Proyecto batalla de chinpokomons en python

Descripción

En este proyecto diseñamos un videojuego de Chinpokomon, donde diversas criaturas se enfrentan entre si en feroces batallas.

Cada chinpokomon tiene un nombre, un nivel de vida, un tipo de naturaleza y una serie de ataques que sabe infligir contra su rival.

Cuando ponemos un chinpokomon a pelear contra otro, estos se atacan entre si, por turnos, el que lleva la vida de su rival a cero primero, es el vencedor.

Patrones de diseño utilizados en el proyecto

Builder:

Para la creacion de cada chiponkomon optamos por usar este patron de diseño ya que de esta manera nos permite producir distintos tipos y representaciones de los chiponkomon, empleando el mismo código de construcción.

La base de creacion de los chiponkomon, comienza de la clase "builder" que tendra las operaciones comunes a la creacion de cada chiponkomon (el nombre, la vida, ataque, naturaleza y random).

La clase "chiponkomon", es el constructor concreto que implentara ademas de los pasos de construccion, las operaciones comunes que podra realizar los chiponkomon, como ejemplo atacar, curarse o parar cuando derroto a su adversario.

El "builderOfChinpokomon" sera el encargado de definir los pasos de construccion especifica de cada chiponkomon, este trabaja en conjunto con la clase "generadorChinpokomon", que ademas de seguir los pasos de builder, creara los atributos (lista de ataques, lista de naturalezas , vida, nombre, etc) que luego se pasara al constructor, y retorna a las distintas criaturas que puede generar(chiponkomon zapato, chiponkomon carnotron y chiponkomon gallotronix).

Singleton:

El patron de diseño singleton se utilizo en la clase Logger, esta se encarga del manejo de los mensajes y estados, dependiendo de la intencion del mensaje.

Los estados son 4 Debug, Error, Warn e Info.

El Logger tambien nos permite manejarlo como un acceso global que se utilizara desde cualquier parte de nuestro proyecto, dependiendo de el estado que se necesite hacer uso, con una única instancia que podemos ir alterando su nivel de loggeo durante runtime.

Composite:

El objetivo a la hora de implementar este patron de diseño fue la posibilidad de que cada chinpokomon puedan tener más de una naturaleza y manejar cada una de esas naturalezas

como si fueran objetos individuales.

Esto ya que la clase "natural", describe todas las operaciones que son comunes a cada tipo de naturaleza y contenedores.

En este caso naturaleza solo sabe decir su nombre (con la property nombre) y retornar "true" o "false" si tiene ventaja

sobre la naturaleza que se le envia (tieneVentaja(naturaleza)).

A partir de esa clase creamos la clase base naturaleza y sus hijos que serian la naturaleza animal, naturaleza robot y naturaleza cosa.

Por último, de la misma clase creamos el contenedor de naturalezas, que sera instanciado en cada chipokomon y almacenara las distintas naturalezas que posean, todas las solicitudes que se envien a este

los delegara a sus hijos.

Ejemplo el metodo "nombre", se pedira todos los nombres de las naturaleza con el mismo metodo y el contenedor retornara el conjunto de nombre de cada naturaleza en su lista.

Cambios al momento de trasladar el código del proyecto java a python

Declaracion de clases:

En python solo definimos clases, por lo tanto la interfaz Builder se declara como Builder(ABC) por abstract base class, que la misma es padre de BuilderOfChinpokomon.

Lo mismo en el caso de InterfaceNaturaleza(ABC) que es implementada por la clase Naturaleza con la sintaxis Naturaleza(InterfaceNaturaleza).

Notaciones:

Se tuvo que hacer uso de las notaciones @property para los getters de los atributos y @nombreInChinpokomon.setter por ejemplo para las propiedades setters, de la misma manera para cada atributo de las clases.

Cambios en el uso de los patrones de diseño:

En el patron BuilderOfChinpokomon se define el método reset para asignar un chinpokomon al llamar al método resultado.

En el patron Logger definimos un singleton con un dict() (diccionario para el manejo de la instancia) y así asegurar una única instancia.

Y con la notación @singleton hacemos uso del mismo en la clase Logger.

En el patron Composite de CompuestoNaturaleza(Naturaleza) para declarar que es hija de la clase Naturaleza, que esta última que hace uso de la clase InterfaceNaturaleza.

Y todas las clases AnimalNaturaleza(Naturaleza), RobototNaturaleza(Naturaleza) y CosaNaturaleza(Naturaleza) de esta manera acceden a sus nombres y ventajas.