Trabajo Práctico V: Interfaces

- Interfaces.
- Extendiendo interfaces.
- Implementando interfaces.
- Variables de Tipo Interfaz.
- Polimorfismo en variables de Tipo Interfaz.
- Utilización del procedimiento de composición (clase que implementa una interfaz y está compuesta por un objeto que también implementa dicha interfaz).
- Interfaces propias del lenguaje de programación.
- Diferencias entre usar clases abstractas e interfaces.
- La Interfaz como herramienta de diseño.
- Herencia, composición e interfaces.
- La clase Object: el método clone().
- Concepto de doble envío. (Double Dispatch).
- Patrón Decorator

Objetivos de la práctica

- Utilizar las Interfaces como herramientas de diseño.
- Escribir Interfaces propias
- Escribir clases que implenten dichas interfaces
- Utilizar variables de tipo Interfaz
- Utilizar interfaces propias del lenguaje Java.
- Comprender las ventajas del principio de segregación de Interfaces.
- Aplicar el procedimiento de composición en una clase que implementa una interfaz y está compuesta por un objeto que también implementa dicha interfaz.
- Realizar correctamente la implementación del método clone().
- Aplicar Double Dispatch

Ejercicio 1:

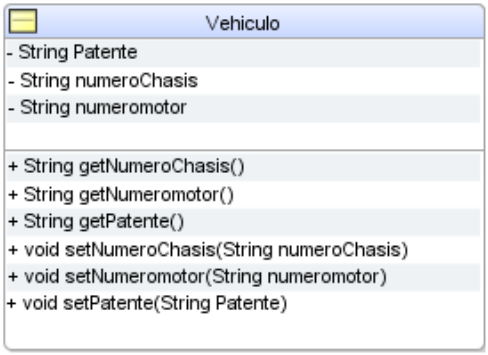
Inciso a)

Escribe la Interfaz Emisor_de_Sonido que contenga el método emiteSonido(); y las clases Perro, Gato, Vaca, Pollito (extendidas de la clase Animal), que implementarán la interfaz Emisor_de_Sonido. En cada caso, el animal imprimirá en consola el sonido correspondiente: El perro “guau”, el gato “miau”, etc.

Escribe también una clase Prueba con método main, donde se creará un ArrayList de emisores de sonido el cual contenga una instancia de cada clase para que emita un sonido.

(La clase Animal posee los atributos String nombre, y int esperanzaDeVida, además de sus correspondientes getters y setters).

Inciso b)



Escribe la clase Ambulancia que extiende la clase Vehículo (La clase Vehiculo tiene los atributos patente y modelo, y sus correspondientes getters y setters) y que implementa la interfaz Emisor_de_Sonido (imprimirá en pantalla el mensaje “sirena sonando”)

Nota que las clases Gato, Perro, y Ambulancia, son Emisores de Sonido, a pesar de pertenecer a líneas de herencia completamente diferentes.

Ejercicio 2

Inciso a)

Escribe un programa para una biblioteca que contenga libros y revistas. Las características comunes que tienen los libros y las revistas son el código, el título y el año de publicación, estas tres características se pasan como parámetro en el momento de la creación.

Los libros tienen además un atributo para saber si están prestados. Al momento de crearse los libros no están prestados.

Las revistas tienen un número, al momento de ser creadas, el número se pasa como parámetro.

Tanto las revistas como los libros, deberán tener (además de los constructores) un método toString() que devuelve todos los atributos en una cadena de texto. También tendrán métodos para devolver cada uno de los atributos.

Se desea contruir la interfaz Prestable, con los métodos prestar(), devolver() y isPrestado(). La clase libro implementará dicha interfaz.

Inciso b)

La biblioteca se ha ampliado y ahora tiene la posibilidad de prestar CD de audio.

Los CD tendrán los atributos código, titulo, interprete, lista de canciones.

A su vez, cada canción tiene un número de pista, un título y una duración en minutos y segundos.

Los discos son Prestables.

Escribe las clases correspondientes y crea el diagrama de clases.

Inciso c)

Modifica el programa anterior de manera que los discos sean además Comparables, su criterio de comparación es alfabéticamente, por nombre de interprete, dos discos del mismo interprete se ordenan de acuerdo al título del álbum.

Ejercicio 3

Para un juego bélico escribe la clase **Unidad,** que tendrá los atributos:
String equipo, int costo, int energia. Deberá tener un constructor que inicialice los tres atributos.
Deberá tener también el método abstracto void recibeDano(cantidad).
Las unidades pueden ser de tipo **Edificio** o de tipo **Personaje**.
Para modelar este juego se decidió utilizar las siguientes interfaces:
interzaf **IPosicionable**, que agrega los métodos int getX(); int getY();
interfaz **IConstruible**, que agrega el método int getTiempoConstruccion();
interfaz **IHostil,** que agregar un método void atacar(Unidad adversario);
interfaz **IMovible**, que agrega el metodo mover(int x, int y);

Considera los siguientes tipos de edificios:

Tipo	**Costo**	**Energia Inicial**	**Tiempo de Construcción**	**Daño recibido**	**Daño producido**
Cuartel	500	3000	60 segundos	50% del parámetro	No puede atacar
TorretaVigilancia	200	2000	40 segundos	100% del parámetro	10 unidades

Considera los siguientes tipos de personajes

Tipo	**Costo**	**Energia Inicial**	**Daño recibido**	**Daño que produce**
Soldado	100	500	100% del parámetro	50 unidades
Medico	40	100	150% del parámetro	No puede atacar

Ejercicio 4 (Composición y delegación)

Tomando como base el ejercicio del juego de estrategia de la guía 4, donde existen diferentes tipos de Personajes, realizar la siguiente modificación.

Cada personaje tiene un atributo de tipo **Posicion** en el escenario.

La clase **Posición** implementa la intarface **Movible**, que nos provee los métodos

- double getPosx(),
- double getPosy(),
- void setXY(double , double),
- void incrementaPos(double valorX, double valorY)
- double distancia(Movible)

La clase **Personaje**, además de tener un atributo posición, implementa la interfáz Movible.

Utilizando interfaces, composición y delegación, escribe las clases e interfaces necesarias para resolver el problema. Realiza un diagrama de clases.

Ejercicio 5 (Double Dispatch)

Escribe las clases necesarias para resolver la siguiente situación:

La clase abstracta material tiene el atributo color de tipo String. Tenemos dos tipos de materiales:

- Madera, que tiene el atributo String tipo; (pino, caoba, cedro etc.)
- Metal, que tiene el atributo String nombre; (cobre, hierro, etc.)

La clase abstracta artesano tiene el atributo nombre, y el método String trabajar(Material mat). Tenemos dos tipos de artesanos: jugueteros y joyeros.

Si el artesano es un juguetero, al trabajar con madera devolverá un String indicando que fabricó un muñequito, si en cambio trabajó con metal indicará que fabricó un autito.

Si el artesano es un joyero, al trabajar con madera devolverá un String indicando que fabricó un par de aros, si en cambio trabajó con metal indicará que fabricó un anillo.

Inciso a)

Soluciona este problema aplicando double dispatch, y escribe la clase prueba en la cual se instanciaran los siguientes objetos y se mostraran los siguientes mensajes:

```
Material cobre=new Metal("cobre","rojo");
Material pino=new Madera("pino","amarillo");
Artesano a1=new Juguetero("Juan");
Artesano a2=new Joyero("Pedro");
System.out.println(a1.trabajar(cobre));
System.out.println(a1.trabajar(pino));
System.out.println(a2.trabajar(cobre));
System.out.println(a2.trabajar(pino));
```

NOTA: Deberás escribir los constructores necesarios, presta atención al orden de los parámetros.

Inciso b)

Realiza las modificaciones necesarias para informar además los detalles propios del artesano y del material usado, por ejemplo:

“Juan fabrico un anillo de oro blanco”

“Pedro fabrico un muñequito de cedro rojo”

NOTA: Presta especial atención a la sobreescritura de los métodos toString, recordando que las clases Metal y Madera poseen atributos diferentes.

Ejercicio 6

Tomando el ejercicio 4 del juego de estrategia, resuelve la siguiente situación:

Inciso a)

Dentro del mapa en el que interactúan los personajes, éstos podrán abrir cofres mágicos, sin embargo, hasta el momento de abrir el cofre no se sabe si el hechizo que contiene es una maldición o una bendición.

De acuerdo al tipo de hechizo y al tipo del Personaje, el resultado de abrir el cofre será el siguiente:

	Guerrero	**Arquero**	**Caballero**
Bendición	Armadura + 200 puntos	Flechas + 5	Vitalidad + 25%
Maldición	Armadura – 200 puntos	Flechas =0	Vitalidad – 50%

Inciso b)

Escribe un nuevo tipo de Personaje, el Dragon (Vitalidad inicial 1000), que puede atacar a enemigos que se encuentren a una distancia menor a 50 restándole tanta vitalidad como lo indique su atributo poder de fuego (inicialmente 100). Por ser un Personaje acorazado, al recibir daño se reduce su vitalidad el 50% de lo indicado.

Al recibir una Maldición, su poder de fuego cae un 30% y su vitalidad se reduce un 20%

Al recibir una Bendición, su poder de fuego aumenta un 40% y su vitalidad aumenta 250 unidades.

Ejercicio 7 (Patrón Decorator)

Se desean escribir las clases para modelar un juego de roll de cartas, donde habrá diferentes personajes que comparan sus características. Gana el personaje que tenga mayor nivel en la habilidad elegida.

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	500	50	70
Elfo	1000	20	100
Hechicera	1000	70	50
Dragon	10000	500	200
Guerrero	1500	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 patrón Decorator.

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

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

Se desea que un personaje pueda encontrar otro personaje del mazo al azar mediante el método `Personaje eligeAdversario()`, por lo tanto los Personajes deberán poder acceder a la referencia del mazo al cual pertenecen. Utiliza el patrón Singleton para evitar la doble referencia entre Mazo y Personaje.

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