Trabalho Prático

Padrões e Desenho de Software

Diogo Moreira NMec: 93127

Padrão Façade

Introdução:

O padrão façade é utilizado para **encapsular** um **subsistema complexo** através do uso de uma **interface** que será usada pelo utilizador, tornando o subsistema **mais fácil de usar**.

Este padrão também tem coupling fraco entre o subsistema complexo e o utilizador. Devido a isto, é possível alterarmos o subsistema sem alterar nada do lado do utilizador.

Apesar de este padrão não adicionar funcionalidades novas torna o sistema muito mais fácil de usar.

Problema:

O jogo **FacadeWonders** é um jogo RPG em que cada jogador cria a sua personagem e luta contra outros. O jogo permite a opção de criar uma **personagem** para o utilizador. Cada personagem tem algo que se chama ***stats*** (HP,STR,SPD,MAG,DEF) que vão ser diferentes dependendo da **classe** da personagem que for escolhida (Warrior,Hunter,Mage,Cleric,…).

No jogo também existe algo chamado de habilidades. Cada personagem tem **habilidades,** predefinidas para a sua classe, que usam MAG para calcular o dano e tem o seu modificador de ataque.

O utilizador também pode **lutar** com outros utilizadores e quem ficar sem HP primeiro perde.

a) Crie o jogo FacadeWonders com a opção de criar uma personagem de diferentes tipos (os **stats** devem ser definidos para cada **classe**).
b) Implemente as **habilidades** sendo que cada classe tem as suas próprias habilidades e cada habilidade tem um modificador de dano que usa magia para calcular o seu dano (exemplo: fireball -> modificador=0.8 e a MAG=16 o dano final vai ser 0.8*16=12.8).
c) Implemente a opção de os utilizadores poderem **lutar entre si** sendo que o combate é **por turnos** e quem tem **SPD** mais alto ataca primeiro.

Solução

Para resolver o problema primeiro foram criados dois enumerados que guardam as diferentes **classes** e **habilidades** que os heróis vão poder escolher. Segundo criamos a classe **Hero** que para cada **classe do herói** escolhido vai atribuir os atributos e habilidades correspondentes. De seguida criamos a classe **Abilities** que serve para dar as habilidades de cada **classe do herói** escolhido. A classe **Combat** só calcula o dano que a habilidade escolhida fará dependendo da habilidade e da defesa do enemigo. Por último temos a classe que implementa o padrão **façade** chamada **FacadeWonders**. Nesta classe o utilizador só precisa de chamar a função **fight (Hero h1,Hero h2)** que iniciará a luta entre dois heróis. A luta entre os dois heróis começa com o herói que tem o maior **SPD**, que é um atributo dos heróis, e através da **escolha** das diferentes habilidades de cada herói ou da opção de desistir (função doChoice) a classe **FacadeWonders** irá calcular o dano que cada habilidade fará. Se o herói chegar a **zero de vida** é declarado que o herói que acabou de atacar **ganhou o combate** (função confirmDeath) ou se isto não acontecer o próximo herói terá agora a **oportunidade** de atacar.

Padrão Visitor

Introdução:

O padrão Visitor é utilizado quando pretendemos criar novas operações para um grupo de classes mas não as queremos alterar. Para criar uma nova operação basta criar um novo visitor que fará essa operação.

Problema:

Uma empresa tem um conjunto de **departamentos** e **cada departamento** tem ID, nome, salário e salário dos seus funcionários.

Agora **dependendo** da **altura do ano** o dinheiro que cada departamento recebe difere mudando também o pagamento que os seus funcionários recebem.

a) Crie os vários departamentos e cada departamento com os salários diferentes (incluindo o salário para os funcionários).

b) Agora use o padrão **Visitor** para implementar a **mudança** no pagamento mensal que cada departamento recebe e paga aos seus funcionários **dependente da altura do ano** (aconselha-se que crie vários visitors para as diferentes altura do ano SpringVisitor, SummerVisitor, FallVisitor, WinterVisitor).

c) Crie mais um **Visitor** mas que quando é chamado avisa que o departamento precisa de manutenção e retira dinheiro ao salario do departamento.

Solução:

Para resolver o problema foi criada uma interface **Visitable** que tem como única função aceitar a “visita” de um **Visitor**. Foram criados vários departamentos que implementam a interface **Visitable** e têm todos salários (salário dos empregados incluído) **diferentes**. De seguida temos a interface **Visitor** que só tem a função **visit** que dependendo do Departamento que “visita” faz algo diferente. Para cada altura do ano criamos um **Visitor** que muda o salário (salário dos empregados incluído) **de cada departamento** para um salário diferente. Por último temos o **ProblemVisitor** que ao ser chamado **decresce** o salário do departamento que o chamou e dá **print** a uma **mensagem** diferente para **cada departamento**.

Referências:

Slides

Youtube

Tutorialspoint

Baeldung

Dzone

Refactoring.guru