

Programación Orientada a Objetos

Proyecto Peaje

Docente

Ing. Yasmín Moya Villa, MSc.

Integrantes:

Jesús De Los Reyes Madrid

Jorge Marín Góngora



Ingeniería de Sistemas

2017

Enunciado Proyecto Peaje.

En los límites de una ciudad, se ha construido un peaje el cual está a punto de iniciar servicios. Los administradores del mismo, quieren llevar registro no sólo de los vehículos que transitan, sino también reunir datos de los conductores. Para ello, se requieren sus servicios para desarrollar una aplicación Orientada a Objetos en Java, que cumpla con las siguientes tareas.

Primero, calcular el precio que debe pagar el vehículo que está a punto de pasar el peaje. El costo es de acuerdo al tipo de vehículo, un vehículo se puede considerar como automóvil o un camión transportador de carga.

Independientemente del tipo, se guardarán los siguientes datos por defecto de un vehículo el cual tendrá asociado una placa, marca, modelo y la fecha de la última revisión técnico mecánica realizada.

Sí el vehículo se considera un camión de carga, se requerirán los siguientes datos extra:

- Número de ejes.
- Cantidad máxima de carga en kilos que puede transportar. ☐ Si el vehículo cuenta con seguro vigente.

Sí el vehículo es un automóvil, no se requiere información extra a rellenar.

Del conductor se desea conocer su nombre, numero de identidad, teléfono y ciudad de residencia.

Se emitirá una factura que tendrá un registro de la fecha y hora en la que el vehículo pasa el peaje, placa y tipo del vehículo, número de identidad del conductor, sí se realizó un descuento y el sentido del destino a recorrer. También contará con un identificador único que se asignará automáticamente al momento de la creación de la misma.

Sí se trata de un camión, se agregarán los siguientes campos extra a la factura. Peso de la carga en Kilos, breve descripción de la misma y detalle de las actividades a realizar en el destino.

El precio total, independientemente del tipo de vehículo, recibirá 50% de descuento sí el conductor antes había pasado por el peaje pero en un sentido contrario en el mismo día. De forma que sí al inicio salió y luego regresó a la ciudad, el costo disminuirá. Para ello debe indicarse sí se realizó o no el descuento en el registro en la facturación y aplicarlo al precio total que calculará el mismo ente, para ello, y solo para el cálculo del costo total del peaje, necesitará los datos del vehículo evaluado.

El precio para un automóvil se calcula:

- Precio base de 5.700 pesos.
- Sí sale de la ciudad se le adicionan 3.500 pesos, si entra, 5.000 pesos.

Por otro lado el precio para un camión se calcula:

- 12.500 pesos de precio base.
- Sí sale de la ciudad se le adicionan 5.000 pesos, si entra, 6.500.
- Más 5.000 pesos multiplicados por el número de ejes del vehículo. Se le adicionan 2.500 pesos multiplicados por número de kilos con los que carga.
- En caso de que el peso cargado supere la capacidad máxima, se cobrarán como sanción 7.000 pesos por kilo de exceso en adición a lo anterior.

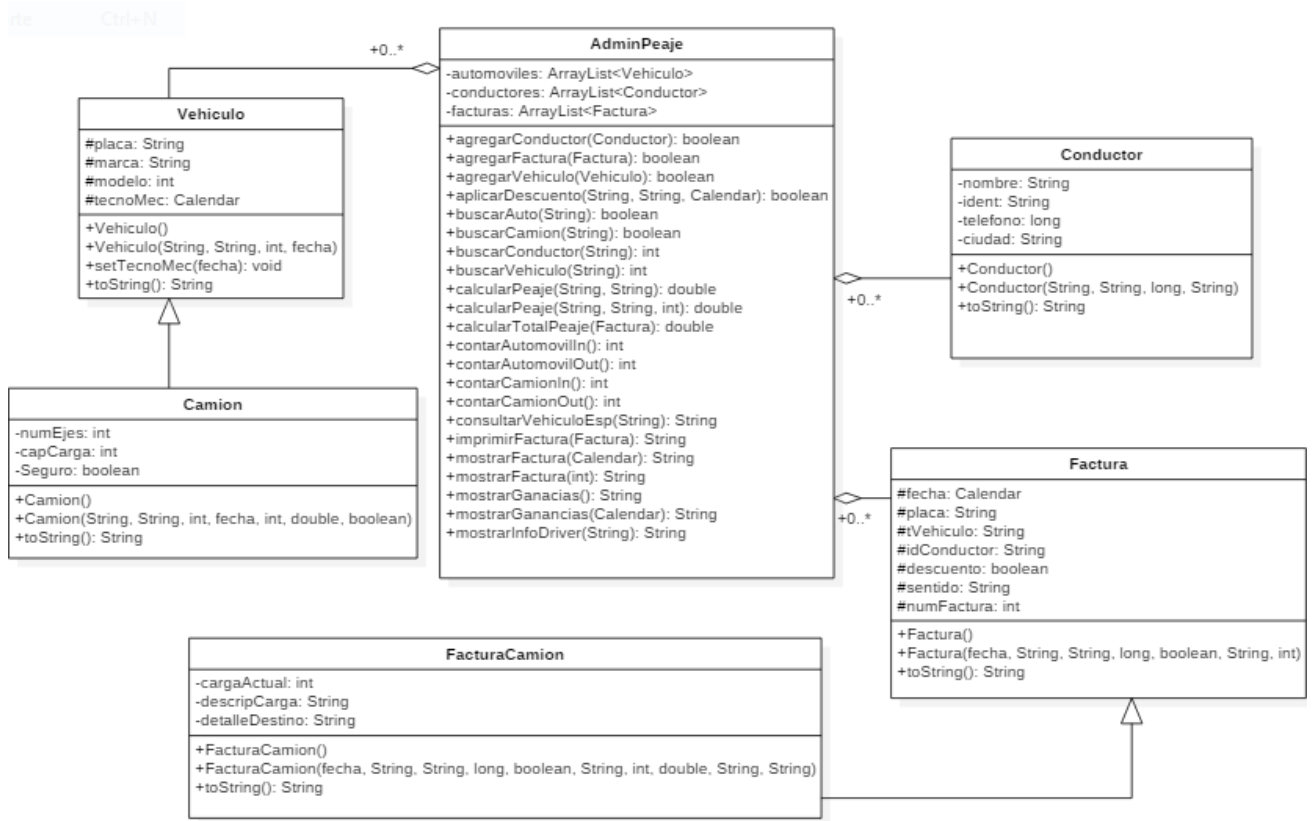
Tener en cuenta que cada vehículo cuenta con información que permite calcular el precio parcial a pagar del peaje y servirá de apoyo para el cálculo total que efectúan las facturas. No olvidar que se debe validar que el identificador del vehículo asignado en la factura corresponde con el vehículo que provee la información complementaria.

La administración central llevará un registro de todas las actividades cumpliendo los siguientes requisitos funcionales. Emitir una nueva factura, que se encarga de crear y almacenar una nueva factura a partir de los datos requeridos en la cual, tanto el vehículo como el conductor deben estar registrados en el sistema. En consecuencia, registrar un nuevo conductor y un nuevo vehículo serán operaciones a realizar por el programa.

También se debe proveer información acerca de los registros almacenados de acuerdo a diferentes filtros, como la cantidad de automóviles o camiones que han transitado el peaje en conjunto con el sentido del destino. Cantidad total de vehículos que han salido o entrado a la ciudad. Entregar un reporte indicando los datos de las facturas registradas en un día específico en conjunto con el costo total del peaje. Consultar los datos de un conductor, vehículo o factura específica (incluyendo el costo total del peaje), de acuerdo a los identificadores únicos de cada uno. Por último, proveer la cantidad de dinero recolectado, ya sea general, o en un día específico.

Recordar que el peaje es el único ente capaz de emitir facturas, por lo cual, solamente recibirá los datos necesarios para la creación de una factura correspondiente y se debe generar un identificador único para cada una, que cumplirá algún criterio designado por el desarrollador mismo.

Diagrama de Clases.

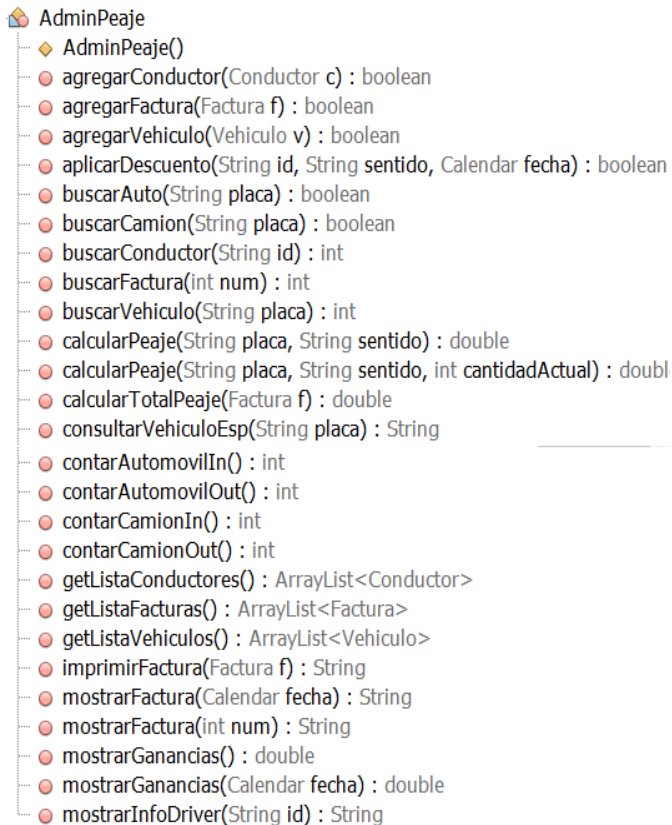


Programación.

Lógica.

Clases:

AdminPeaje:



```
AdminPeaje
AdminPeaje()
agregarConductor(Conductor c) : boolean
agregarFactura(Factura f) : boolean
agregarVehiculo(Vehiculo v) : boolean
aplicarDescuento(String id, String sentido, Calendar fecha) : boolean
buscarAuto(String placa) : boolean
buscarCamion(String placa) : boolean
buscarConductor(String id) : int
buscarFactura(int num) : int
buscarVehiculo(String placa) : int
calcularPeaje(String placa, String sentido) : double
calcularPeaje(String placa, String sentido, int cantidadActual) : double
calcularTotalPeaje(Factura f) : double
consultarVehiculoEsp(String placa) : String
contarAutomovilIn() : int
contarAutomovilOut() : int
contarCamionIn() : int
contarCamionOut() : int
getListaConductores() : ArrayList<Conductor>
getListaFacturas() : ArrayList<Factura>
getListaVehiculos() : ArrayList<Vehiculo>
imprimirFactura(Factura f) : String
mostrarFactura(Calendar fecha) : String
mostrarFactura(int num) : String
mostrarGanancias() : double
mostrarGanancias(Calendar fecha) : double
mostrarInfoDriver(String id) : String
```

Calendar: Es una clase abstracta, por lo que no podemos hacer un new de ella. La forma de obtener una instancia es llamando al método *getInstance()*, que nos devolverá alguna clase hija de *Calendar* inicializada con la fecha/hora actual.

```
Calendar today = Calendar.getInstance();
```

GregorianCalendar: Es una clase hija que devuelve el calendario estándar para el mundo occidental.

```
today is java.util.GregorianCalendar[time=1394883514531,areFieldsSet=true,areAllFieldsSet=true,
lenient=true,zone=sun.util.calendar.ZoneInfo[id="Europe/Paris",
offset=3600000,dstSavings=3600000,useDaylight=true,transitions=184,
lastRule=java.util.SimpleTimeZone[id=Europe/Paris,offset=3600000,dstSavings=3600000,
useDaylight=true,startYear=0,startMode=2,
startMonth=2,startDay=-1,startDayOfWeek=1,startTime=3600000,startTimeMode=2,endMode=2,
endMonth=9,endDay=-1,endDayOfWeek=1,endTime=3600000,endTimeMode=2]],firstDayOfWeek=2,
minimalDaysInFirstWeek=4,ERA=1,YEAR=2014,MONTH=2,WEEK_OF_YEAR=11,WEEK_OF_MONTH=2,
DAY_OF_MONTH=15,DAY_OF_YEAR=74,DAY_OF_WEEK=7,DAY_OF_WEEK_IN_MONTH=3,AM_PM=1,HOUR=0,
HOUR_OF_DAY=12,MINUTE=38,SECOND=34,MILLISECOND=531,ZONE_OFFSET=3600000,DST_OFFSET=0]
```

Métodos:

getTime: devuelve el *Calendar* convertido a *Date*.

```
today is Sat Mar 15 12:41:10 CET 2014
```

Para obtener un *Calendar* en una fecha hora concreta tenemos dos opciones. Una de ellas, sabiendo que lo que realmente vamos a obtener es un *GregorianCalendar*, es hacer un *new GregorianCalendar (...)* pasando como parámetros el año, mes, día, hora, minuto, segundo. No es necesario pasar todo, ya que *GregorianCalendar* tiene varios constructores con menos parámetros.

```
Calendar sameDate = new GregorianCalendar(2010, Calendar.FEBRUARY, 22, 23, 11, 44);
```

La otra opción para obtener un *Calendar* en una fecha/hora concreta es llamar a su método **set()** para fijar los campos que queramos cambiar. El método **set()** admite dos parámetros, uno para identificar el campo concreto a cambiar (año, mes, día, hora, minuto, segundo, milésimas de segundo) y un segundo parámetro que sería el valor a poner. El siguiente código muestra bastantes de las posibilidades.

```
Calendar sameDate = Calendar.getInstance();

sameDate.set(Calendar.YEAR, 2010);
// Month. 0 is January, 11 is November
sameDate.set(Calendar.MONTH, Calendar.AUGUST);
sameDate.set(Calendar.DAY_OF_MONTH, 23);

// Either 12-hour clock plus AM/PM
sameDate.set(Calendar.HOUR, 10);
sameDate.set(Calendar.AM_PM, Calendar.PM);
// or 24-hour clock
sameDate.set(Calendar.HOUR_OF_DAY, 22);

sameDate.set(Calendar.MINUTE, 36);
sameDate.set(Calendar.SECOND, 22);
sameDate.set(Calendar.MILLISECOND, 123);
```

agregarConductor: Este método es de tipo de dato boolean, recibe un objeto de tipo Conductor, en el cuál se encuentran los datos suministrados. Antes de intentar agregar, este se apoya en otro método llamado **buscarConductor**. En el método **agregarConductor** se verifica que si lo obtenido por **buscarConductor** es igual a -1, se agregar el conductor a **listaConductores** y retorna **true**, sino es igual -1, el método retorna false.

```

public boolean agregarConductor(Conductor c) {
    if (buscarConductor(c.getIdent()) == -1) {
        listaConductores.add(c);
        return true;
    }
    return false;
}

```

agregarFactura: Este método es de tipo de dato boolean. Recibe un objeto de tipo Factura, en el cuál se encuentran los datos suministrados. Antes de intentar agregar, este se apoya en otro método llamado **buscarFactura**. En el método **agregarFactura** se verifica que lo obtenido en **buscarFactura** sea igual a -1, si se da ese caso, se agrega la factura en el ArrayList **listaFacturas** y como resultado se retorna un **true**, si lo obtenido por **buscarFactura** es diferente de -1, entonces **agregarFactura** retorna un false, debido a que la factura que intenta crear ya existe.

```

public boolean agregarFactura(Factura f) {
    if (buscarFactura(f.numFactura) == -1) {
        listaFacturas.add(f);
        return true;
    }
    return false;
}

```

agregarVehiculo: Este método es de tipo de dato boolean. Recibe un objeto de tipo Vehículo, en el cuál se encuentran los datos suministrados. Antes de intentar agregar, este se apoya en otro método llamado **buscarVehiculo**. En el método **agregarVehiculo** se verifica que lo obtenido en **buscarVehiculo** sea igual a -1, si se da ese caso, se agrega la factura en el ArrayList **listaVehiculos** y como resultado se retorna un **true**, si lo obtenido por **buscarVehiculo** es diferente de -1, entonces **agregarVehiculo** retorna un false, debido a que la factura que intenta crear ya existe.

```

public boolean agregarVehiculo(Vehiculo v) {
    if (buscarVehiculo(v.placa) == -1) {
        listaVehiculos.add(v);
        return true;
    }
    return false;
}

```

aplicarDescuento: Este método es de tipo de dato boolean. Recibe 3 parámetros: La identificación del conductor (tipo de dato String), el sentido hacia el cual va el vehículo (tipo de dato String) y la fecha en que este pasa por el vehículo (tipo de dato Calendar). Dentro de este se encuentra un for each el cual se encarga de utilizar un iterador de tipo factura que compara en cada posición del ArrayList **listaFacturas** con los parámetros ingresados. En las condiciones se verifica que los parámetros Identificación y Fecha sean iguales y que Sentido sea diferente a los obtenidos por el iterador, si ese caso se da, devuelve un **true**, eso indica que existe descuento. En dado caso que al finalizar el ciclo no se logre efectuar la condición anterior se

devuelve un **false**, no hubo descuento.

```
public boolean aplicarDescuento(String id, String sentido, Calendar fecha){  
    for(Factura f:listaFacturas){  
        if(!f.isDescuento())  
            if(id.equalsIgnoreCase(f.idConductor))  
                if(fecha.getTime().getDate()==f.fecha.getTime().getDate())  
                    if(fecha.getTime().getMonth()==f.fecha.getTime().getMonth())  
                        if(fecha.getTime().getYear()==f.fecha.getTime().getYear())  
                            if(!sentido.equalsIgnoreCase(f.sentido))  
                                return true;  
            }  
    }  
    return false;  
}
```

buscarAuto: Este método es de tipo de dato boolean. Recibe como parámetro la placa del automóvil (tipo de dato String). Dentro de este, se encuentra un for each el cuál se encarga de utilizar un iterador de tipo vehículo que se encarga de buscar en cada posición del ArrayList **listaVehiculos**. En el ciclo se encuentra un condicional el cual indica que si la placa obtenida como parámetro y la obtenida por el iterado son iguales y aparte el objeto obtenido no es instancia de camión, devuelve un **true**, indicando que el automóvil fue encontrado. Si al finalizar el ciclo se da el caso que esa condición no logró cumplir, se devuelve un **false** indicando que la búsqueda no fue efectiva.

```
public boolean buscarAuto(String placa){  
    for(Vehiculo v:listaVehiculos){  
        if(!(v instanceof Camion) && placa.equalsIgnoreCase(v.getPlaca()))  
            return true;  
    }  
    return false;  
}
```

buscarConductor: Este método es de tipo de dato **int**. Recibe como parámetro la identificación del conductor (tipo de dato String). En este se encuentra un for each el cuál se encarga de utilizar un iterador de tipo de dato Conductor que se encarga de verificar en cada posición del ArrayList **listaConductores**. En el ciclo hay un condicional que verifica que el parámetro obtenido sea igual a la identificación obtenida por el iterador, en caso que esa condición se cumpla, se retorna la posición obtenida en la cual se encuentra el conductor. Si al finalizar el ciclo la condición no se cumple, este retorna un -1, indicando que no se logró encontrar al conductor.

```
public int buscarConductor(String id){  
    for(Conductor c:listaConductores){  
        if(id.equalsIgnoreCase(c.getIdent()))  
            return listaConductores.indexOf(c);  
    }  
    return -1;  
}
```


buscarFactura: Este método es de tipo de dato int. Recibe como parámetro el número de la Factura (tipo de dato int). En este se encuentra un for each el cuál se encarga de utilizar un iterador de tipo de dato Factura que se encarga de verificar en cada posición del ArrayList **listaFacturas**. En el ciclo hay un condicional que verifica que el parámetro obtenido sea igual al número de la factura obtenida por el iterador, en caso que esa condición se cumpla, se retorna la posición obtenida en la cual se encuentra la Factura. Si al finalizar el ciclo la condición no se cumple, este retorna un -1, indicando que no se logró encontrar la Factura.

```
public int buscarFactura(int num){
    for(Factura f:listaFacturas){
        if(num==f.getNumFactura())
            return listaFacturas.indexOf(f);
    }
    return -1;
}
```

buscarCamion: Este método es de tipo de dato boolean. Recibe como parámetro la placa del camión (tipo de dato String). Dentro de este, se encuentra un for each el cuál se encarga de utilizar un iterador de tipo vehículo que se encarga de buscar en cada posición del ArrayList **listaVehiculos**. En el ciclo se encuentra un condicional el cual indica que si la placa obtenida como parámetro y la obtenida por el iterado son iguales y aparte el objeto obtenido es instancia de camión, devuelve un **true**, indicando que el camión fue encontrado. Si al finalizar el ciclo se da el caso que esa condición no logró cumplir, se devuelve un **false** indicando que la búsqueda no fue efectiva.

```
public boolean buscarCamion(String placa){
    for(Vehiculo v:listaVehiculos){
        if( v instanceof Camion && placa.equalsIgnoreCase(v.getPlaca()))
            return true;
    }

    return false;
}
```

buscarVehiculo: Este método es de tipo de dato int. Recibe como parámetro la placa de un vehículo (tipo de dato String). En este se encuentra un for each el cuál se encarga de utilizar un iterador de tipo de dato Vehículo que se encarga de verificar en cada posición del ArrayList **listaVehiculos**. En el ciclo hay un condicional que verifica que el parámetro obtenido sea igual a la placa obtenida por el iterador, en caso que esa condición se cumpla, se retorna la posición obtenida en la cual se encuentra el vehículo. Si al finalizar el ciclo la condición no se cumple, este retorna un -1, indicando que no se logró encontrar el vehículo.

```

public int buscarVehiculo(String placa){
    for(Vehiculo v:listaVehiculos){
        if(placa.equalsIgnoreCase(v.getPlaca()))
            return listaVehiculos.indexOf(v);
    }
    return -1;
}

```

calcularPeaje: Este método es de tipo de dato double. Recibe como parámetros la placa del vehículo (tipo de dato String) y el sentido hacia el cual va (tipo de dato String). Cuenta con una variable llamada valor, cantidad base que aumenta de acuerdo a ciertas condiciones. Dentro de este, hay un for each el cual se apoya en el método **buscarAuto**, si este devuelve **true**, se continua al condicional que valida el sentido y de acuerdo a esto, aumenta el valor y devuelve la cantidad a pagar. Si el método **buscarAuto**, devuelve un **false**, se procede a que **calcularPeaje**, retorne -1. Se utilizar para calcular los valores a pagar de los automóviles.

```

public double calcularPeaje(String placa, String sentido){
    double valor=5700;
    if(buscarAuto(placa)){
        if(sentido.equalsIgnoreCase("entrada"))
            return valor+5000;
        else
            return valor+3500;
    }else
        return -1;
}

```

calcularPeaje: Este método es de tipo de dato double. Recibe como parámetros placa (tipo de dato String), sentido hacia el cual va (tipo de dato String) y cantidad de carga actual (tipo de dato int). Cuenta con 5 variables: valor (tipo de dato double), pos (tipo de dato int) en la cual se guarda la posición obtenida por el método **buscarVehiculo**, capCarga (tipo de dato int), numEjes (tipo de dato int) y vehículo (tipo de dato Vehículo). Dentro de este, hay un condicional que valida que la posición obtenida sea diferente de menos y que si se encontró la placa, se procede a asignar a capCarga la capacidad de Carga del camión, para este se hace un casteo para que vehículo se comporte como camión. También se asigna numEjes a la cantidad de número de ejes del camión obtenido. En otro condicional, se valida que el sentido pasado como parámetro, depende lo obtenido, se aumenta la variable valor. En otro condicional, se valida si la diferencia entre la carga pasada como parámetro y la carga obtenida es positiva se añade cierto valor, si la diferencia es negativa, se hace el mismo procedimiento que el anterior pero se añaden otra cantidad de dinero como sanción al sobrepeso. Además, se añade un valor adicional por eje Si la placa no se encuentra retorna -1 y si la posición obtenida es -1, se retorna -1.

```

public double calcularPeaje(String placa, String sentido, int cantidadActual){
    double valor=12500;
    int pos=buscarVehiculo(placa);
    Vehiculo v;
    if(pos!=-1){
        if(buscarCamion(placa)){
            v=listaVehiculos.get(pos);
            double capCarga=((Camion)v).getCapCarga();
            double numEjes=((Camion)v).getNumEjes();

            if(sentido.equalsIgnoreCase("entrada"))
                valor+=6500;
            else
                valor+=5000;

            if(capCarga-cantidadActual<0){
                valor+=( (cantidadActual-capCarga)*7000);
            }

            valor+=cantidadActual*2500;
            valor+=numEjes*5000;

            valor+=cantidadActual*2500;
            valor+=numEjes*5000;

            return valor;
        }else
            return -1;
    }else
        return -1;
}

```

calcularPeajeTotal: Este método es de tipo de dato double. Recibe como parámetro una factura (tipo de dato Factura). Cuenta con una variable llamada valor, la cual cambia de acuerdo a los condicionales, si la factura pertenece a camión, tiene un incremento y sino pertenece tiene otro. Dentro de un condicional se utiliza el método **aplicarDescuento**, si este retorna **true**, se aplica el 50% de descuento al valor de la factura, si retorna **false**, no se aplica el descuento.

```

public double calcularTotalPeaje(Factura f){
    double valor;
    if(f instanceof FacturaCamion){
        valor=calcularPeaje(f.getPlaca(), f.getSentido(), ((FacturaCamion)f).getCargaActual());
    }
    else{
        valor=calcularPeaje(f.getPlaca(), f.getSentido());
    }

    if(aplicarDescuento(f.getIdConductor(), f.getSentido(), f.getFecha())){
        valor=valor*0.5;
    }
    return valor;
}

```

consultarVehiculoEspecifico: Este método es de tipo de dato String. Recibe como parámetro la placa del vehículo (tipo de dato String). Cuenta con dos variables: pos (tipo de dato int) la cual se le asigna la posición obtenida por el método **buscarVehiculo** y v (tipo de dato Vehículo). Dentro de este se encuentra un condicional el cual evalúa que lo obtenido en pos sea diferente a -1, si es así, continua y asigna el la posición obtenida en el ArrayList **listaVehiculos** a v. Luego valida que v sea instancia de la clase Camión y si es así, retorne su información, en caso de que no sea instancia, indica que ese vehículo es un automóvil y retorna su información. Si pos es igual a -1, retorna un **null**.

```

public String consultarVehiculoEsp(String placa){
    int pos=buscarVehiculo(placa);
    Vehiculo v;
    if(pos!=-1){
        v=listaVehiculos.get(pos);
        if(v instanceof Camion)
            return ((Camion)v).toString();
        else
            return v.toString();
    }

    return null;
}

```

contarAutomovilIn: Este método es de tipo de dato int. Cuenta con una variable llamada cont (tipo de dato int). En este se encuentra un for each, el cual cuenta con un iterador de tipo Factura que se encargara de buscar en cada posición del ArrayList **listaFacturas**. Dentro de este se encuentra un condicional el cual valida que sea un automóvil y que el sentido hacia el cual va sea de entrada, si eso se cumple, aumenta el contador. Al finalizar el ciclo, retorna el valor del contador.

```

public int contarAutomovilIn(){
    int cont=0;
    for(Factura f:listFacturas){
        if(f.tVehiculo.equalsIgnoreCase("automovil"))
            if(f.sentido.equalsIgnoreCase("entrada"))
                cont++;
    }
    return cont;
}

```

contarAutomovilOut: Este método es de tipo de dato int. Cuenta con una variable llamada cont (tipo de dato int). En este se encuentra un for each, el cual cuenta con un iterador de tipo Factura que se encargara de buscar en cada posición del ArrayList **listaFacturas**. Dentro de este se encuentra un condicional el cual valida que sea un automóvil y que el sentido hacia el cual va sea de salida, si eso se cumple, aumenta el contador. Al finalizar el ciclo, retorna el valor del contador.

```

public int contarAutomovilOut(){
    int cont=0;
    for(Factura f:listFacturas){
        if(f.tVehiculo.equalsIgnoreCase("automovil"))
            if(f.sentido.equalsIgnoreCase("salida"))
                cont++;
    }
    return cont;
}

```

contarCamionIn: Este método es de tipo de dato int. Cuenta con una variable llamada cont

(tipo de dato de Factura que este se era sea de del contar

```
public int
for(
    int
    for(
    }
    return
}
```

```
public String imprimirFactura(Factura f){
    StringBuilder sb = new StringBuilder();
    sb.append(f.toString());

    Vehiculo v = listaVehiculos.get(buscarVehiculo(f.placa));
    if(v instanceof Camion){
        int numEjes=((Camion) v).getNumEjes();
        int totalNumEjes=numEjes*5000;
        int cantActual= ((FacturaCamion)f).getCargaActual();
        int capCarga=((Camion) v).getCapCarga();
        int totalCant=cantActual*2500;
        int totalCarga=0;

        if(cantActual-capCarga>0){
            totalCarga=(cantActual-capCarga)*7000;
        }

        sb.append("\n");
        sb.append("\nPRECIO BASE: $12500");
        sb.append("\nDESTINO: ");
        sb.append((f.sentido.equalsIgnoreCase("entrada"))?"$6500":"$5000");
        sb.append("\nNÚMERO DE EJES: $5000 x ");
        sb.append(numEjes);
        sb.append(" = $");
        sb.append(totalNumEjes);
        sb.append("\nCANTIDAD : $2500 x ");
        sb.append(cantActual);
        sb.append(" = $");
        sb.append(totalCant);
        if(cantActual-capCarga>0){
            sb.append("\nKILOS DE EXCESO : $7000 x ");
            sb.append(cantActual-capCarga);
            sb.append(" = $");
            sb.append(totalCarga);
        }
    }else{
        sb.append("\n");
        sb.append("\nPRECIO BASE: $5700");
        sb.append("\nDESTINO: ");
        sb.append((f.sentido.equalsIgnoreCase("entrada"))?"$5000":"$3500");
    }
    double valor;
    valor=calcularTotalPeaje(f);
    sb.append("\nVALOR DEL PEAJE: $ ");
    sb.append(valor);
    return sb.toString();
}
```

or de tipo
i. Dentro de
lo hacia el cual
torna el valor

contarC
cont (tipo
Factura q
este se er
va sea de
contador.

able llamada
erador de tipo
i. Dentro de
lo hacia el cual
rna el valor del

```
public int contarCamionOut(){
    int cont=0;
    for(Factura f:listaFacturas){
        if(f.tVehiculo.equalsIgnoreCase("Camion"))
            if(f.sentido.equalsIgnoreCase("salida"))
                cont++;
    }
    return cont;
}
```

imprimirFactura: Este método es de tipo de dato String. Recibe como parámetro una factura (tipo de dato Factura), cuenta con una variable llamada valor (tipo de dato double), a la cual se le asignará el valor obtenido por el método **calcularPeajeTotal**, al cual se le pasa como parámetro la factura que se le pasa como parámetro a **imprimirFactura**. Se crea un StringBuilder para poder concatenar la información obtenida de la factura con el valor de esta y poder mostrarla.

mostrarFactura: Este método es de tipo de dato String. Recibe como parámetro una fecha (tipo de dato Calendar). Cuenta con un StringBuilder. Dentro del método, hay un for each el cual cuenta con un iterador de tipo Factura que se encarga de recorrer cada posición del ArrayList **listaFacturas**. A continuación, hay un condicional que compara que la fecha pasada por parámetro concuerda con alguna de las fechas de las facturas que obtiene el iterador, cada factura que vaya encontrando se guarda en el StringBuilder y este las va concatenando. Al final del ciclo, se muestra el StringBuilder, en el cual se encuentran las facturas de dicha fecha, en caso que no existieran facturas de esa fecha, retorna el StringBuilder con su valor inicial, **null**.

```

public String mostrarFactura(Calendar fecha){
    StringBuilder sb = new StringBuilder();
    for(Factura fact: listaFacturas){
        if(fecha.getTime().getDate()==fact.fecha.getTime().getDate()){
            if(fecha.getTime().getMonth()==fact.fecha.getTime().getMonth()){
                if(fecha.getTime().getYear()==fact.fecha.getTime().getYear()){
                    sb.append(imprimirFactura(fact));
                    sb.append("\n-----\n");
                }
            }
        }
    }

    return sb.toString();
}

```

mostrarFactura: Este método es de tipo de dato String. Recibe como parámetro el número de la factura (tipo de dato int). Cuenta con dos variables: pos (tipo de dato int), que se le asigna lo obtenido por el método **buscarFactura**, y f (tipo de dato Factura). A continuación, se encuentra un condicional que valida que lo obtenido por pos sea igual a -1, si es así, retorna null, sino se cumple esta condición a f se le asigna la posición obtenida en el ArrayList **listaFacturas** y se retorna el método **imprimirFactura** que tiene como parámetro f que es la posición en la que está la factura.

```

public String mostrarFactura(int num){
    int pos;
    Factura f;
    pos=buscarFactura(num);
    if(pos==-1){
        return null;
    }
    else{
        f=listaFacturas.get(pos);
        return imprimirFactura(f);
    }
}

```

mostrarGanancias: Este método es de tipo de dato double. Cuenta con una variable llamada acum (tipo de dato double). Dentro de este, hay un for each en cual cuenta con un iterador de tipo de dato Factura que se encargara de recorrer cada posición del ArrayList **listaFacturas**. Dentro del ciclo, está el acumulador que se va aumentando cada vez que se le suma el método **calcularPeajeTotal**, que tiene como parámetro el iterador. Al final del ciclo, retorna acum.

```

public double mostrarGanancias(){
    double acum=0;

    for(Factura f: listaFacturas){
        acum+=calcularTotalPeaje(f);
    }

    return acum;
}

```

mostrarGanancias: Este método es de tipo de dato double. Recibe como parámetro una fecha (tipo de dato Calendar). Cuenta con una variable llamada acum (tipo de dato double). Dentro del método, hay un for each el cual cuenta con un iterador de tipo Factura que se

encarga de recorrer cada posición del ArrayList **listaFacturas**. En el ciclo se va aumentando el valor de acum debido a que se le va sumando el valor de cada factura por medio del método **calcularPeajeTotal** que se le pasa como parámetro el iterador. Al final del ciclo, se retorna el valor de acum que son las ganancias obtenidas en una fecha específica.


```
public double mostrarGanancias(Calendar fecha){
    double acum=0;

    for(Factura fact: listaFacturas){
        if(fecha.getTime().getDate()==fact.fecha.getTime().getDate())
            if(fecha.getTime().getMonth()==fact.fecha.getTime().getMonth())
                if(fecha.getTime().getYear()==fact.fecha.getTime().getYear())
                    acum+=calcularTotalPeaje(fact);
    }
    return acum;
}
```

mostrarInfoDriver: Este método es de tipo de dato String. Recibe como parámetro la identificación del conductor (tipo de dato String). Este cuenta con un for each, el cual utilizar un iterador de tipo de dato Conductor y recorre todas las posiciones del ArrayList **listaConductores**. En el ciclo, existe un condicional que valida que la identificación obtenida por parámetro se compare con cada una de las obtenidas por el iterador y en caso que alguna concuerde, se muestre la información del conductor, en caso de que no, retorne null.


```
public String mostrarInfoDriver(String id){
    for(Conductor c: listaConductores){
        if(c.getIdent().equalsIgnoreCase(id)){
            return c.toString();
        }
    }
    return null;
}
```

Camión:

 Camion :: Vehiculo

- ◆ Camion()
- ◆ Camion(String placa, String marca, int modelo, Fecha tecnoMec, int num)
- getCapCarga(): int
- getNumEjes(): int
- getSeguro(): boolean
- setCapCarga(int capCarga)
- setNumEjes(int numEjes)
- setSeguro(boolean seguro)
- toString(): String ↑ Vehiculo

Conductor:

 Conductor

- ◆ Conductor()
- ◆ Conductor(String nombre, String ident, long telefono, String ciudad)
- getCiudad() : String
- getIdent() : String
- getNombre() : String
- getTelefono() : long
- setCiudad(String ciudad)
- setIdent(String ident)
- setNombre(String nombre)
- setTelefono(long telefono)
- toString() : String ↑ Object

Factura:

 Factura

- ◆ Factura()
- ◆ Factura(Calendar fecha, String placa, String tVehiculo, String idConductor, boolean descuento, String sentido, int nf)
- getFecha() : Calendar
- getIdConductor() : String
- getNumFactura() : int
- getPlaca() : String
- getSentido() : String
- getTVehiculo() : String
- isDescuento() : boolean
- setDescuento(boolean descuento)
- setFecha(Calendar fecha)
- setIdConductor(String idConductor)
- setPlaca(String placa)
- setSentido(String sentido)
- getSentido() : String
- getTVehiculo() : String
- isDescuento() : boolean
- setDescuento(boolean descuento)
- setFecha(Calendar fecha)
- setIdConductor(String idConductor)
- setPlaca(String placa)
- setSentido(String sentido)
- setTVehiculo(String tVehiculo)
- toString() : String ↑ Object

Factura Camión:


```

FacturaCamion :: Factura
  FacturaCamion()
  FacturaCamion(int cargaActual, String descripCarga, String detalleDestino, Calendar fecha, String placa, String tVehiculo, String idConductor, boolean descuento, String sentido, int nf)
  getCargaActual() : int
  getDescripCarga() : String
  getDetalleDestino() : String
  setCargaActual(int cargaActual)
  setDescripCarga(String descripCarga)
  setDetalleDestino(String detalleDestino)
  toString() : String ↑ Factura

```

Vehículo:

```

Vehiculo
  Vehiculo()
  Vehiculo(String placa, String marca, int modelo, Calendar tecnoMec)
  getMarca() : String
  getModelo() : int
  getPlaca() : String
  getTecnoMec() : Calendar
  setMarca(String marca)
  setModelo(int modelo)
  setPlaca(String placa)
  setTecnoMec(Calendar tecnoMec)
  toString() : String ↑ Object

```

Vista

Métodos auxiliares de JFrame.

llenarCombo: Este método de dato void. Cuenta con un variable llamada c (tipo de dato Calendar), se le asigna una instancia de la clase hija GregorianCalendar. Cuenta con 4 ciclos for, los cuales cumplen diferentes funciones. El primer ciclo se utiliza un método llamado **getTime().getYear()** que retorna un entero que hace referencia al año actual menos 1900, es decir, estando en el 2017, retornará 117. La variable que controla se inicializa en 1901 más el valor que obtenemos del método para poder mostrar el año actual, la variable se va decrementando hasta llegar a 1990 y así permitir un rango de fechas entre 1990 y el año actual, este nos permitirá llenar el ComboBox perteneciente al modelo del vehículo y mostrar un posterior al actual. El segundo se encarga de llenar el ComboBox que hace referencia a los días, este inicializa la variable que lo controla en 0 y llega hasta 30, al momento de imprimir, se muestra la

variable más uno, para mostrar en pantalla los número del 1 a 31 (cantidad de días del mes). El tercero, se utilizar para llenar el ComboBox de meses, cuenta con una variable que se inicializa en 0 y llega hasta 11, en pantalla se muestra la variable más uno para indicar que los número van del 1 hasta el 12 (cantidad de meses del año). El cuarto ciclo, se utiliza un método **getTime().getYear()** anteriormente mencionado. La variable que controla se inicializa en 1900 más el valor que obtenemos del método para poder mostrar el año actual, la variable se va decrementando hasta llegar a 1990 y así permitir un rango de fechas entre 1990 y el año actual (ComboBox correspondiente a los años).

```
private void llenarCombo() {
    Calendar c= GregorianCalendar.getInstance();

    for(int i=1901+c.getTime().getYear();i>1960;i--){
        cbModelo.addItem(String.valueOf(i));
    }

    for(int i=0;i<31;i++){
        cbDia.addItem(String.valueOf(i+1));
        cbDia1.addItem(String.valueOf(i+1));
        dia.addItem(String.valueOf(i+1));
        dia1.addItem(String.valueOf(i+1));
    }

    for(int i=0;i<12;i++){
        cbMes.addItem(String.valueOf(i+1));
        cbMes1.addItem(String.valueOf(i+1));
        mes.addItem(String.valueOf(i+1));
        mes1.addItem(String.valueOf(i+1));
    }

    for(int i=1900+c.getTime().getYear();i>=1990;i--){
        cbYear.addItem(String.valueOf(i));
        cbYear1.addItem(String.valueOf(i));
        year.addItem(String.valueOf(i));
        year1.addItem(String.valueOf(i));
    }
}
```

comprobarDigito: Este método es de tipo de dato void. Recibe como parámetro un evento correspondiente a la tecla ingresada. Cuenta con una variable llamada carácter que se le asigna lo obtenido por el evento. A continuación, existe un condicional que valida que lo ingresado sea diferente a un dígito, en caso de que se cumpla, se invoca un método llamado **evt.Consume** el cual no permite mostrar en el textField la tecla presionada.

```
private void comprobarDigito(java.awt.event.KeyEvent evt){
    char caracter = evt.getKeyChar();
    if(!Character.isDigit(caracter)){
        evt.consume();
    }
}
```

comprobarNoDigito: Este método es de tipo de dato void. Recibe como parámetro un evento correspondiente a la tecla ingresada. Cuenta con una variable llamada carácter que se le asigna lo obtenido por el evento. A continuación, existe un condicional que valida que lo ingresado sea un dígito, en caso de que se cumpla, se invoca un método llamado **evt.Consume** el cual no permite mostrar en el textField la tecla presionada.

```
private void comprobarNoDigito(java.awt.event.KeyEvent evt){
    char caracter = evt.getKeyChar();
    if(Character.isDigit(caracter)){
        evt.consume();
    }
}
```

ocultarCamposAgFact: Este método es de tipo de dato void. Oculta los paneles factura y factura camión, los botones borrar y mostrar los coloca no disponibles y los textField de Identificación de conductor y Placa los permite editar.

```
private void ocultarCamposAgFact() {
    pnlFact.setVisible(false);
    pnlFactCam.setVisible(false);
    btnBorrarF.setEnabled(false);
    btnMostrarF.setEnabled(false);
    txtPlaca.setEditable(true);
    txtIdConductor.setEditable(true);
}
```

reiniciarCamposGan: Este método es de tipo de dato void. Modifica los ComboBox a su posición inicial (índice 0).

```
private void reiniciarCamposConsGan() {
    dia.setSelectedIndex(0);
    mes.setSelectedIndex(0);
    year.setSelectedIndex(0);
}
```

reiniciarCamposConsFact: Este método es de tipo de dato void. Borra el textField de Identificación consultaFactura y modifica los ComboBox a su posición inicial (índice 0).

```
private void reiniciarCamposConsFact() {
    txtIdCF.setText("");
    dial.setSelectedIndex(0);
    mes1.setSelectedIndex(0);
    year1.setSelectedIndex(0);
}
```

reiniciarCampoAgFact: Este método es de tipo de dato void. Borra los textField de Placa, conductor, cantidad actual, descripción y actividades. Además, coloca los ComboBox de día, mes, year y sentido en su posición inicial.

```
private void reiniciarCampoAgFact() {
    txtPlaca.setText("");
    txtIdConductor.setText("");
    txtCantidadActual.setText("");
    txtDescripcion.setText("");
    txtActividades.setText("");
    cbDia.setSelectedIndex(0);
    cbMes.setSelectedIndex(0);
    cbYear.setSelectedIndex(0);
    cbSentido.setSelectedIndex(0);
}
```

reiniciarCampoAgCond: Este método es de tipo de dato void. Borra los textField de Nombre, identificación, ciudad y teléfono.

```
private void reiniciarCampoAgCond() {  
    txtNombre.setText("");  
    txtId.setText("");  
    txtCiudad.setText("");  
    txtTel.setText("");  
}
```

reiniciarCampoAgVeh: Este método es de tipo de dato void. Borra los textField de Placa, marca, número de ejes y capacidad de carga. También reinicia la posiciones de los ComboBox de modelo, día, año y year a su inicial (índice 0). Además, en el CheckBox de seguro se deselectan las dos opciones.

```
private void reiniciarCampoAgVeh() {  
    txtPlacaV.setText("");  
    txtMarca.setText("");  
    cbModelo.setSelectedIndex(0);  
    cbDia.setSelectedIndex(0);  
    cbMes.setSelectedIndex(0);  
    cbYear1.setSelectedIndex(0);  
    txtNumEjesV.setText("");  
    txtCapCarga.setText("");  
    chbSi.setSelected(false);  
    chbNo.setSelected(false);  
}
```

chbSiMouseClicked: Este método es de tipo de dato void. Recibe como parámetro el evento del mouse. A continuación, contiene un condicional que verifica que si la opción si (chbSi) fue seleccionada, la otra se deselecta automáticamente. En caso de que no, indica que la otra está seleccionada.

```
private void chbSiMouseClicked(java.awt.event.MouseEvent evt) {  
    if(chbSi.isSelected()){  
        chbNo.setSelected(false);  
    }  
    else  
        chbNo.setSelected(true);  
}
```

chbNoMouseClicked: Este método es de tipo de dato void. Recibe como parámetro el evento del mouse. A continuación, contiene un condicional que verifica que si la opción no (chbNo) fue seleccionada, la otra se deselecciona automáticamente. En caso de que no, indica que la otra está seleccionada.

```
private void chbNoMouseClicked(java.awt.event.MouseEvent evt) {  
    if(chbNo.isSelected())  
        chbSi.setSelected(false);  
    else  
        chbSi.setSelected(true);  
}
```

btnContinuarMouseClicked: Este método es de tipo de dato void. Recibe como parámetro el evento generado por el mouse. Este cuenta con una estructura try catch. Dentro del try se encuentra declaradas las variables Placa (tipo de dato String), id (tipo de datos String), posV (tipo de dato int) al cual se le asigna la posición obtenida por el método **buscarVehiculo** que recibe de parámetro una placa, posC (tipo de dato int) al cual se le asigna la posición obtenida por el método **buscarConductor** al cual se le pasa por parámetro la id y veh (tipo de dato Vehículo). A continuación, se encuentra un condicional que valida que posV y posC sean diferentes de -1, si eso se cumple, habilita el panel de factura (pnlFact) y habilita los botones de mostrar (btnMostrarF) y borrar (btnBorrarF). Además, a veh se le asigna la posición obtenida por **ap.getListaVehiculos().get(posV)**. A continuación, existe un condicional que valida si veh es instancia de camión, en caso de que eso sea cierto, activa el panel de Camion (pnlCamion). Después, del condicional no se permite modificar los textField de Placa e Identificación del conductor. Si la condición anterior no se cumple, se envía un mensaje comunicando que no se encontraron los datos suministrados. En el bloque del catch se prueba con una excepción para verificar que todos los campos estén ingresados. En caso de que no, envía un mensaje pidiendo que ingrese todos los campos.

```
private void btnAgregarCMouseClicked(java.awt.event.MouseEvent evt) {  
    try{  
        String nc=txtNombre.getText();  
        String id=txtId.getText();  
        String city=txtCiudad.getText();  
        long tel=Long.parseLong(txtTel.getText());  
        Conductor c = new Conductor(nc, id, tel, city);  
  
        if(ap.agregarConductor(c)){  
            JOptionPane.showMessageDialog(null, "El Conductor ha sido agregado");  
        }  
        else  
            JOptionPane.showMessageDialog(null, "No se pudo agregar el nuevo conductor");  
  
        reiniciarCampoAgCond();  
    }catch(NumberFormatException e){  
        JOptionPane.showMessageDialog(null, "Rellene todos los campos");  
    }  
}
```

btnAgregarCMouseClicked: Este método es de tipo de dato void. Recibe como parámetro el evento generado por el mouse. Cuenta con un bloque try catch. Dentro del bloque try hay cinco variables. La primera es nc (tipo de dato String) que se le asigna el textField de Nombre (txtNombre), id (tipo de dato String) al cual se le asigna el textField de la identificación del conductor (txtId), city (tipo de dato String) se le asigna el textField de la residencia del

conductor (txtCiudad), tel (tipo de dato long) para asignarle el textField de teléfono (txtTel) se hace un casteo. Luego, se crea un objeto c de tipo de dato Conductor y al constructor se le pasan por parámetros las variables creadas anteriormente. A continuación, hay un condicional que valida que si **agregarConductor** que recibe como parámetro el objeto c devuelva un **true**, si es así, notifica que se agregó correctamente. En caso de que no, notifica que no se logró agregar y además, invoca el método **reiniciarCampoAgCond** para que los campos queden vacíos. En el bloque del catch, se prueba con una excepción y si se cumple, envía un mensaje diciéndole que llene todos los campos.

```
private void btnAgregarCMouseClicked(java.awt.event.MouseEvent evt) {
    try{
        String nc=txtNombre.getText();
        String id=txtId.getText();
        String city=txtCiudad.getText();
        long tel=Long.parseLong(txtTel.getText());
        Conductor c = new Conductor(nc, id, tel, city);

        if(ap.agregarConductor(c)){
            JOptionPane.showMessageDialog(null, "El Conductor ha sido agregado");
        }
        else
            JOptionPane.showMessageDialog(null, "No se pudo agregar el nuevo conductor");

        reiniciarCampoAgCond();
    }catch(NumberFormatException e){
        JOptionPane.showMessageDialog(null, "Rellene todos los campos");
    }
}
```

btnMostrarCMouseClicked: Este método es de tipo de dato void. Recibe como parámetro el evento que genera el mouse al momento de hacer click. Contiene un bloque try catch. Dentro del bloque try se encuentra una variable llamada id (tipo de dato String) que recibe lo obtenido en el textField de Identificación (txtId). A continuación, hay un condicional que valida el método **mostrarInfoDriver** que recibe como parámetro la identificación (id) no esté vacío, si eso se cumple, modifica el textArea y muestra la información del conductor. Si la condición no se cumple, notifica que no se encontró la identificación ingresada. En el bloque del catch se usa una excepción, en caso de que este, envía un mensaje pidiendo que llene todos los campos.

```
private void btnMostrarCMouseClicked(java.awt.event.MouseEvent evt) {
    try{

        String id=txtIdCC.getText();

        if(ap.mostrarInfoDriver(id)!=null)
            txtaCC.setText(ap.mostrarInfoDriver(id));
        else{
            JOptionPane.showMessageDialog(null, "La identificación ingresada no se encuentra registrada");
        }
    }catch(NumberFormatException e){
        JOptionPane.showMessageDialog(null, "Debe rellenar todos los campos");
    }
}
```

btnBorrarCCMouseClicked: Este método es de tipo de dato void. Recibe como parámetro el evento generado por el mouse. Vacía lo que tiene el textArea que contiene los datos del conductor y el textField de la identificación.

```
private void btnBorrarCCMouseClicked(java.awt.event.MouseEvent evt) {
    txtaCC.setText("");
    txtIdCC.setText("");
}
```

btnConsultarCFMouseClicked: Este método es de tipo de dato void. Recibe como parámetro el evento generado por el mouse. Cuenta con una estructura try catch. Dentro del try se crea una variable llamada nf (tipo de dato int) que recibe el textField de Identificación (id), para eso se hace un casteo para que el texto ingresado se comporte como entero. A continuación, hay un condicional que valida que el método **mostrarFactura** que recibe como parámetro el número de la factura (nf) no esté vacío. En caso de que sea así se modifica el textArea de Impresión factura y se muestra lo obtenido por **mostrarFactura**, en caso de que no notifica que el número de la factura no se encontró. En el bloque del catch se plantea un excepción, si se ejecuta envía un mensaje diciendo que llene todos los campos.

```
private void btnConsultarCFMouseClicked(java.awt.event.MouseEvent evt) {
    try{
        int nf= Integer.parseInt(txtIdCF.getText());

        if(ap.mostrarFactura(nf)!=null){
            txtaImpresion1.setText(ap.mostrarFactura(nf));
        }else{
            JOptionPane.showMessageDialog(null, "El número de Factura ingresado no se encuentra registrado.");
        }
        reiniciarCamposConsFact();
    }catch(NumberFormatException ex){
        JOptionPane.showMessageDialog(null, "Debe rellenar todos los campos");
    }
}
```

btnBorrarrCFMouseClicked: Este método es de tipo de dato void. Recibe como parámetro el evento generado por el mouse. Borra el contenido del textArea (txtaImpresion) de la consulta de la factura.

```
private void btnBorrarrCFMouseClicked(java.awt.event.MouseEvent evt) {
    txtaImpresion1.setText("");
}
```

btnLimpiarCGMouseClicked: Este método es de tipo de dato void. Recibe como parámetro el evento generado por el mouse. Borra los contenidos de los textField de ganancias por fecha (txtGananciasE) y ganancias generales (txtGananciasG). Además, invoca la función **reiniciarCamposConsGan** para que los borra lo que tienen los campos.

```
private void btnLimpiarCGMouseClicked(java.awt.event.MouseEvent evt) {
    txtGananciasE.setText("");
    txtGananciasG.setText("");
    reiniciarCamposConsGan();
}
```

btnConsultarElMouseClicked: Este método es de tipo de dato void. Recibe como parámetro el evento generado por el mouse. Cuenta con un bloque try catch. Dentro del try se crea una variable llamada valor (tipo de dato double) que se le asigna el método

mostrarGanancias. A continuación, hay un condicional que valida que valor sea diferente a cero, en ese caso, el textField de ganancias generales (txtGananciasE) se modifica y se le asigna el contenido de valor, para ello se hace un casteo para trabajar la variable como tipo de dato String. En caso de que el condicional no se haga efectivo, envía una notificación comunicando que no se registraron ganancias. En el catch prueba con una excepción, en caso de que se cumpla, se envía un mensaje notificando que ocurrió un error.

```
private void btnConsultarElMouseClicked(java.awt.event.MouseEvent evt) {
    try{
        double valor= ap.mostrarGanancias();
        if(valor!=0){
            txtGananciasG.setText((String.valueOf(valor)));
        }else JOptionPane.showMessageDialog(null, "No se han registrado ganancias");

        reiniciarCamposConsGan();
    }catch(NullPointerException e){
        JOptionPane.showMessageDialog(null, "Se ha producido un error");
    }
}
```

btnConsultarEMouseClicked: Este método es de tipo de dato void. Recibe como parámetro el evento generado por el mouse. Cuenta con un bloque try catch. Dentro del try se crean tres variables. La primera de dd (tipo de dato int) al cual se le asigna lo obtenido al momento de seleccionar un elemento del ítem de día, mm (tipo de dato int) al cual se le asigna lo obtenido al momento de seleccionar un elemento del ítem de mes y por último yy (tipo de dato int) al cual se le asigna lo obtenido al momento de seleccionar un elemento del ítem de año. A mm se le disminuye 1 debido a que en la vista los meses van de 1 a 12 pero en el código la clase Calendar los trabaja de 0 a 11. Se crea una variable c (tipo de dato Calendar) al cual se le asigna el calendario en base al calendario Gregoriano. Luego, por medio de c se obtiene la fecha ingresada por el usuario. Se crea una variable llamada valor (tipo de dato double) al cual se le asigna el método **mostrarGanancias**, luego hay un condicional que valida que lo obtenido por valor sea diferente de cero, si es así, se modifica el textField de Ganacias por fecha (txtGananciasE). En caso de que no, se notifica que no se obtuvieron ganancias. Después de esto se invoca el método **reiniciarCamposConsGan**, para vaciar los campos. En el catch se prueba con una excepción, en caso de que se cumpla se muestra un mensaje.

```
private void btnConsultarEMouseClicked(java.awt.event.MouseEvent evt) {
    try{
        int dd=Integer.parseInt(dia.getSelectedItem().toString());
        int mm=Integer.parseInt(mes.getSelectedItem().toString());
        int yyyy=Integer.parseInt(year.getSelectedItem().toString());
        mm--;
        Calendar c = GregorianCalendar.getInstance();
        c.set(yyyy, mm, dd);
        double valor=ap.mostrarGanancias(c);
        if(valor!=0){
            txtGananciasE.setText(String.valueOf(valor));
        }else{
            JOptionPane.showMessageDialog(null, "No se recaudaron ganancias a la fecha ingresada.");
        }
        reiniciarCamposConsGan();
    }catch(NumberFormatException e){
        JOptionPane.showMessageDialog(null, "Debe rellenar todos los campos.");
    }
}
```

btnMostrarVMouseClicked: Este método es de tipo de dato void. Recibe como parámetro el evento que genera el mouse. Contiene un bloque try catch. Dentro del try, se

crean tres variables, una llamada placa (tipo de dato String) a la cual se le asigna el textField de la placa del vehículo (txtPlacaVE), otra llamada pos (tipo de dato int) que se le asigna la posición obtenida por la función **buscarVehiculo** que recibe como parámetro la placa y por último tipo (tipo de dato String) que se inicializa con "Automóvil". A continuación, viene un condicional que valida que pos sea diferente a -1, en ese caso, se crea un objeto v de tipo de dato Vehículo y se le asigna la posición pos dentro del ArrayList **listaVehiculos**. Luego se hay otro condicional en el que se valida que v sea instancia de camión, si se cumple ese caso se cambia el contenido de tipo por Camión, además, se modifica el textArea y se le añade lo obtenido por tipo. Si pos es igual a -1, retorna un mensaje. En el catch hay una excepción que permite verificar si los campos están llenos, en caso de que falte uno, envía una notificación.

```
private void btnMostrarVMouseClicked(java.awt.event.MouseEvent evt) {
    try{
        String placa= txtPlacaVE.getText();
        int pos=ap.buscarVehiculo(placa);
        String tipo="AUTOMÓVIL";
        if(pos!=-1){
            Vehiculo v=ap.getListVehiculos().get(pos);
            if(v instanceof Camion){
                tipo="CAMIÓN";
            }
            txaVE.setText(v.toString()+"\nTIPO DE VEHÍCULO: "+tipo);
        }else JOptionPane.showMessageDialog(null, "No se encontró ningún vehículo con la placa ingresada.");
    }catch(NumberFormatException e){
        JOptionPane.showMessageDialog(null, "Debe rellenar los campos");
    }
}
```

btnBorrarMouseClicked: Este método es de tipo de dato void. Recibe como parámetro el evento generado por el click del mouse. Permite borrar el textArea de Consultar Vehículo (txaVE) que muestra la información de este y el textField en el cual se ingresa la placa (txtPlacaVE).

```
private void btnBorrarMouseClicked(java.awt.event.MouseEvent evt) {
    txaVE.setText("");
    txtPlacaVE.setText("");
}
```

jrAutomovilMouseClicked: Este método es de tipo de dato void. Recibe como parámetro el evento generado por el click del mouse. Este sea activa cuando el RadioButon de automóvil se selecciona, entonces hay un condicional que valida que si RadioButon es de camión está seleccionado, que los deseccione y oculte panel de camión. El otro condicional valida que si no se selecciona el RadioButon de Automóvil, se seleccione el RadioButon de Camión y se active el panel.

```
private void jrAutomovilMouseClicked(java.awt.event.MouseEvent evt) {
    if(jrCamion.isSelected()){
        jrCamion.setSelected(false);
        pnlCamion.setVisible(false);
    }
    else{
        pnlCamion.setVisible(false);
    }
    if(!jrAutomovil.isSelected()){
        jrCamion.setSelected(true);
        pnlCamion.setVisible(true);
    }
}
```

txtCantidadActualKeyTyped: Este método es de tipo de dato void. Recibe como parámetro el evento generado por el teclado. Se apoya en el método **comprobarDigito** para validar que lo único ingresado al momento de usar el textField de cantidad actual sean dígitos.

```
private void txtCantidadActualKeyTyped(java.awt.event.KeyEvent evt) {  
    comprobarDigito(evt);  
}
```

txtCiudadKeyTyped: Este método es de tipo de dato void. Recibe como parámetro el evento generado por el teclado. Se apoya en el método **comprobarNoDigito** para validar que lo único ingresado al momento de usar el textFiel de Ciudad no sean dígitos.

```
private void txtCiudadKeyTyped(java.awt.event.KeyEvent evt) {  
    comprobarNoDigito(evt);  
}
```

txtNumEjesVKeyTyped: Este método es de tipo de dato void. Recibe como parámetro el evento generado por el teclado. Se apoya en el método **comprobarDigito** para validar que lo único ingresado al momento de usar el textField de número de ejes sean dígitos.

```
private void txtNumEjesVKeyTyped(java.awt.event.KeyEvent evt) {  
    comprobarDigito(evt);  
}
```

txtCapCargaKeyTyped: Este método es de tipo de dato void. Recibe como parámetro el evento generado por el teclado. Se apoya en el método **comprobarDigito** para validar que lo único ingresado al momento de usar el textField de capacidad de carga sean dígitos.

```
private void txtCantidadActualKeyTyped(java.awt.event.KeyEvent evt) {  
    comprobarDigito(evt);  
}
```

txtIdCCKeyTyped: Este método es de tipo de dato void. Recibe como parámetro el evento generado por el teclado. Se apoya en el método **comprobarDigito** para validar que lo único ingresado al momento de usar el textField de Identificación de consulta conductor sean dígitos.

```
private void txtIdCCKeyTyped(java.awt.event.KeyEvent evt) {  
    comprobarDigito(evt);  
}
```

txtIdCFKeyTyped: Este método es de tipo de dato void. Recibe como parámetro el evento generado por el teclado. Se apoya en el método **comprobarDigito** para validar que lo único ingresado al momento de usar el textField de Identificación de consulta Factura sean dígitos.

```
private void txtIdCFKeyTyped(java.awt.event.KeyEvent evt) {  
    comprobarDigito(evt);  
}
```

txtIdConductorKeyTyped: Este método es de tipo de dato void. Recibe como parámetro el evento generado por el teclado. Se apoya en el método **comprobarDigito** para validar que lo único ingresado al momento de usar el textField de Identificación al momento de agregar al conductor sean dígitos.

```
private void txtIdConductorKeyTyped(java.awt.event.KeyEvent evt) {  
    comprobarDigito(evt);  
}
```

txtNombreKeyTyped: Este método es de tipo de dato void. Recibe como parámetro el evento generado por el teclado. Se apoya en el método **comprobarNoDigito** para validar que lo único ingresado al momento de usar el textField de nombre al momento de agregar al conductor no sean dígitos.

```
private void txtNombreKeyTyped(java.awt.event.KeyEvent evt) {  
    comprobarNoDigito(evt);  
}
```

txtTelKeyTyped: Este método es de tipo de dato void. Recibe como parámetro el evento generado por el teclado. Se apoya en el método **comprobarDigito** para validar que lo único ingresado al momento de usar el textField de teléfono al momento de agregar al conductor sean dígitos.

```
private void txtTelKeyTyped(java.awt.event.KeyEvent evt) {  
    comprobarDigito(evt);  
}
```