INFORME EJERCICIO 2

Principios de diseño

Principio de sustitucion de Liskov

-El principio de sustitucion de liskov consiste en que las clases derivadas deben ser sustituibles por sus clases base. Este principio lo utilizamos en la clase ProjectItem,Team y Worker las clases Worker y Tean heredan los metodos de ProjectItem pero si cambiar su comportamiento.Por tanto se cumple el principio sustitucion de Liskov porque nos daría el mismo resultado utlizando las funciones de ProjectItem utilizando las mismas funciones en las clases Tean y Worker.

Principio de Inversion de dependencia

-El principio de diseño de inversión de la dependencia consiste en depender de abstracciones y no de concreciones. Este principio lo utlizamos al hacer que la clase Project dependa de la clase abstracta ProjectItem a la hora de representar su compontes.Con esto evitamos que la clase Project tuviese una dependencia con las clases concretas Worker y Team. Ademas utilizamos el interfaz Employee para pasarles las horas trabajadas en un projecto un trabajador espefico con este interfaz podremos poner nuevo clases que reciban horas del projecto.

Principio de Segregacion de Interfaces

-El principio de diseño de segregación de interfaces consiste en utilizar muchos interfaces específicos para cada cliente son mejores que un único interfaz de propósito general. Este principio lo utlizamos al tener un intefaz especifico que tiene la única funcion pasarles las horas trabajas en un projecto un empleado y ademas tener una clase abstracta que solo tiene la funcion de representar la organizacion de un projecto en forma de arbol.

Principio de Responsabilidad Unica

-El principio de responsabilidad única consiste en cada objeto debe tener una responsabilidad única que esté enteramente encapsulada en la clase. Este principio se cumple ya que las 4 clases que nos encotramos (Project, ProjectItem, Worker y Team) estan diseñadas para un unico propósito y no tenemos ninguna clase Dios que lo haga todo.Por ejemplo la clases Woker y Team dividen el trabajo que tendría que hacer ProjectItem a la hora de organizar la gerarquia del proyecto.Y las tres las clases mencionadas anteriormente sirven para dividir más trabajo que tendría que hacer la clase Project a la hora de representar un projecto.

Principio Abierto-Cerrado

-El principio de diseño solid abierto-cerrado consiste en que las entidades software deberían ser abiertas para permitir su extensión, pero cerradas frente a la modificación

Este principio lo cumple ya que al hacer que las clases que representarían la organizacion de un projecto hereden de una clase abstracta facilitaria la creacion de una nueva organizacion del projecto sin necesidad de modificar el codigo.

Entre los tipos de herencia nos encontramos con la de especalizacion, ya que, por ejemplo, la clase Team sobreescribe la clase ProjectItem en el metodo "addProject" para que incluya a todos sus componentes dentro del proyecto. Nos encontramos tambien con una herencia de especificacion en los metodos abstractos de la clase ProjectItem que son implementados por Team y Worker.

PATRON:

Patron composicion: es un patron estructural que se utliza para componer objectos en estructuras de árbol que representan jerasquías todo-parte.

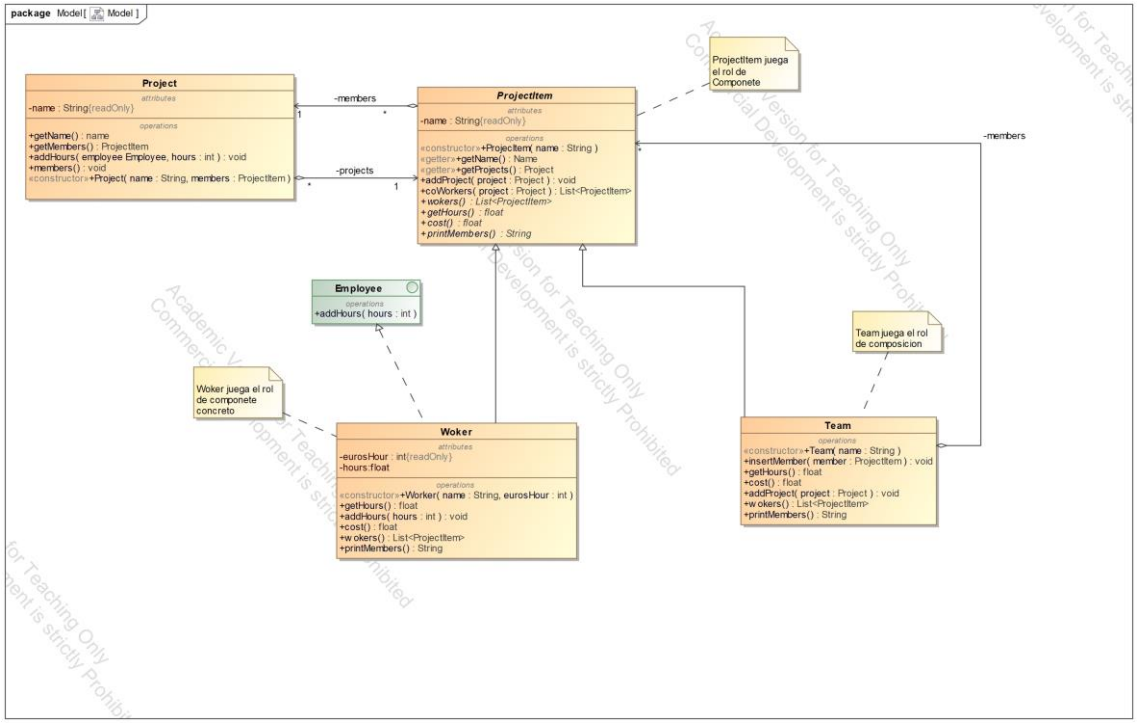
Utilazomos el patrón composición debido a que la el problema sigue una estructura de arbol, donde la raiz es un proyecto y los equipos son nodos de ese arbol que terminan con los trabajadores haciendo el papel de hojas.

Lo que hemos hecho es crear una clase Project como cliente y una clase abstracta ProjectItem como componente, luego hemos creado una clase.

Los roles que juegan nuestras clases en el patron composicion sería la clase ProjectItem juega el rol de Componete ya que declara el comportamiento que tienen que seguir las clases Worker y Tean siendo las fuciones workers(),getHours(), cost(), printMembers() las operaciones que se delagan a las subclases.

La clase Tean juega el rol de composicion porque define un comportamento de los objetos que componen que puede ser un worker ó otro objeto team.la clase Worker juega el rol de Compoente concreto porque implementaria la fucion elemental porque n ohabría otra clase depues de esa.

Diseño clases



Diseño dinamico