Ejercicios de Clases abstractas en POO

Relación de ejercicios

Ejercicio 1. Implementa una clase Infante que permita almacenar la fecha de nacimiento y nombre de un niño o niña. Deben existir las siguientes tres clases que hereden de ellas:

a) Lactante(`TipoLactancia` *tipoLactancia*). La lactancia puede ser directa o mixta.

b) Infantil. Se deberá almacenar el nombre de la guardería y si es alérgico o no.

c) Escolar. Se debe almacenar el nombre del colegio y el nombre de tres deportes que practique. Todos los escolares practican siempre tres deportes.

Además, debemos implementar los siguientes métodos (debes decidir en qué lugar es más apropiado implementarlos).

- Todos los infantes comen. Implementa un método *comer()* que responda: "Estoy comiendo".
- Todos los infantes duermen. Implementa un método *dormir()* que responda "Estoy durmiendo".
- Implementa un método que devuelva la edad del infante.
- Todos los infantes tienen que tener un método *hablar()* que debe ser implementado en cada una de las clases hijas (es decir, tiene que ser abstracto). Cada clase hija dirá algo coherente con el tipo de clase que es: "No sé hablar", "Hablo con frases sencillas" o "Hablo muy bien", respectivamente para cada clase hija (Lactante, Infantil y Escolar).
- Todos los lactantes, cuando se llama a comer indican "Tengo lactancia directa" o "Tengo lactancia mixta", según corresponda.
- Infantiles y escolares tienen un método *estudioEn()* que responde con "Estoy en la guardería *nombreGuardería*" o "Estudio en el colegio *nombreColegio*", según corresponda.
- Los infantiles, al llamar al método comer() si son alérgicos deben indicar "Cuidado que soy alérgic@". Si no lo son, el mensaje será el de por defecto: "Estoy comiendo".

- Los escolares tienen un método denominado deportes() que devuelve el mensaje: "Practico *deporte1*, *deporte2* y *deporte3*".

Implementa un objeto de cada tipo, en un programa principal, y prueba todos los casos posibles.

Ejercicio 2. Implementa una clase denominada Hardware que nos va a permitir modelar un determinado componente o producto hardware.

➔ Todo Hardware tiene las siguientes propiedades:

- Precio
- Descripción
- Valoración (lista de enteros entre 1 y 5)
- Opiniones (lista de cadenas de texto)

➔ Debemos tener dos constructores: el primero con todas las propiedades y otro con el precio y la descripción.

➔ Las funcionalidades que tiene todo Hardware, además de la manipulación de sus propiedades precio y descripción (*getters* y *setters*) , son las siguientes:

- Agregar una nueva valoración. Si la valoración no está entre 1 y 5 se debe lanzar una Excepción (busca en Internet o pregunta al profesor cómo hacerlo).
- Obtener la media de todas las valoraciones.
- Agregar una nueva opinión.
- Consultar todas las opiniones.
- Modificar una opinión a través de su índice. Devolverá un booleano indicando si se ha podido modificar o no.
- Obtener características. Dicho método informará de las características particulares (devolverá una cadena de texto) de cada tipo de Hardware que herede de este clase. Por ejemplo, de un Monitor, informará sobre sus pulgadas y de un Disco Duro sobre su capacidad en TB. Por defecto, un Hardware no devuelve ningún texto.

➔ Posteriormente, debes implementar las dos siguientes clases hijas:

- Monitor(precio, descripcion, valoracion, opiniones, pulgadas)
- DiscoDuro(precio, descripcion, valoracion, opiniones, capacidad)

➔ Todas las clases anteriores deben tener sobrescrito el método toString() teniendo en cuenta lo siguiente:

- Todo Hardware mostrará su descripción, precio y la media de sus valoraciones.
- Todo Monitor mostrará la información del Hardware además de sus características (pulgadas).
- Todo DiscoDuro mostrará la información del Hardware además de sus características (capacidad).

Ejercicio 3. Implementa un programa que genere dos Monitores (uno con un constructor y otro con el otro constructor) y dos Discos Duros (cada uno con un constructor). Posteriormente, crea una lista de Hardware y muestra por pantalla la información (toString()) de los cuatro junto con el número de opiniones que tienen. Se indica un ejemplo de salida por pantalla.

```
Información sobre nuestros productos
-------------------------------------
| Monitor de oficina | 99,90 euros | 4,5 de valoración | 17'' -> 2 opiniones
| Monitor gaming | 300,00 euros | 0,0 de valoración | 27'' -> 0 opiniones
| Disco HDD | 40,00 euros | 2,5 de valoración | 64 TB -> 2 opiniones
| Disco SSD | 200,00 euros | 0,0 de valoración | 16 TB -> 0 opiniones
```

Agrega, posteriormente, una opinión con su correspondiente valoración al producto que quieras y vuelve a mostrar la información de los productos.

Ejercicio 4. Modifica la clase Hardware para que la Opinion sea una clase con fecha, opinión (texto) y usuario (texto). Hardware únicamente debe tener un constructor con el precio y la descripción. El método toString() deberá mostrar todas la información de cada Opinion. La fecha y el usuario no se deben poder modificar. Además, la fecha debe establecerse al día de hoy (cuando se crea el objeto Opinion), es decir, no es un parámetro que deba llegar al constructor. Posteriormente, implementa un programa que genere una lista de Monitores con un único Monitor, haga una Opinion y posteriormente modifique dicha Opinion.

Ejercicio 5. Implementa una clase LineaPedido que tenga dos atributos: *producto* (de tipo Hardware) y *unidadesCompradas* (de tipo entero y mayor de cero). Debe tener, además de los getters y setters correspondientes, dos métodos:

- Un método, llamado `subtotal`, que devuelva el resultado de multiplicar las *unidadesCompradas* por el precio del *producto*.
- Un método, llamado `imprimir`, que devuelva una cadena de texto con la información de la línea de pedido (por orden: unidades, descripción, precio unitario y subtotal). Se indica un ejemplo:

```
3       Disco SSD                   80,50   241,50
```

Ejercicio 6. Implementa una clase Pedido que tenga una colección de líneas de pedido (LineaPedido) y una fecha y hora de pedido (que se debe setear en el constructor al momento actual al crear un nuevo Pedido). Ten en cuenta las siguientes características:

- Importa el orden en el que vamos introduciendo los Pedidos.
- Únicamente debe existir un constructor sin parámetros.
- No debe existir ningún método *setter*.

La funcionalidad que debe tener es la siguiente:

- Calcular el total del Pedido mediante un método llamado `total`.
- Añadir una LineaPedido al Pedido indicando el Hardware comprado y las unidades. Si ya existe dicho Hardware, se deben incrementar las unidades compradas. El método responde a la firma `agregarLinea(Hardware, int)`
- Método `imprimir` que devuelva una cadena de texto con la información del Pedido junto con el total a facturar. Se indica un ejemplo de salida por pantalla.

```
1       Monitor gaming          125,50  125,50
3       Disco SSD               80,50   241,50

                TOTAL: 367.0 euros
```

Ejercicio 7. Nos han contratado de una tienda de informática y tenemos que implementar un programa funcional para gestionar Pedidos de Hardware. Para ello, tenemos que crear las clases que se indican a continuación.

Antes de ponernos a implementar, vamos a realizar una toma de contacto con un concepto que usaremos en el futuro: las **vistas**. Las vistas nos van a permitir interactuar con el usuario (ya sea para mostrarle información/datos o para pedírselos), es decir, son la interfaz de comunicación con el usuario. Al igual que tenemos el paquete *app* o el paquete *modelo*, vamos a tener también este nuevo paquete *vista* en el proyecto. Pídele al profesor que te hable sobre MVC.

- **`AppTienda`**: debe permitir crear productos de `Hardware` y crear `Pedido`s. Para ello, haremos uso de esta clase y las siguientes. Para lograr la funcionalidad, hay que mostrar un menú que le pida al usuario qué desea hacer (usaremos la clase `MenuView` para ello). Para facilitar el uso del programa, la lista de productos puede contener ya varios productos creados por código (añadiéndolos directamente a la lista al inicializarla). Como atributos, debemos tener las dos siguientes listas:
    - Lista para almacenar los productos creados.
    - Lista para almacenar los pedidos creados.

```
TIENDA
*******
1. Crear un nuevo producto
2. Crear un nuevo pedido
3. Mostrar todos los productos disponibles
4. Mostrar todos los pedidos actuales
0. Salir
```

- **`MenuView`**: debe estar en un paquete llamado *vista* y contenener el siguiente método:
    - `solicitarAccionDeMenu`. Este método debe mostrar el menú y devolver qué desea realizar el usuario de las 5 opciones ofrecidas (por ejemplo, el número). Dicho número deberá ser procesado por la clase *AppTienda* y apoyarse en el resto de clases para cumplir la funcionalidad de la aplicación.

- **`ProductoCrearView`**: debe estar en un paquete llamado *vista*. Las vistas nos van a permitir interactuar con el usuario (ya sea para mostrarle datos o para pedírselos), es decir, son la interfaz de comunicación con el usuario. Al igual que tenemos el paquete *app* o el paquete *modelo*, vamos a tener también este nuevo paquete *vista* en el proyecto. Esta clase debe implementar el siguiente método público:
    - `crearProducto`. Tiene que permitir crear productos de `Hardware` (`Monitor` o `DiscoDuro`) pidiéndole al usuario, por consola, los datos necesarios. Será usado por la clase *AppTienda* para dicha funcionalidad. Cada vez que se cree un producto, se debe incorporar a la lista de productos de *AppTienda*.

- **`ProductoMostrarView`**: también debe estar en el paquete *vista*. Debe contener el siguiente método público:
    - `mostrarProductos`. Recibirá, por parámetro, una lista de `Hardware` y mostrará la información (*toString()*) de cada producto `Hardware` de la lista.

- **`PedidoCrearView`**: debe estar en el paquete *vista*. Debe implementar el siguiente método público:
    - `crearPedido`. Este método permitirá crear un `Pedido` generando líneas de pedido (`LineaPedido`) hasta que el usuario decida finalizar el `Pedido`. Por cada línea de `Pedido` se le debe pedir al usuario que indique qué producto (`Hardware`) desea comprar y cuántas unidades del mismo. Cada vez que se cree un pedido, se debe incorporar a la lista de pedidos de *AppTienda*.

- **`PedidoMostrarView`**: debe estar en el paquete *vista* y contener el siguiente método público:
    - `mostrarPedidos`. Recibirá, por parámetro, una lista de `Pedido` y mostrará la información (*imprimir()*) de cada `Pedido` de la lista. Si no existen pedidos, se debe informar al usuario de ello.