GUÍA DE EJERCICIOS

Nº 8 y 9

ING. LUISINA DE PAULA

DESARROLLO WEB FULLSTACK

CON JAVA

Guías de Ejercicios Nº 8 y 9: Programación Orientada a Objetos - Parte 2

1) Repaso Objetos

- Crear una clase llamada VideoJuego, que tenga los siguientes atributos: codigo, titulo, consola, cantidadJugadores, categoría (tener en cuenta todos sus atributos, constructores, métodos getters y setters).

    - ✓ Crear un vector de tipo VideoJuego que pueda almacenar 5 videojuegos. Crear 5 videojuegos y guardarlos en el vector.
    - ✓ Recorrer el vector creado y mostrar por pantalla el titulo, consola y cantidad de jugadores de los videojuegos almacenados.
    - ✓ Cambiar el nombre y la cantidad de jugadores de dos videojuegos. Mostrar por pantalla los datos de todos los videojuegos luego del cambio.
    - ✓ Recorrer el vector y mostrar por pantalla únicamente a los videojuegos que sean de la consola “Nintendo 64”.

- Crear una clase llamada Fruta, que tenga los siguientes atributos: nombre, color, calorías, tipoCascara, esComestible (tener en cuenta todos sus atributos, constructores, métodos getters y setters).
    - ✓ Crear un vector de tipo Fruta que pueda almacenar 5 frutas. Crear 5 frutas y guardarlas en el vector.
    - ✓ Recorrer el vector creado y mostrar por pantalla el nombre y las calorías de las frutas almacenadas.
    - ✓ Cambiar todos los datos de dos frutas. Mostrar por pantalla los datos de todas las frutas luego del cambio.
    - ✓ Recorrer el vector y mostrar por pantalla únicamente a las frutas que sean de color verde.

2) Herencia, Polimorfismo y encapsulamiento

Ejercicio Nº 1

- Crear una clase **Planta** con los atributos: nombre, alto del tallo, tieneHojas, clima ideal. (con sus métodos y constructores correspondientes)
- Crear sus clases hijas que compartan sus atributos y métodos:
    - **Árbol:** variedad, tipo de tronco, radio de tronco, color, tipo de hojas.
    - **Flor:** color de pétalos, cantidad promedio de pétalos, color del pistilo, color de los pétalos, variedad de flor, estación que florece
    - **Arbusto**: Ancho arbusto, esDomestico, variedad arbusto, color de hojas, sePodaONo
- Una vez creada las clases crear los siguientes métodos:
    - **Árbol:** método para mostrar un mensaje en pantalla que diga “Hola soy un árbol”
    - **Flor:** método para mostrar un mensaje en pantalla que diga “Hola soy una flor”
    - **Arbusto:** método para mostrar un mensaje en pantalla que diga “Hola soy un arbusto”
- Crear en el Main un objeto de cada clase hija. Llamar a los 3 métodos desde cada objeto.
- Cambiar el modificador de acceso de los métodos de public a private. Intentar acceder desde el main a estos métodos.

Ejercicio Nº 2

- Crear una clase Vehiculo con los atributos: patente, num de chasis, motor, color, marca, modelo cantidad de asientos (con sus métodos y constructores correspondientes).
- Crear sus clases hijas que compartan sus atributos y métodos:
    - Auto: materialasientos, cantidad_caballos
    - Moto: cilindrada, material_manubrio
    - Colectivo: aptoDiscapacitados, poseeLectorSube, tipoColectivo
    - Camion: tieneAcoplado, cantidadEjes
- Una vez creada las clases, crear en el Main un vector de tipo Vehículo y almacenar 3 autos, 3 motos, 2 colectivos y 1 camión (crear un objeto para cada uno de ellos).
- Crear los siguientes métodos (en cada subclase correspondiente):
    - Auto: método para mostrar un mensaje en pantalla que diga “Hola soy un auto y mi marca es: ” (mostrar marca)

- Moto: método para mostrar un mensaje en pantalla que diga “Hola soy una moto y mi cilindrada es de: ” (mostrar cilindrada)
- Colectivo: método para mostrar un mensaje en pantalla que diga “Hola soy un colectivo y mi cantidad de asientos es de: ” (mostrar cantidad de asientos).
- Camion: método para mostrar un mensaje en pantalla que diga “Hola soy un camión y mi cantidad de Ejes es de: ” (mostrar cantidad de ejes).

- Recorrer el vector y ejecutar el método que corresponde en cada posición del mismo. PISTA: *Todos tendrán el mismo método, con el mismo nombre pero únicamente cambia el mensaje que muestran…* ¿Cómo se puede hacer para que todos posean el mismo método pero solo cambien sus datos que muestran? IMPLEMENTAR HERENCIA y sobreescritura de métodos.

3) Clases abstractas

Ejercicio Nº 1

- Crear una clase abstracta llamada consola, la cual tenga los atributos: codigo_consola, nombre, empresaDesarrollo y año de lanzamiento. Al mismo tiempo, crear un método abstracto “cargarJuego” que indique un mensaje por pantalla que diga “Cargando juego. Por favor espere”.
  A partir de la clase abstracta creada crear las siguientes subclases hijas:

  - ✓ **Nintendo64:** La cual tiene un atributo propio norma y otro para determinar si lee cartuchos piratas. Al mismo tiempo, implementa un método propio “leerCartucho” el cual recibirá el nombre de un juego como parámetro e indicará un mensaje por pantalla indicando esta situación. Por ejemplo: “Leyendo cartucho Banjo Kazooie”.
  - ✓ **PlayStation2:** La cual tiene los atributos propios norma, chipeadaONo y tamañoMemoryCard. Al mismo tiempo, implementa un método propio “leerDvd” el cual recibirá el nombre de un juego como parámetro e indicará un mensaje por pantalla indicando esta situación. Por ejemplo: “Leyendo DVD Fifa 2009”. Por otro lado, también implementa un método propio llamado “grabarEnMemory” el cual debe informar al usuario que se ha guardado correctamente un juego en la memory card.
  - ✓ **Xbox One:** La cual tiene una serie de atributos propios para manejar: Si está conectada a internet o no, si la calidad 4K está activada y si se permite la descarga automática de actualizaciones. Al mismo tiempo implementa un método propio “leerJuegoDigital” el cual recibirá el nombre de un juego como parámetro e indicará un mensaje por pantalla indicando esta situación. Por ejemplo: “Leyendo GTA V desde tienda”.

Una vez desarrolladas cada una de las clases, se solicita desde la clase Main, crear una

instancia de cada una de ellas y llamar a sus correspondientes métodos. *Tener en cuenta que todas las consolas, al heredar de una clase abstracta, deben implementar el/los método/s de su clase madre utilizando sobreescritura de métodos*.

4) Clases Abstractas + Interfaces

Ejercicio Nº 1

Un fanático de Pókemon desea implementar para el modelado de un videojuego los diferentes ataques de cada una de estas criaturas. Para ello, cuenta con una clase abstracta llamada Pokemon, la cual posee los atributos: num_pokedex, nombrePokemon, pesoPokemon, sexo, temporadaQueAparece y tipo, e implementa métodos para los ataques comunes que suele tener la mayoría, entre ellos se encuentran: atacarPlacaje(), atacarArañazo() y atacarMordisco(). Sin embargo, este fanático también desarrolló una serie de interfaces para contemplar los ataques de Pókemons de cierto tipo:

- ✓ **IElectrico:** con los métodos atacarImpactrueno(), atacarPunioTrueno(), atacarRayo(), atacarRayoCarga().
- ✓ **IPlanta:** con los métodos atacarParalizar(), atacarDrenaje(), atacarHojaAfilada(), atacarLatigoCepa().
- ✓ **IFuego:** con los métodos atacarPunioFuego(), atacarAscuas(), atacarLanzallamas().
- ✓ **IAgua:** con los métodos atacarHidrobomba(), atacarPistolaAgua(), atacarBurbuja(), atacarHidropulso().

A partir de estas interfaces, el Pokefanático desea crear las clases que manejen a los personajes principales del videojuego, los cuales son los pokemons starters de la primera temporada (Charmander, Bulbasaur y Squirtle) y Pikachu; para ello tener en cuenta que: Charmander es de tipo fuego, Bulbasaur es de tipo planta, Squirtle es de tipo agua y Pikachu de tipo eléctrico.

Una vez implementadas la clase abstracta e interfaces, sobrescribir los métodos correspondientes para adaptarlos a cada Pókemon mostrando un mensaje en pantalla que indique qué pókemon es y qué ataque está realizando, por ejemplo: “Soy Charmander y estoy atacando con Ascuas” o “Soy Pikachu y estoy atacando con placaje”. Luego de realizar lo mencionado, crear las instancias necesarias y llamar a cada uno de los métodos de cada Pokemon.