Vamos a suponer que tenemos una fábrica de ensamblaje de Vehículos ( vehículos de Combustión y vehículos eléctricos)
Estos vehículos se pueden ensamblar a gusto del cliente, cada vehículo tiene un precio base y el precio de cada componente se acumula al precio base.

## Clases:

Se crearán las siguientes clases:

**Engine,** que será una clase que no podremos instanciar y tendrá como atributos:
String model, int power ( caballos), int basePrice ( en Euros), int maxSpeed.
Definirá un ***único*** constructor, al que le podrremos pasar los siguientes parámetros ( String model, int power, int basePrice, int maxSpeed).
Sus setters / getters

Se creará una clase **ElectricEngine,** que tendrá los mismos atributos que la clase Engine, además del atributo int comsumption ( watios por cada 100 km):

Definirá un ***único*** constructor, al que le podrremos pasar los siguientes parámetros ( String model, int power, int basePrice, int maxSpeed, int consumption).
Sus setters / getters

Se reescribirá el método toString de forma que tengamos una salida como la siguiente:

```
Electric Engine
****************
Model = 900
Power = 30
Max Speed = 45
Compsumption = 30
Price = 900
```

**CombustionEngine,** que tendrá los mismos atributos que la clase Engine, además de los atributos float comsumption ( medio por cada 100 km) y int engineDisplacement (Cilindrada):

Definirá un ***único*** constructor, al que le podrremos pasar los siguientes parámetros ( String model, int power, int basePrice, int maxSpeed, int engineDisplacement, float consumption).
Sus setters / getters

Se reescribirá el método toString de forma que tengamos una salida como la siguiente:

```
Combustion Engine
****************
Model = 5000
Power = 150
Engine Displacement = 1500
Max Speed = 150
Compsumptio = 5.6
Price = 5000
```

**Wheel,** que tendrá los atributos String model, int basePrice, int radio, int thickness.

Definirá un ***único*** constructor, al que le podrremos pasar los siguientes parámetros ( String model, int basePrice, int radio, int thickness).
Sus setters / getters

Se reescribirá el método toString de forma que tengamos una salida como la siguiente:

```
Wheel
****************
Model = Michelin Pilot Sport 5
Radio = 20; Thickness = 65; Price = 100
```

**Battery,** que tendrá los atributos String model, int basePrice, int capacity ( amperios), int power ( voltios), int batteryLevel ( porcentaje).

Definirá un ***único*** constructor, al que le podrremos pasar los siguientes parámetros ( String model, int basePrice, int capacity, int power).
Cuando instanciemos un objeto de la clase Battery, el nivel de batería siempre estará a cero.
Sus setters / getters

Se reescribirá el método toString de forma que tengamos una salida como la siguiente:

```
Battery
****************
Model = Battery Moto Yuasa
Power = 12; Capacity = 150; Batery Level = 0; Price = 34
```

**MotorizedVehicle,** que será una clase que no podremos instanciar y tendrá los atributos String model, int numWheels, int basePrice, ArrayList<Wheel> wheels, Engine engine.

Definirá un ***único*** constructor, al que le podrremos pasar los siguientes parámetros ( String model, int numWheels, int basePrice).
Cuando instanciemos un objeto de la clase Battery, el nivel de batería siempre estará a cero.
Sus setters / getters

Un método abstracto int getPrice().

El método void setWheels(ArrayList<Wheel> wheels). Al cual le podremos pasar un arrayList de Wheel, este método comprobará que el grosor y el radio de todas las ruedas sea igual, mostrando un mensaje informativo en caso contrario, y también comprobará que no ensamblamos más ruedas de las que permite el vehículo ( numWheels).
Si todo es correcto, ensamblará las ruedas del arrayList en nuestro vehículo.

El método addWheels(Wheel w)
Este método añadirá una rueda a nuestro vehículo, comprobando primero que las ruedas actuales, si es que hay, tienen el mismo radio y el mismo grosor, y que no tenemos el número máximo de ruedas en nuestro vehículo, mostrando un mensaje informativo en cada uno de los casos.

**CombustionVehicle,** que tendrá los mismos atributos que la clase MotorizedVehicle, además de los atributos int tankCapacity, int tankLevel.
Esta clase implementará la interfaz FillTank que se mencionará más adelante.

Definirá un ***único*** constructor, al que le podrremos pasar los siguientes parámetros ( String model, int numWheels, int basePrice, CombustionEngine engine, int tankCapacity).
Cuando instanciemos un objeto de la clase CombustionVehicle, el nivel del depósito siempre estará a cero.
Sus setters / getters

El método int getPrice() que nos devolverá el precio del vehículo en función del precio base y el precio de cada uno de los componentes del vehículo.

Se reescribirá el método toString de forma que tengamos una salida como la siguiente:

```
########################
Combustion Vehicle
########################
Model = Seat Arroba

Base Price = 9000

Combustion Engine
****************
Model = 5000
Power = 150
Engine Displacement = 1500
Max Speed = 150
Compsumptio = 5.6
Price = 5000

Weels

Wheel
****************
Model = Michelin Pilot Sport 5
Radio = 20; Thickness = 65; Price = 100

Wheel
****************
Model = Michelin Pilot Sport 5
Radio = 20; Thickness = 65; Price = 100

Wheel
****************
Model = Michelin Pilot Sport 5
Radio = 20; Thickness = 65; Price = 100

Wheel
****************
Model = Michelin Pilot Sport 5
Radio = 20; Thickness = 65; Price = 100


Total Price = 14400
```

**ElectricVehicle,** que tendrá los mismos atributos que la clase MotorizedVehicle, además de los atributos int numMaxBatterys, int batteryLevel, ArrayList<Battery> batteries.
Esta clase implementará la interfaz ChargeBattery que se mencionará más adelante.

Definirá un ***único*** constructor, al que le podrremos pasar los siguientes parámetros ( String model, int numWheels, int basePrice, ElectricEngine engine, int numMaxBatterys).
Cuando instanciemos un objeto de la clase ElectricVehicle, el nivel de batería siempre estará a cero.

Sus setters / getters

El método int getPrice() que nos devolverá el precio del vehículo en función del precio base y el precio de cada uno de los componentes del vehículo.

El método setBtteries(ArrayList<Battery> batteries ), que ensamblará un arrayList de baterías, siempre u cuando no excedamos el número máximo de baterías, en cuyo caso no las ensamblará y mostrará un mensaje informativo.

El método addBatery(Battery b), que nos añadirá una nueva batería a nuestro vehículo, siempre y cuando no sobrepasemos el número máximo de baterías, en cuyo caso no la ensamblará y mostrará un mensaje informativo.

El método float getBatteryLevel, que nos devolverá la media de los niveles de las baterías ensambladas.

Se reescribirá el método toString de forma que tengamos una salida como la siguiente:

```
########################
Electric Vehicle
########################
Model = Honda TZR

Base Price = 8000

Electric Engine
****************
Model = 800
Power = 25
Max Speed = 30
Compsumption = 25
Price = 800

Weels

Wheel
****************
Model = Bridgestone Trail Wing TW202
Radio = 39; Thickness = 15; Price = 84

Wheel
****************
Model = Bridgestone Trail Wing TW202
Radio = 39; Thickness = 15; Price = 84



Bateries

Battery
****************
Model = Battery Optima Redtop Rtc42
Power = 12; Capacity = 650; Batery Level = 0; Price = 198Battery
****************
Model = Battery Optima Redtop Rtc42
Power = 12; Capacity = 650; Batery Level = 0; Price = 198

Total Price = 9364
```

## Interfaces:

Se crearán las interfaces

**ChargeBattery**, que implementará un único método void charge, que pondrá todos los niveles de las baterías al 100%

**FillTank**, que implementará un único método void fill, que pondrá el nivel del depósito al 100%

Crearemos una clase principal donde crearemos al menos un objeto de cada clase
Se dejan los siguientes a modo de ejemplo

```
ElectricEngine me1 = new ElectricEngine("Magnetic Protection IP54",20, 700, 40, 20);
ElectricEngine me2 = new ElectricEngine("Vectorial Brusatori Protection VL132X",25, 800, 30, 25);
ElectricEngine me3 = new ElectricEngine("Vectorial Brusatori Protection VL315L",30, 900, 45, 30);


CombustionEngine mc1 = new CombustionEngine("Volkswagen 1.5 TFSI de 150 CV",150, 5000, 150, 1500, 5.6f);


CombustionEngine mc2 = new CombustionEngine("BMW PHEV xDrive25e X1",220, 8700,190,  1700, 3.9f);
CombustionEngine mc3 = new CombustionEngine("Ford Gasolina 1.0 EcoBoost 125 CV",120, 4300, 150, 1350, 3.5f);


Wheel w1 = new Wheel("Michelin Pilot Sport 5",100,20,65);
Wheel w2 = new Wheel("Pirelli Suv Scorpion R19",88,23,65);
Wheel w3 = new Wheel("Goodyear Efficient Grip Performance 195-65 ",75,25,85);
Wheel w4 = new Wheel("Bridgestone Trail Wing TW202 ",84,39,15);


ArrayList<Wheel> aw1 = new ArrayList<Wheel>();
for (int i=0; i<4 ; i++) {
         aw1.add(w1);
}
ArrayList<Wheel> aw2 = new ArrayList<Wheel>();
for (int i=0; i<4 ; i++) {
         aw2.add(w2);
}
Battery b1 = new Battery("Battery Moto Yuasa",34,150,12);
Battery b2 = new Battery("Battery Moto Bosch Yb5lb",29,160,12);
Battery b3 = new Battery("Battery Optima Redtop Rtc42",198,650,12);
Battery b4 = new Battery("Battery Tudor 55ah",80,475,12);

CombustionVehicle cv1 = new CombustionVehicle("Seat Arroba", 4, 9000, mc1, 50);
cv1.setWheels(aw1);
CombustionVehicle cv2 = new CombustionVehicle("Ford Apofis", 4, 11000, mc3, 54);
cv2.setWheels(aw2);


ElectricVehicle ev1 = new ElectricVehicle("BMW Serie 200",4, 14000, me1,30);
ArrayList<Wheel> aw3 = new ArrayList<Wheel>();
for (int i=0; i<4 ; i++) {
         aw3.add(w3);
}
ev1.setWheels(aw3);
ArrayList<Battery> ab3 = new ArrayList<Battery>();
for (int i=0; i<4 ; i++) {
         ab3.add(b1);
}
ev1.setBtteries(ab3);


ElectricVehicle ev2 = new ElectricVehicle("Honda TZR",2, 8000, me2,10);
ev2.addWheels(w4);
ev2.addWheels(w4);
ArrayList<Battery> ab4 = new ArrayList<Battery>();
for (int i=0; i<4 ; i++) {
         ab4.add(b3);
}
ev2.setBtteries(ab4);
```

Se creará un arrayList al que llamaremos **vehiculos** donde iremos añadiremos los vehículos creados.

Crearemos un método no  estático que nos muestre los vehículos del arrayList vehiculos.

Crearemos un método no estático que nos muestre los vehiculos  del arrayList **vehiculos** ordenados por precio de forma ascendente, para ello nos haremos uso de la interfaz **Comparable**.

Crearemos un método no estático que nos muestre los vehiculos  del arrayList **vehiculos** ordenados por la velocidad máxima de sus motores,  de forma ascendente, para ello haremos uso de la interfaz **Comparator**.

## Importante, Penalizaciones:

Cada atributo no privado -0,5 hasta un máximo de 3 puntos
toString no implementado correctamente --> - 1 cada uno hasta un máximo de 3 puntos
Más de un constructor en las clases -1 punto
Cualquier atributo que empiece con mayuscula -0,5 por atributo o -1 punto al examen si hay mas de uno
cualquier acento ñ ç que veamos -0,5 o -1 punto al examen si hay mas de uno

Clases que han de ser abstractas y no lo son -1 punto por clase
Cada método no implementado correctamente tal y como indica el enunciado -1 punto

Alguna de las clases no se implementa correctamente ( falla constructor, algun metodo, el tipo de un atributo) -1 punto por clase
Fallo compilacion -1 puntos
Cada clase no implementada -1 punto

Main
No se crea un arrayList de vehículos -1 punto
No se muestran vehículos ordenados por precio (Comparable) -1,5 punto
No se muestran vehículos ordenados por velocidad máxima ( Comparator) -1,5 punto