EJERCICIOS EXTRA – COMPOSICIÓN DE OBJETOS

1. Vamos a crear dos clases que se relacionen mediante composición y que "se asocien mutuamente", atendiendo al siguiente modelo:

Cliente
- nombre: texto - dni: texto - direccionPostal: texto - email: texto - usuario: Usuario
+ Cliente (nombre: texto, dni: texto, Usuario usu) + Cliente (nombre: texto, dni: texto) + getters y setters de todas las propiedades

Usuario
- nombre: texto - password: texto - cliente: Cliente
+ Usuario (nombre: texto, pass: texto, Cliente cli) + getters y setters de todas las propiedades

Queremos que nuestro modelo permita crear un cliente sin un usuario asociado, pero NO permita crear un usuario sin cliente asociado. Para ello añadiremos la siguiente lógica a los constructores:

- Si se intenta crear un usuario y nos pasan un *null* como valor para la propiedad cliente entonces se debe emitir el mensaje "Advertencia: este usuario se asociará a un cliente anónimo" entonces se creará un nuevo objeto Cliente que tenga el valor "anónimo" en todos los textos y que su usuario sea el que se está creando (pista: this).
- Sin embargo, si se intenta crear un cliente y nos pasan un *null* como valor para la propiedad usuario entonces se creará el cliente con total normalidad, almacenando un *null* en la propiedad usuario.

Crea una clase de pruebas con un método *main* que realice:

- Crea un cliente sin usuario llamado "Pepe", con el DNI que quieras.
- Crea un usuario de nombre "pepeuser" con la password que quieras asociado al cliente "Pepe".
- Crea un usuario "juanuser", con la contraseña que quieras y que no se asocie a ningún cliente.
- Ahora pídele al usuario anterior que te devuelva una referencia al "cliente anónimo" que tiene asociado utilizando el método *getCliente().*
- Imprime el nombre y DNI del "cliente anónimo" obtenido.
- Crea un cliente sin usuario llamado "Juan", con el DNI que quieras.
- Ahora utiliza el método *setCliente(…)* del usuario "juanuser" para que se le asocie el cliente "Juan".
- Vuelve a pedirle que te devuelva la referencia con *getCliente()* e imprime su nombre y DNI.

2. Vamos a crear tres clases que utilicen la composición para relacionarse. Para ello simularemos que tenemos un registro de las ventas de coches a cliente, atendiendo al siguiente modelo:

Cliente
- nombre: texto - dni: texto - direccionPostal: texto - email: texto
+ Cliente (nombre: texto, dni: texto) + getters y setters de todas las propiedades

Coche
- marca: texto - modelo: texto - matricula: texto - cliente: Cliente - concesionario: Concesionario
+ Coche (marca: texto, modelo: texto, matricula: texto, Cliente cli, Concesionario con) + getters y setters de todas las propiedades

Concesionario
- nombre: texto - cif: texto - direccionPostal: texto - email: texto
+ Concesionario (nombre: texto, cif: texto, direccionPostal: texto) + getters y setters de todas las propiedades

Crea una clase de pruebas con un método *main* que realice las siguientes acciones:

- Crea tres clientes llamados “John”, “Anne” y “Tom”, con el DNI que quieras.
- Crea un concesionario con el nombre “Concesur”, invéntate el resto de los datos.
- Crea un concesionario con el nombre “TuCoche”, invéntate el resto de los datos.
- Ahora crea un coche Seat Ibiza con matrícula “1234-IBZ” que haya sido comprador por John en el concesionario “Concesur”.
- Ahora crea un coche Opel Corsa con matrícula “4321-COR” que haya sido comprador por Anne en el concesionario “TuCoche”.
- Ahora crea un coche Seat Toledo con matrícula “5678-TOL” que haya sido comprador por Tom en el concesionario “Concesur”.
- Ahora crea un coche Opel Meriva con matrícula “8765-MER” que haya sido comprador por John en el concesionario “TuCoche”.

3. Crea las siguientes clases:

Moneda
- nombre: texto - simbolo: texto
+ constructor con todos los parámetros + getters todas las propiedades

Dinero
- cantidad: real - moneda: Moneda
+ constructor con todos los parámetros + getters todas las propiedades + toString() devuelve texto

La clase Moneda representará la moneda de un país y la clase Dinero representará una cantidad de moneda. Observa el siguiente ejemplo de cómo podrían crearse objetos de ambas clases:

```
Moneda euro = new Moneda("euro", "€");
Dinero precioCamiseta = new Dinero(12.99, euro);
```

También podríamos hacerlo con una notación más abreviada:

```
Dinero precioTablet = new Dinero(125.99, new Moneda("euro", "€"));
```

La cantidad de la clase Dinero siempre deberá ser positiva, en el caso de que se quiera crear un objeto Dinero con una cantidad negativa, se le cambiará el signo para que se guarde siempre un número positivo. Si se da esta situación se debe imprimir un mensaje de advertencia en la consola.

El método toString() de la clase Dinero debe devolver una cadena de texto que se corresponda con la concatenación de la cantidad y el símbolo de la moneda. Observa:

```
System.out.println("Precio: "+precioTablet.toString());
```

Se imprime:

```
Precio: 125.99€
```

Crea una clase de pruebas con un método *main* que realice las siguientes acciones:

- Crea dos objetos moneda uno para el "euro" y otro para el "dólar"
- Después crea dos objetos dinero con los valores 300.50€ y -128.99$
- Imprime los objetos en pantalla.
- Crea otro objeto con la notación más abreviada que se corresponda con 12.78Fr (francos suizos).

***Nota de interés**: este tipo de objetos se les llama "value object" y se usan en la industria como objetos "inmutables", es decir que no se les puede modificar una vez creados (observa que no tienen setters)*

4. Continuando con el ejercicio anterior, añade una nueva clase:

Articulo
- código: entero - nombre: texto - precio: Dinero
+ constructor con todos los parámetros + getters todas las propiedades + setters de *nombre* y *precio* + toString() devuelve texto

La clase Articulo representará un artículo de una tienda online internacional. El método toString() devolverá una cadena con el formato:

Artículo *código / nombre / precio*

Crea una clase de pruebas con un método *main* que realice las siguientes acciones:

- Crea un artículo con valores: 1, "Auriculares SONY FW22", 30€
- Crea otro artículo con valores: 2, "Teclado Mehosy RT302", 50€
- Crea otro artículo con valores: 3, "Ratón inalámbrico Delta 3", 25€
- A continuación, imprime en pantalla los tres artículos.
- Súbele el precio al primer artículo a 39.00€ y vuélvelo a imprimir.
- Cambia el precio del segundo artículo a -50€ y vuélvelo a imprimir.

5. ¿Serías capaz de crear una clase *Carrito* que tuviera un array capaz de almacenar hasta 10 objetos Articulo? Su diagrama podría ser algo así:

Carrito
- array: Articulo - numArticulos: entero
+ constructor sin parámetros + agregarArticulo(Articulo a) + calcularPrecioTotal() devuelve real + mostrarCarrito() devuelve real

- El constructor debe crear el array (hacer el new…)
- La propiedad *numArticulos* nos servirá para saber cuántos artículos hay almacenados en el carrito.
- Al llamar a agregarArticulo se usará la propiedad *numArticulos* como punto de inserción en el array, siempre que no se haya alcanzado el tamaño máximo. Además se incrementará la propiedad *numArticulos* para que refleje que ahora hay un artículo más.
- mostrarCarrito() recorrerá el carrito y llamará al método toString() de cada artículo.