1. Retomamos la clase Vehículo y le hacemos las siguientes modificaciones:

Vehículo
+ numRuedas: entero + potencia: real + litrosEnDeposito: real **+ consumoPor100Km: real** + arrancado: lógico
+ setNumRuedas(nuevoNumRuedas: entero) + setPotencia (nuevaPotencia: real) **+ setConsumoPor100Km (nuevoConsumoPor100Km: real)** + reponerCombustible(numLitrosRepuesto: real) + recorrerDistancia (numKm: real) + arrancar() + apagar() **+ getNumRuedas() devuelve entero** **+ getPotencia() devuelve real** **+ getConsumoPor100Km() devuelve real** **+ getLitrosEnDeposito() devuelve real** **+ calculaPorcentajeDesgaste() devuelve real**

Donde:

- Los métodos *setXXX* establecen un nuevo valor en la propiedad correspondiente.
- En este caso la propiedad *consumoPor100Km* guarda el consumo de combustible por cada 100 km recorridos.
- Los métodos *arrancar* y *apagar* modifican el valor de la propiedad *arrancado*.
- El método *reponerCombustible* añade una cantidad de litros a los ya existentes en el depósito.
- El método *recorrerDistancia* modificar el valor de la propiedad *litrosEnDeposito* actualizándolo de modo que se aplique el *consumoPor100Km* proporcionalmente según los km recorridos.
- Los métodos *getXXXX* devuelve un número (al main) con el valor de la propiedad correspondiente.
- El método *calculaPorcentajeDesgaste ()* devuelve un número real que se corresponde con el tanto por ciento de desgaste del vehículo según el número de kilómetros recorrido. Para ello se considera que un vehículo con 0 km recorridos tiene un 0% de desgaste y un vehículo que ha recorrido 200.000 km tiene un 100% de desgaste.

Por último, realiza una clase *PruebaVehiculo* con un método *main* que permita probar un objeto de la clase Vehículo y que pruebe todos los métodos de esta clase.

OJO: La clase Vehículo no debe imprimir nada, todos los textos los debe imprimir el main.

2. Crea una clase llamada RelojDespertador que se corresponda con el siguiente diagrama UML.

RelojDespertador
¿?
+ setHora(hora: entero, minutos: entero) + setHoraAlarma(horaAlarm entero, minAlarm entero) + getHoraActual() devuelve entero + getMinutoActual() devuelve entero + getHoraAlarma() devuelve entero + getMinutoAlarma() devuelve entero + activarAlarma() + desactivarAlarma() + void sonarAlarma() + imprimirHoraActual() + imprimirHoraAlarma() + imprimirEstadoAlarma()

Tomando las siguientes consideraciones:

- Las propiedades debes deducirlas de la información asociada a los métodos. Pregúntate ¿qué información necesito guardar para que los métodos puedan hacer su trabajo.
- El método *setHora* establece la hora actual del reloj.
- El método *setHoraAlarma* establece la hora a la que sonará la alarma.
- Los métodos getXXX devolverán al *main* la información que corresponda.
- El método *activarAlarma* marcará de algún modo que la alarma está lista para sonar.
- El método *desactivarAlarma* marcará de algún modo que la alarma no va sonar.
- El método *sonarAlarma* imprimirá en pantalla el texto “PI PI PI PI… PI PI PI PI”
- Los métodos *imprimeXXXX*, imprimirán un texto y el valor de las propiedades que correspondan.

3. Crea una clase llamada CajaRegistradora que se corresponda con el siguiente diagrama UML.

CajaRegistradora
+ importeCliente: real + numArticulosCliente: entero
+ abrirCaja() + nuevoCliente() + registrarArticulo(precio: real) + imprimirTicketCliente()

Vamos a simular el comportamiento simplificado de una caja registradora de supermercado, tomando las siguientes consideraciones:

- La caja se abre por la mañana con *abrirCaja* y se ponen todas las propiedades a cero.
- Cuando la caja recibe a un cliente entonces se ejecuta el método *nuevoCliente* que debe poner a cero tanto el número de artículos registrados para ese cliente (*numArticulosCliente*) como el importe total que el cliente va a pagar (*importeCliente)*.
- Cada vez que el cliente ponga un artículo encima de la caja registradora se ejecutará el método *registrarArticulo* que debe incrementar en 1 el número de artículos registrados para ese cliente (*numArticulosCliente*) y también debe actualizar la propiedad *importeCliente* incrementando el valor que tuviera con el del *precio* del producto.
- Por último, el método *imprimirTicketCliente* imprime un mensaje que indica "El cliente ha comprado XX artículos por un precio total de XXX.XXX euros"

Ahora vamos a utilizar nuestra caja. Para ello creamos una clase *PruebaCajaRegistradora* que tenga un método *main* que realice las siguientes acciones en orden:

- Declara y crea un objeto CajaRegistradora.
- Se abre la caja
- Llega el primer cliente
- Pasa un artículo con precio 12.95
- Pasa un artículo con precio 2.48
- Pasa un artículo con precio 20.06
- Se imprime el ticket del cliente
- Llega otro cliente
- Pasa un artículo con precio 5.95
- Pasa un artículo con precio 2.48
- Pasa un artículo con precio 2.48
- Pasa un artículo con precio 7.88
- Se imprime el ticket del cliente

Haz las sumas con una calculadora y comprueba si el programa ofrece los resultados correctos.

4. Vamos a dotar de más complejidad a la clase anterior para ello crearemos una clase nueva llamada CajaCompleja que se corresponda con el siguiente diagrama UML:

CajaCompleja
+ importeCliente: real + numArticulosCliente: entero **+ numClientesAtendidos: entero** **+ importeTotalCaja: real** **+ numArticulosVendidos: entero**
+ abrirCaja() + nuevoCliente() + registrarArticulo(precio: real) **+ anularArticulo(precio: real)** + imprimirTicketCliente() **+ imprimeCierreCaja()** **+ calculaPrecioMedioArticulosVendidos() devuelve real** **+ calculaImporteMedioPorCliente() devuelve real**

OJO: Aunque he marcado en **negrita** las novedades, vas a tener que retocar también métodos que ya existían en la versión anterior. Las novedades son las siguientes:

- La propiedad *numClientesAtendidos* llevará la cuenta de los clientes que se han atendido a lo largo del día, incrementándose en 1 cada vez que se ejecute el método *nuevoCliente.*
- Además, del *numArticulosCliente* que lleva la cuenta del número de artículos comprados por un cliente, ahora también queremos llevar la cuenta del total de artículos vendidos a todos los clientes durante el día. Para ello, usaremos la propiedad *numArticulosVendidos*. Deberás retocar el método *registrarArticulo* para que contemple esta novedad.
- Además, del *importeCliente* que lleva la cuenta del importe que debe pagar el cliente al ir pasando sus artículos por la caja, ahora también queremos llevar el saldo total que debe haber en la caja. Para ello, usaremos la propiedad *importeTotalCaja*. Deberás retocar el método *registrarArticulo* para que contemple esta novedad.
- El método *anularArticulo* permite al cliente anular la compra de un artículo. Para ello debe restar el precio que recibe como parámetro de la propiedad *importeCliente.* Plantéate cómo afecta este método a las propiedades *importeTotalCaja* y *numArticulosVendidos.*
- El método *imprimeCierreCaja* mostrará por pantalla un mensaje como el siguiente: “Se han vendido un total de XX artículos por un importe total de XXX.XX euros”
- El método *calculaPrecioMedioArticulosVendidos* calculará y devolverá el precio promedio de los artículos vendidos durante el día.
- El método *calculaImporteMedioPorCliente* calculará y devolverá el importe promedio que ha gastado cada cliente durante el día.

Crea una clase *PruebaCajaCompleja* que tenga un método *main* que te permita comprobar el correcto funcionamiento de todos los métodos, tanto los antiguos como los nuevos.