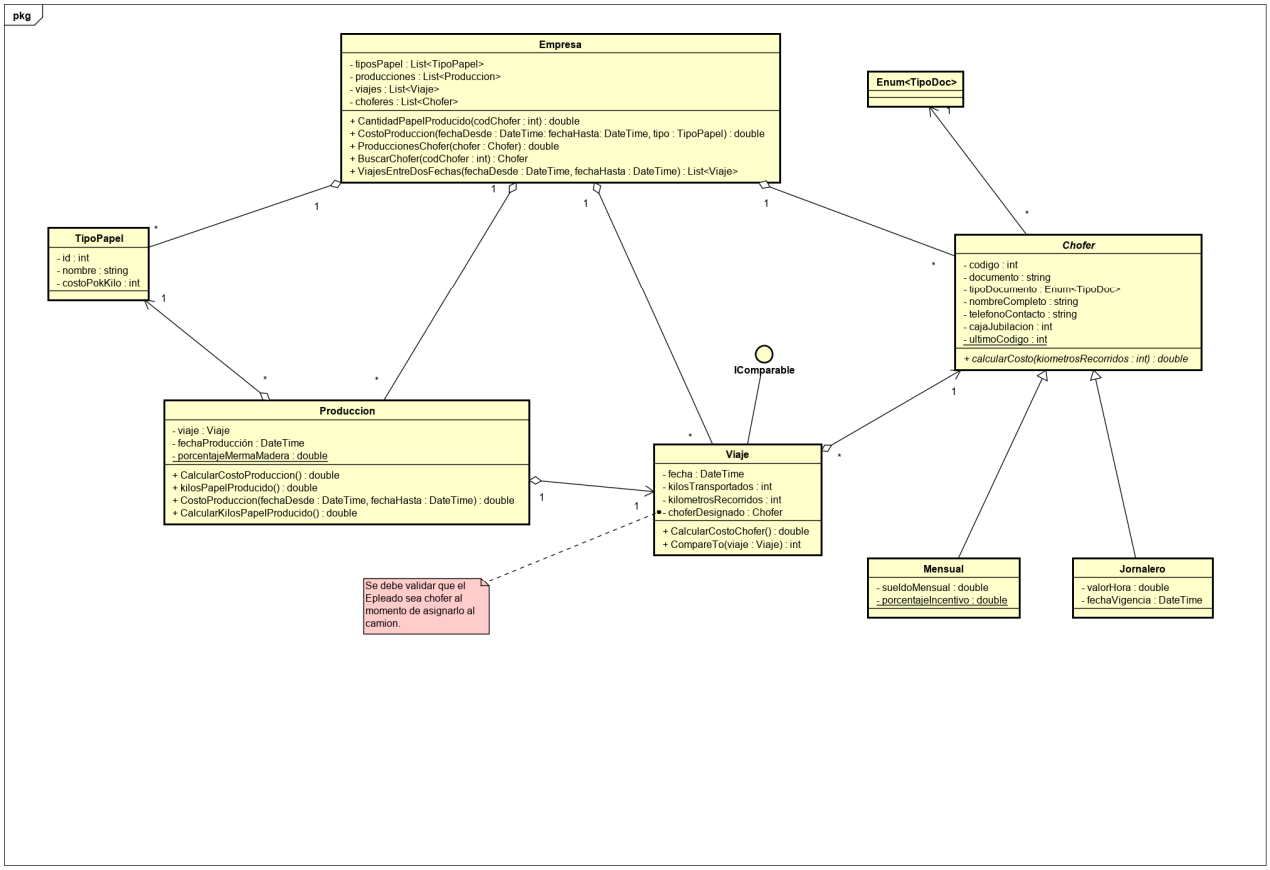
Solución Examen Octubre 2019 – Desde Marzo 2019



Dado un chofer, obtener la cantidad de papel producido considerando todas las producciones que se realizaron con madera que se trajo en viajes realizados por dicho chofer.

Empresa

```
public double CantidadPapelProducido (int codChofer){
        double cantidadPapel =0;
        Chofer chofer = BuscarChofer(codChofer);
        if(chofer!=null){
                cantidadPapel=ProduccionesChofer(chofer);
        }
        return cantidadPapel;
}
```

```
public double ProduccionesChofer(Chofer chofer){
        double cantidadPapel=0;
        foreach(Produccion produccion in Producciones){
                cantidadPapel+=produccion.KilosPapelProduccion();
        }
        return cantidadPapel;
}

public Chofer BuscarChofer(int codChofer){
        int i =0;
        Chofer chofer =null;
        while(i<choferes.Count && chofer==null){
                if(Choferes[i].Codigo==codChofer){
                        chofer=choferes[i];
        }
        i++;
        }
        return chofer;
}
```

Produccion

```
public double KilosPapelProduccion(Chofer chofer){
        double cantidadPapelProducido=0;
        if(viaje.Chofer.Equals(chofer)){
                cantidadPapelProducido=CalcularkilosPapelProducido;
        }
        return cantidadPapelProducido;
}
```

Dado un rango de fechas, mostrar el costo total de producción de papel de un tipo determinado. Para el cálculo de costo se debe tener en cuenta la cantidad de kilos de papel producidos

Sistema

```
public double CostoProduccion(DateTime fechaDesde, DateTime fechaHasta, int tipoPapel){
        double costoProduccionPapel =0;
        foreach(Produccion produccion in producciones){
        if(produccion.FechaProduccion>=fechaDesde && produccion.fecha <=fechaHasta){
                costoProduccionPapel+=produccion.CalcularCostoProduccion();
        }
        }
        return costoProduccionPapel;
}

public double CalcularkilosPapelProducido(){
double cantidadMermaPapel=viaje.KilosTransportados*Produccion.PorcentajeMermaMadera
/100;
double cantidadPapelProducido=viaje.KilosTransportados - cantididadMermaPapel;
cantidadPapelProducido=cantidadPapelProducido/250;
return cantidadPapelProducido;
}
```

Produccion

```
public double CalcularCostoProduccion(){
        double costoPapel=0;
        double kilosPapelProducido=CalcularkilosPapelProducido();
        costoPapel=kilosPapelProducido*tipoPapel.CostoKilo;
        double costoChofer=viaje.CalcularCostoChofer();
        costoPapel+=costoChofer;
        }
```

Viaje

```
public double CalcularCostoChofer(){
        return choferDesignado.CalcularCosto(kilometrosRecorridos);
}
```

Chofer

```
public abstract double CalcularCosto(int kilometrosRecorridos);
```

Jornalero

```
public override double CalcularCosto(int kilometrosRecorridos){
        double cantidadHorasTrabajadas=kilometrosRecorridos/100;
        return cantidadHorasTrabajadas*valorHora;
}
```

Mensual

```
public override double CalcularCosto(int kilometrosRecorridos){
        double cantidadHorasTrabajadas=kilometrosRecorridos/100;
        double costoMensual=0;
        if(kilometrosRecorridos>500){
                costoMensual=sueldoMensual/240 * (kilometrosRecorridos * 2);
        }
        else{
                costoMensual=sueldoMensual/240 * (kilometrosRecorridos);
        }
        double valorIncentivo=costoMensual * porcentajeIncentivo/100;
        return costoMensual + valorIncentivo;
}
```

Listar los viajes realizados entre dos fechas dadas ordenados por kilómetros recorridos en forma descendente.

Sistema.

```
public List<Viaje>ViajesEntreDosFechas(DateTime fechaDesde, DateTime fechaHasta){
        List<Viaje>viajesEntreFechas=new List<Viaje>();
                foreach(Viaje viaje in viajes){
                        if(viaje.Fecha>=fechaDesde && viaje.Fecha <=fechaHasta){
                                viajesEntreFechas.Add(viaje);
                        }
                }
        viajesEntreFechas.Sort();
        return viajesEntreFechas;
}
```

Viaje

```
public class Viaje : IComparable<Viaje>{
public override int CompareTo(Viaje viaje){
        return (kilometrosRecorridos - viaje.KilometrosRecorridos ) *-1;
}
}
```

Solucion requerimiento sin MVC

R04 – Dada una caja de jubilación, listar los choferes que pertenezcan a esa caja de jubilación y que hayan realizado los viajes con mayor cantidad de kilos transportados.

Sistema

```
public List<Chofer>EmpleadosQuePertenecenACaja(int caja){
List<Chofer>choferesCaja = new List<Chofer>();
int mayorCantidadKilos = int.MINVALUE;
foreach(Viaje viaje in viajes){
if(viaje.ChoferPerteneceACaja(caja)){
if(viaje.KilosTransportados>mayorCantidadKilos){
```

```
mayorCantidadKilos=viaje.KilosTransportados;
choferesCaja.Clear();
choferesCaja.Add(viaje.Chofer);
}
else if(mayorCantidadKilos==viaje.KilosTransportados){
if(!choferes.Contains(viaje.Chofer)){
choferesCaja.Add(viaje.Chofer);
}
}
}
}
return choferesCaja;
}
```

Viaje

```
public bool ChoferPerteneceACaja(int caja){
return choferDesignado.CajaJubilacion==caja;
}
```