Tema 3 – Introducción a la programación orientada a objetos -

Ejercicios II

1. **Crea una clase llamada Empleado:**

   - Con los **atributos privados** *nombre*, y *teléfono*.
   - Implementa los **métodos** necesarios para acceder a estos dos atributos de manera **pública** (*getters* y *setters*).
   - Añade el **atributo estático y privado** *numEmpleados*. Este atributo almacenará el número de instancias que se han creado de la clase *Empleado*.
   - Implementa un **método público** para conocer el valor de *numEmpleados*.
   - Crea una clase aparte con el método *main* para probar todas las funcionalidades de la clase *Empleado*.

2. **En un almacén se guarda fruta para su posterior venta. Por cada cargamento se tiene la siguiente información: nombre de la fruta, procedencia, número de kilos, precio coste por kilo y precio venta por kilo. Codificar una clase para manejar esta información de forma que contenga las siguientes operaciones:**

   - Constructor
   - Método que devuelva la información de cada cargamento de fruta.
   - Método "**rebajar**" que rebaja el precio de venta en una cantidad pasada como parámetro, (el precio de venta nunca puede ser menor que el precio de coste).
   - Método "**vender**": se le pasa el número de kilos a vender y si hay suficiente cantidad, se decrementa el número de kilos y se devuelve el importe de la venta, sino da error.

- Método que nos diga si dos cargamentos de fruta tienen la misma procedencia.
- Llevar en todo momento el beneficio obtenido por el almacén.

Para probar dicha clase hacer un main que:

- Dé de alta 3 cargamentos y muestre su información.
- Diga si los dos primeros tienen la misma procedencia.
- Rebaje el precio del tercero.
- Realice ventas de los tres cargamentos.
- Muestre el beneficio obtenido por el almacén.

3. **En una agencia de viajes se guarda la siguiente información por cada viaje: Ciudad de origen y ciudad de destino, código, importe del viaje por persona, número de adultos, número de niños. Se pide hacer una clase Viaje que contenga al menos los siguientes métodos:**

   - Constructor. El código del viaje se forma con las dos primeras letras del origen (en mayúsculas), las dos primeras letras del destino (en mayúsculas) y el número de viaje.
     Nota: hay que usar funciones de la clase String
   - Método que **reserve** un número de plazas pasado como parámetro. Si no tiene plazas suficientes no reserva ninguna. Los niños tienen un 20% de descuento sobre el valor del billete. Devuelve el importe total de la reserva, 0 si no se puede realizar.
   - Método que **modifique** el número de plazas de un viaje, sólo si es posible, es decir nunca puede haber más reservas que plazas. Retorna si se ha podido realizar o no la modificación.
   - Método toString.
   - Método que muestre el total recaudado por la agencia.

   Codificar un main que realice las siguientes operaciones:

   - Crear dos viajes.
   - Mostrad la información de ambos viajes.
   - Reservad varias plazas de ambos viajes.
   - Modificad el número de plazas del primer viaje.
   - Mostrad de nuevo la información de ambos viajes.
   - Mostrad el total recaudado por la agencia.

4. **Por cada alumno del DAW guardamos su número de matrícula y sus notas en tres asignaturas. Hacer una clase alumno que guarde esta información. Realizar un programa que cree tres alumnos, les asigne notas y a continuación los muestre ordenados de mayor a menor nota media.**

5. **Al ejercicio anterior añadirle una variable que guarde el número de alumnos con nota media aprobada.**

6. **Crea una clase llamada Vehículo:**

   - Con los **atributos privados** *numRuedas* (número de ruedas), *velMax* (velocidad máxima) y *peso*.

   - Implementa los **métodos** necesarios para acceder a estos atributos de manera **pública** (*getters* y *setters*).

   - La clase dispondrá de un **constructor** que necesitará como parámetros los valores iniciales para todos sus atributos.

   - Crea un **método** público *boolean esIgual(Vehiculo)* que sirva para comparar dos vehículos, de manera que devuelva true o false dependiendo de si son iguales o no (tienen todos sus atributos el mismo valor o no). El método recibirá como parámetro un objeto de la clase *Vehículo*.

   - Crea un **método** público void *copia(Vehiculo)* que copiará un vehículo en otro. El método recibirá como parámetro un objeto de la clase *Vehículo* del cual se copiarán sus valores.

   - Crea una clase aparte con el método *main* para probar todas las funcionalidades de la clase *Vehículo*.

7. **Crea una clase llamada Empleado:**

   - Con los atributos privados nombre, y teléfono.
   - Implementa los métodos necesarios para acceder a estos dos atributos de manera pública (getters y setters).
   - Implementa un **constructor de copia** que permita copiar un objeto de tipo Empleado en otro.
   - Implementa el método **toString()**.
   - Implementa el método **equals(Object o)** para comprobar si dos objetos de tipo Empleado son iguales.
   - Añade el atributo estático y privado numEmpleados. Este atributo almacenará el número de instancias que se han creado de la clase Empleado.
   - Implementa un método público para conocer el valor de numEmpleados.
   - Crea una clase aparte con el método main para probar todas las funcionalidades de la clase Empleado.

8. **Crea una clase llamada Cuenta:**
   - Tendrá los siguientes atributos: titular y cantidad (puede tener decimales).
   - El titular será obligatorio y la cantidad es opcional. Crea dos constructores que cumpla lo anterior.
   - Implementar un constructor de copia que permita crear una cuenta exactamente igual que otra cuenta ya existente.
   - Crea sus métodos get, set, toString y equals.
   - Tendrá dos métodos especiales:
     - **ingresar(double cantidad):** se ingresa una cantidad a la cuenta, si la cantidad introducida es negativa, no se hará nada.
     - **retirar(double cantidad):** se retira una cantidad a la cuenta, si restando la cantidad actual a la que nos pasan es negativa, la cantidad de la cuenta pasa a ser 0.

9. **Haz una clase llamada Persona que siga las siguientes condiciones:**
   - Sus atributos son: nombre, edad, DNI, sexo (H hombre, M mujer), peso y altura. No queremos que se accedan directamente a ellos. Piensa que modificador de acceso es el más adecuado, también su tipo. Si quieres añadir algún atributo puedes hacerlo.
   - Por defecto, todos los atributos menos el DNI serán valores por defecto según su tipo (0 números, cadena vacía para String, etc.). Sexo será hombre por defecto, usa una constante para ello.

- Se implantarán varios constructores:
  - Un constructor por defecto.
  - Un constructor con el nombre, edad y sexo, el resto por defecto.
  - Un constructor con todos los atributos como parámetro.
- Los métodos que se implementaran son:
  - **calcularIMC():** calculara si la persona está en su peso ideal (peso en kg/(altura^2 en m)), si esta fórmula devuelve un valor menor que 20, la función devuelve un -1, si devuelve un número entre 20 y 25 (incluidos), significa que está por debajo de su peso ideal la función devuelve un 0 y si devuelve un valor mayor que 25 significa que tiene sobrepeso, la función devuelve un 1.
  - **esMayorDeEdad():** indica si es mayor de edad, devuelve un booleano.
  - **comprobarSexo(char sexo):** comprueba que el sexo introducido es correcto. Si no es correcto, será H. No será visible al exterior.
  - **toString():** devuelve toda la información del objeto.
  - **equals(Object o):** devuelve true si dos personas son iguales, false en caso contrario.

- Ahora, crea una clase ejecutable que haga lo siguiente:
  - Pide por teclado el nombre, la edad, sexo, peso y altura.
  - Crea 3 objetos de la clase anterior, el primer objeto se creará teniendo en cuenta las anteriores variables pedidas por teclado, el segundo objeto tendrá como valor de sus atributos todos los anteriores menos el peso y la altura y el último se creará sin parámetros. Después utilizar los métodos set para darle a los atributos un valor.
  - Para cada objeto, deberá comprobar si está en su peso ideal, tiene sobrepeso o por debajo de su peso ideal con un mensaje.
  - Indicar para cada objeto si es mayor de edad.
  - Por último, mostrar la información de cada objeto.

10. **(Difícil)** Crea una clase **Carro** de la compra que contenga los datos del cliente: nombre, dirección y NIF. Al carro se podrá añadir un producto. El **Producto** debe guardar el nombre, el precio base y el tipo de I.V.A. que se le puede aplicar (10%, 17% o 21% según sea un producto básico, ordinario o de lujo).

Cuando el cliente decida realizar el pedido se mostrará por pantalla la lista de todos sus datos y los valores del producto, con su precio base y precio con I.V.A.
Para las dos clases anteriores piensa qué criterios debe cumplir cada uno de los campos. En la clase principal (la que tenga el método main) comprueba si tu clase carro de la compra funciona correctamente.
Prueba todo el programa haciendo que el método main añada un producto al carro y luego muestre por pantalla la factura final.

11. Vamosa crear una clase de **BilletesAvión.** Cada billete tendrá un nombre y un NIF de viajero que puede viajar en una de las tres clases: turista, business o VIP. Cuando llegue a facturar se le asignará un asiento determinado. Además, puede haber contratado el catering o no, el poder ir al baño o no. También se debe guardar el número de maletas que lleva. Un cliente puede hacer una reserva inicial dando solo su nombre y nif o eligiendo una clase además.

12. Modifica la clase anterior para establecer **costes**. Un billete en turista costará 200€, en business 400 € y en VIP 600 €. Tanto business como VIP llevan el catering y el baño incluidos. Para los viajeros de clase turista que deseen contratar estos servicios, el catering costará 20€ y el baño 10€. Además la clase turista llevará una maleta incluida, business tres y VIP no tendrá límite en el número de maletas. Cada maleta adicional por encima del límite en la clase correspondiente costará 50 €. El viajero podrá consultar en cualquier momento el coste total del billete.

13. Implementa un método que recibe un número variable de parámetros de entrada de tipo String e imprima el número de parámetros recibidos y a continuación la posición del parámetro y su valor. Ejemplo:

Llamada al método:

```
pruebaParametros('Ana', 'Juan', 'Adrián');
```

Salida:

```
El número de parámetros es 3
Posición 0: Ana
Posición 1: Juan
Posición 2: Adrián
```

14. Implementa un método que sume todos los parámetros (números enteros) que le lleguen (no se conoce de antemano el número de parámetros).

15. Sobrecarga el método anterior de tal manera que reciba de forma fija 2 números reales pudiendo venir a continuación más números enteros como parámetros o no. El método devolverá la suma de todos los parámetros que se le pasen en la llamada.

16. Crea una clase llamada **Calcula**. Implementa los siguientes métodos estáticos, que devolverán en cada caso el menor o mayor número de los pasados como parámetros. En el caso de las cadenas devolverá la de mayor o menor longitud.

- ***int mayor(int … num)***
- ***float mayor(float … num)***
- ***String mayor(String … cadena)***
- ***int menor(int … num)***
- ***float menor(float … num)***
- ***String menor(String … cadena)***

Crea una clase aparte con el método ***main*** para probar todas las funcionalidades de la clase *Calcula*.

17.Crea una clase llamada Nombres**, capaz de gestionar una lista de nombres de un tamaño determinado:**

- El constructor recibirá como parámetro el número máximo de nombres que albergará.
- Tendrá los siguientes métodos:
- int anadir(String): Añade a la lista el nombre pasado como parámetro. Devuelve -1 si la lista está llena, 0 si se añade con éxito y 1 si ya existía el nombre (no admite repetidos).
- boolean eliminar(String): Elimina de la lista el nombre pasado como parámetro. Devuelve true en caso de éxito y false si no encuentra el nombre a eliminar.
- void vaciar(): Elimina todos los nombres de la lista.
- String mostrar(int): Devuelve el nombre que se encuentra en la posición pasada como parámetro (la primera es la posición 0).
- int numNombres(): Devuelve el número de nombres que hay actualmente.

- int maxNombres(): Devuelve el número máximo de nombres que puede albergar.
- boolean estaLlena(): Devuelve true si la lista está llena y false en caso contrario.