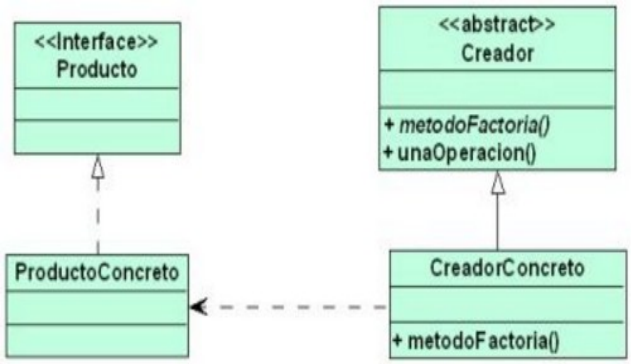
Trabajo Práctico XI: Patrones parte 1

Concepto de Patrón de diseño (Design Pattern).

Ejemplo de Patrones:

- El Patrón Factory Method
- El Patrón Singleton
- El patrón Observer-Observable. Interfaz Observer. Clase Observable.
- El Patrón Decorator

Ejercicio 1 (factory method)



Podemos utilizar este patrón cuando definamos una clase a partir de la que se crearán objetos pero sin saber de qué tipo son, siendo otras subclases las encargadas de decidirlo.

Se tienen dos tipos de archivos, de audio y de video, con el único comportamiento que es “reproducir”.

Aplicar el patrón de forma de lograr lo siguiente:

```
public static void main(String[] args)
{
        IArchivo video = Creador.getArchivo( Creador.VIDEO );
        video.reproducir();
        IArchivo audio = Creador.getArchivo( Creador.AUDIO );
        audio.reproducir();
}
```

Nota: Creador.VIDEO y Creador.AUDIO son dos constantes numéricas.

Ejercicio 2 (observer)

Se desea escribir un juego de atención a mascotas virtuales (aunque parezca mentira, en la decada del 90 fueron muy populares los Tamagotchi).

Las mascotas implementan la interfaz Runnable y tienen un atributo nombre de tipo String. Dentro de un período de tiempo al azar, entre 3 y 10 segundos, requerirá algún tipo de atención (también al azar).

La mascota podrá informar que tiene sed, tiene hambre, o esta aburrida (quiere jugar).

Utilizando el patron Observer escribe una aplicación que implemente 4 mascotas. Otra clase de

tipo Oberver será la encargada de mostrar por consola las peticiones de cada mascota.

Agrega además otro tipo de obervador, que realize el informe en un panel dentro de una ventana (Ambos observadores deben trabajar en forma simultanea).

Nota: Esta aplicación no “atiende” a la mascota, solo informa sus necesidades.

Ejercicio 3 (Decorator)

Se desean escribir las clases para modelar un juego de cartas de roll, donde habrá diferentes personajes.

Los personajes pueden lanzar ataques cuerpo a cuerpo o ataques a distancia, además los personajes tendrán un nivel de armadura.

La clase abstracta Personaje tendrá los métodos getAtaqueDistante y getAtaqueCorto, que devolverán un número double indicando la potencia del ataque. Además tendrá el método getArmadura que devuelve un double indicando el nivel de armadura. Cada tipo de personaje tendrá estos valores base:

	getArmadura	getAtaqueCorto	getAtaqueDistante
Mago	5	50	70
Elfo	10	20	100
Hechicera	10	70	50
Dragon	100	500	200
Guerrero	15	100	100

A su vez, cada uno de estos personajes, podrá ser de alguno de los elementos básicos, esto provocara cambios en los valores de base antes mencionados:

	getArmadura	getAtaqueCorto	getAtaqueDistante
Tierra	+25%	-20%	-30%
Aire	-10%	+20%	+10
Agua	-15%	+20%	Valor base
Fuego	-50%	+80%	+70%

Así por ejemplo, un dragón de fuego tendrá un ataque corto con un valor de 900, una hechicera de aire tendrá una armadura de 9, etc.

Lo clase Fuego, agrega el método Incendiar() y la clase Aire agrega el método invocarHuracan (Estos métodos solo muestran un mensaje por consola)

Escribe las clases correspondientes utilizando el patron Decorator.

Aplica el patrón Factory para crear los diferentes tipos de personajes.

Los personajes estarán en un mapa, por lo tanto el Mapa tendrá una colección de Personajes.

Se desea que un personaje pueda encontrar al personaje del mapa mas cercano, por lo tanto los Personajes deberán poder acceder a la referencia del mapa al cual pertencen. Utiliza el patron Singleton para evitar la doble referencia entre Mapa y Personaje.

Escribe tambien una clase Prueba con un método main donde instanciar un mapa con algunos personajes de las 20 clases posibles y verificar el funcionamiento de los métodos.