Documentação de Projeto: Fábrica de Personagens de RPG

Versão: 1.0

Data: 25 de Setembro de 2025

1. Introdução

Este documento detalha a arquitetura e o design de um pequeno projeto de software cujo objetivo é demonstrar a implementação do padrão de projeto Abstract Factory. O projeto consiste na criação de uma família de elementos de jogos de RPG, como classes de personagens e equipamentos, que podem ser instanciados em diferentes estilos (por exemplo, "Guerreiro" e "Espada") de forma consistente.

1.1. Objetivo do Projeto

O objetivo principal é criar um sistema flexível que permita ao cliente (a aplicação que consome os elementos de jogo) solicitar a criação de novos personagens sem se acoplar às suas classes concretas. Isso permitirá que a aparência da aplicação seja alterada com o mínimo de modificação no código cliente.

2. Padrão de Projeto: Abstract Factory

Nesta seção, descrevemos o padrão de projeto Abstract Factory, que é a base da arquitetura deste projeto.

2.1. Descrição

O Abstract Factory é um padrão de projeto criacional que fornece uma interface para criar famílias de objetos relacionados ou dependentes sem especificar suas classes concretas. Em outras palavras, ele funciona como uma "fábrica de fábricas", onde uma fábrica abstrata define a interface para a criação de um conjunto de produtos, e as fábricas concretas implementam essa interface para criar produtos de uma família específica.

No contexto deste projeto, teremos:

Produtos Abstratos: Interfaces para os elementos do jogo, como Personagem e Arma.

Produtos Concretos: Implementações específicas dos elementos, como Guerreiro, Espada, Assassino e Adaga.

Fábrica Abstrata: Uma interface JogadorFactory que declara os métodos para criar os produtos abstratos (ex: criarPersonagem() e criarArma()).

Fábricas Concretas: Classes que implementam a JogadorFactory para um estilo específico, como GuerreiroFactory e AssassinoFactory.

O código cliente interage apenas com a JogadorFactory e com os produtos abstratos (Personagem, Arma), permanecendo completamente desacoplado das implementações concretas.

2.2. Diagrama de Classes

Snippet de código

```
classDiagram
    class Cliente
    class JogadorFactory {
        <<interface>>
        + criarPersonagem (): Personagem
        + criarArma (): Arma
        + atribuirVida(): int
        + atribuirDano(): int
        + atribuirAcerto(): double
    }
    class GuerreiroFactory {
```

```
    + criarPersonagem (): Personagem
    + criarArma (): Arma
    + atribuirVida(): int
    + atribuirDano(): int
    + atribuirAcerto(): double
}
class AssassinoFactory {
    + criarPersonagem (): Personagem
    + criarArma (): Arma
    + atribuirVida(): int
    + atribuirDano(): int
    + atribuirAcerto(): double
}
class MagoFactory {
    + criarPersonagem (): Personagem
    + criarArma (): Arma
    + atribuirVida(): int
    + atribuirDano(): int
    + atribuirAcerto(): double
}
class Personagem {
    <<interface>>
    + getPersonagem(): String
}
class Arma {
    <<interface>>
```

```
    + getArma(): String
}
class Guerreiro {
    + getPersonagem(): String
}
class Assassino {
    + getPersonagem(): String
}
class Mago {
    + getPersonagem(): String
}
class Espada {
    + getArma(): String
}
class Adaga {
    + getArma(): String
}
class Cajado {
    + getArma(): String
}
```

Client ..> JogadorFactory

GuerreiroFactory --|> JogadorFactory

AssassinoFactory --|> JogadorFactory

MagoFactory --|> JogadorFactory

GuerreiroFactory..> Guerreiro

GuerreiroFactory..> Espada

AssassinoFactory..> Assassino

AssassinoFactory..> Adaga

MagoFactory..> Mago

MagoFactory..> Cajado

Guerreiro --|> Personagem

Assassino --|> Personagem

Mago --|> Personagem

Espada --|> Arma

Adaga --|> Arma

Cajado --|> Arma

3. Vantagens e Desvantagens da Implementação

A escolha do padrão Abstract Factory traz consigo uma série de benefícios e alguns pontos de atenção que devem ser considerados.

3.1. Vantagens

Isolamento das Classes Concretas: O código cliente não depende das classes concretas dos produtos. Ele trabalha apenas com interfaces abstratas, o que facilita a troca de famílias de produtos sem alterar o código que os utiliza.

Consistência entre Produtos: Ao usar uma fábrica concreta, garante-se que todos os produtos criados por ela pertencem à mesma família e são compatíveis entre si. Por exemplo, a GuerreiroFactory sempre criará um Guerreiro e uma Espada, mantendo a consistência.

Facilita a Troca de Famílias de Produtos: A aplicação pode ser configurada para usar uma fábrica concreta diferente em tempo de execução. Isso permite alterar completamente a aparência e o comportamento de uma família de objetos com uma única mudança na inicialização do sistema.

Promove o Princípio Aberto/Fechado: É possível introduzir novas famílias de produtos (novas classes de personagem) sem modificar o código cliente existente. Basta criar uma nova fábrica concreta e as classes de produtos correspondentes.

3.2. Desvantagens

Dificuldade para Adicionar Novos Produtos: A principal desvantagem do padrão é a dificuldade em adicionar novos tipos de produtos à família. Se decidirmos que nossa fábrica de Jogador também deve criar um novo elemento, como um contador de stamina, seria necessário modificar a interface da JogadorFactory e, consequentemente, todas as suas classes de fábricas concretas. Isso viola o Princípio Aberto/Fechado no que diz respeito à adição de novos produtos.

Aumento da Complexidade: Para projetos pequenos e simples, a implementação do Abstract Factory pode introduzir uma grande quantidade de novas interfaces e classes, tornando o código mais complexo e verboso do que o necessário.

Abstração Excessiva: Em cenários onde a flexibilidade de trocar famílias de produtos não é um requisito claro, o uso deste padrão pode levar a uma engenharia excessiva (over-engineering).

4. Estrutura de Arquivos do Projeto

/src

|-- Personagem.java (interface)

|-- Arma.java (interface)

|-- Guerreiro.java

|-- Espada.java

|-- Assassino.java

|-- Adaga.java

|-- Mago.java

|-- Cajado.java

|-- JogadorFactory.java (interface)

|-- GuerreiroFactory.java

|-- AssassinoFactory.java

|-- MagoFactory.java

|-- Jogador.java (Cliente)

|-- Main.java

5. Conclusão

O padrão Abstract Factory é uma solução poderosa para sistemas que precisam criar famílias de objetos relacionados, garantindo consistência e flexibilidade. Para este projeto, ele se mostra ideal para gerenciar diferentes classes de personagens. No entanto, é crucial avaliar a necessidade de tal abstração, especialmente em projetos menores, devido à complexidade adicional que sua implementação pode acarretar.