gatillar ronda”

14. Luego de presionar “Gatillar ronda” → Mensaje “Agencia en estado de contratación, se generaron listas de postulantes”

15. Luego de presionar “Gatillar ronda” dos veces “Agencia en estado de búsqueda laboral, se pueden crear y eliminar tickets ”.

Los resultados obtenidos de los test 1-7 y 14-15 fueron los esperados. Mientras que los resultados de los test 8-13 no fueron los esperados. Cuando se ingresa un valor no válido en los límites de remuneración no aparece ninguno de los dos carteles que tendrían que aparecer.

Test Cambio Panel

Aquí se testeo el cambio de panel al panel LogIn. Esto se realiza mediante el método cerrar sesión del controlador. Dado que es el mismo método que se usa para cerrar sesión desde el panel cliente, solo se prueba una vez y se optó por probarlo desde este panel por una cuestión de comodidad.

Escenarios posibles y resultado esperado:

1. Presionar boton “Cerrar sesion” → Vuelve al panel Login

Los resultados obtenidos fueron los esperados para todos los escenarios, por lo que se concluye que no hay errores en el cambio de paneles desde este panel

Test de persistencia

Test Persistencia Con datos

Aquí se testeo el método cargarAgencia() de la clase agencia, la cual se encarga de persistir toda la información de una agencia en un archivo.

Para el testeo, se creó un cliente de tipo Empleado pretenso y un ticket. Esto se guardó mediante el método previamente mencionado en el archivo y luego se verificó que lo guardado no sea NULL.

Luego de realizar el test, se concluye que el método realiza lo esperado y no tiene errores.

Test de Persistencia Conjunto vacío

Aquí se testeo que se cree un archivo vacío y que, si no se crearon clientes, la persistencia sea vacía. Se utilizó el método guardarAgencia() para verificar que el archivo fue creado y el método cargarAgencia() para verificar que el archivo persistido este vacío.

Luego de realizar el test, se concluye que el método realiza lo esperado y no tiene errores.

Test Persistencia Agencia DTO

La persistencia se realiza en codificación XML mediante el patrón DTO. Esto es crear un objeto Agencia de tipo DTO para poder persistir y despersistir sus datos.

Se testearon los métodos AgenciaFromAgenciaDTO y AgenciaDTOFromAgencia. El primer método mencionado funciona sin generar errores, mientras que el segundo, no transfiere correctamente los datos del límite inferior puesto por la agencia.

Test Persistencia XML

Este test prueba que se pueda abrir y cerrar el archivo persistido Agencia.xml, también se prueba que se pueda leer y que se generen los archivos persistidos con datos cargados y sin datos (vacío).

Los métodos testeados: testAbrirInput, testAbrirOutput, TestCerrarInput, TestCerrarOutput, testLeer, testEscribir1 , testEscribir2. Estos 2 últimos test generan archivos persistidos con los datos que se utilizan para el test.

Todas las pruebas dan los resultados esperados, por lo que se puede concluir que no hay errores en la persistencia.

Recorte

A lo largo del proceso del testing decidimos realizar recortes para minimizar las pruebas realizadas en los casos en los que eran demasiadas o muy extensas**:**

- En la clase EmpleadoPretenso, solo se testeó los atributos únicos (apellido y edad) ya que los demas que son heredados de la clase Cliente, fueron testeados en el constructor de Empleador.
- En los métodos de comparación de tickets, se testearon únicamente los tickets con los atributos a comparar cambiados. Es decir, por ejemplo, si queremos comparar la locación de dos tickets, solo se testearon los casos en los que cambia la locación, no los otros parámetros.
- En el método de cerrar cesión, se testeó únicamente cuando un usuario está logueado, no con los 3 tipos de usuario diferentes.
- En el test de GUI, se realizaron cortes en en panel en común que tienen registrar empleador y empleado. Se realizó el test en una clase, mientras que se dió por hecho en otra.
- En el test de GUI, se realizaron cortes al registrar un nuevo cliente para el llenado de textFields, solo se probó que se bloquee el registro en caso que un solo campo esté vacío o incompleto.

Conclusiones

En conclusión, y como cierre de este trabajo práctico, la ejecución de las pruebas reveló varios problemas en el código que no se habían detectado previamente. Este proceso no solo demostró la importancia de las pruebas, sino que también facilitó la identificación de estos errores. Además, el uso de la clase Robot facilitó la validación del comportamiento de la interfaz de usuario en diferentes escenarios. Esto permitió asegurar que los elementos de la interfaz respondieran correctamente a las acciones del usuario, como clics y entradas de teclado.

Además, destacamos una vez más lo fructífero, en términos de aprendizaje, que es esta metodología de proyecto en relación a un examen donde haya que desarrollar un código en tiempo límite. Si bien el proceso del testing puede llegar a ser tedioso, resulta un paso fundamental a la hora del proceso de desarrollo de software.

Logramos entender la importancia del testing y lo vital que resulta para aplicar en las situaciones reales y cotidianas de la vida laboral, ya que al principio de la materia ninguno de nosotros entendíamos que el proceso de testeo era tan vital como el proceso de codear un programa.

Aprendimos a trabajar con contratos, a comprenderlos y a aplicarlos a la realidad, ya sea tanto para desarrollar un código como para testear el mismo. También entendimos la importancia de que éste sea bien documentado y no deje lugar a ambigüedades, ya que en los momentos que sucedió, sabíamos que esos requerimientos podrían ser problemas que deberíamos volver a tratar a futuro.

También comprendimos la necesidad de ser cuidadosos con nuestros códigos, a comentarlos en lugares donde podría haber dudas y a documentarlos de manera correcta, ya que la persona que reciba ese código ya sea para continuar nuestra labor como programadores, como para testearlo, puede tener sus dudas acerca de lo que hicimos, y la

forma correcta de evacuarlas es viendo una documentación de los mismos que no deje lugar a ambigüedades.