Universidade Tecnológica Federal do Paraná

Engenharia de Software - Curso de Arquitetura de Software (AS27S)

INSTRUTOR: Prof. Dr. Gustavo Santos

Ibanez Fernandes da Silva Junior, 2033500

CCH - Design Patterns Builder

Problema

Suponha que temos um sistema para criação de personagens de um jogo de RPG. Cada personagem tem atributos como nome, classe, raça, habilidades e equipamentos. A criação de um personagem envolve várias etapas, e queremos uma forma flexível de criar diferentes tipos de personagens.

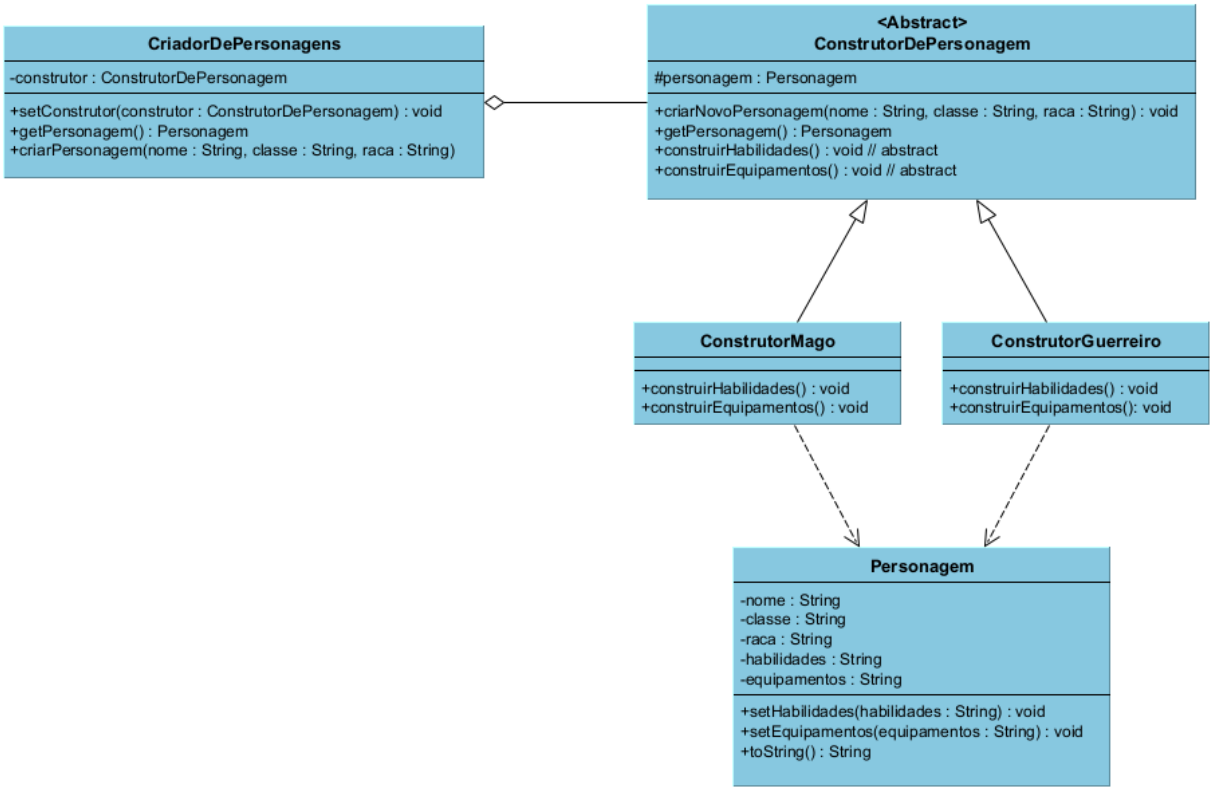
Descrição da Solução

Vamos utilizar o padrão de projeto Builder para criar os personagens de forma flexível e passo a passo.

O padrão de projeto Builder é um padrão de criação que permite construir objetos complexos passo a passo. Ele separa a construção de um objeto complexo da sua representação final, permitindo que o mesmo processo de construção possa criar diferentes representações.

O padrão Builder é útil quando você precisa criar objetos que possuem várias etapas de construção ou quando o processo de construção é complexo e requer a configuração de várias opções. Ele ajuda a evitar a criação de construtores com muitos parâmetros ou métodos setter em uma classe, melhorando a legibilidade e a manutenção do código.

Visão Geral



Exemplo de Código

Java

```java
// Produto: Personagem
public class Personagem {
    private String nome;
    private String classe;
    private String raca;
    private String habilidades;
    private String equipamentos;

    public Personagem(String nome, String classe, String raca) {
        this.nome = nome;
        this.classe = classe;
        this.raca = raca;
    }

    public void setHabilidades(String habilidades) {
        this.habilidades = habilidades;
    }

    public void setEquipamentos(String equipamentos) {
        this.equipamentos = equipamentos;
```

```
    }

    @Override
    public String toString() {
       return "Personagem: " + nome + ", Classe: " + classe + ", Raça: " + raca
       + "\nHabilidades: " + habilidades + "\nEquipamentos: " + equipamentos;
    }
}

// Builder: Construtor de Personagens
public abstract class ConstrutorDePersonagem {
    protected Personagem personagem;

    public Personagem getPersonagem() {
        return personagem;
    }

    public void criarNovoPersonagem(String nome, String classe, String raca) {
        personagem = new Personagem(nome, classe, raca);
    }

    public abstract void construirHabilidades();
    public abstract void construirEquipamentos();
}

// Concrete Builder: Construtor de Guerreiro
public class ConstrutorGuerreiro extends ConstrutorDePersonagem {
    @Override
    public void construirHabilidades() {
        personagem.setHabilidades("Ataque Físico, Defesa");
    }

    @Override
    public void construirEquipamentos() {
        personagem.setEquipamentos("Espada, Armadura Pesada");
    }
}

// Concrete Builder: Construtor de Mago
public class ConstrutorMago extends ConstrutorDePersonagem {
    @Override
    public void construirHabilidades() {
        personagem.setHabilidades("Magia de Fogo, Magia de Gelo");
    }

    @Override
    public void construirEquipamentos() {
        personagem.setEquipamentos("Varinha, Manto Mágico");
    }
}

// Diretor: Criador de Personagens
public class CriadorDePersonagens {
    private ConstrutorDePersonagem construtor;

    public void setConstrutor(ConstrutorDePersonagem construtor) {
        this.construtor = construtor;
```

```
    }

    public Personagem getPersonagem() {
        return construtor.getPersonagem();
    }

    public void criarPersonagem(String nome, String classe, String raca) {
        construtor.criarNovoPersonagem(nome, classe, raca);
        construtor.construirHabilidades();
        construtor.construirEquipamentos();
    }
}

// Exemplo de uso
public class ExemploBuilder {
    public static void main(String[] args) {
        CriadorDePersonagens criador = new CriadorDePersonagens();

        ConstrutorDePersonagem construtorGuerreiro = new ConstrutorGuerreiro();
        criador.setConstrutor(construtorGuerreiro);
        criador.criarPersonagem("Guerreiro1", "Guerreiro", "Humano");
        Personagem guerreiro = criador.getPersonagem();
        System.out.println(guerreiro);

        ConstrutorDePersonagem construtorMago = new ConstrutorMago();
        criador.setConstrutor(construtorMago);
        criador.criarPersonagem("Mago1", "Mago", "Elfo");
        Personagem mago = criador.getPersonagem();
        System.out.println(mago);
    }
}
```

Temos a classe Personagem como o produto final que queremos construir. Ela possui atributos como nome, classe, raça, habilidades e equipamentos, bem como métodos getter e setter para esses atributos.

A classe abstrata ConstrutorDePersonagem atua como o construtor abstrato, definindo os métodos abstratos para construir as diferentes partes do personagem. Cada subclasse concreta, como ConstrutorGuerreiro e ConstrutorMago, implementa esses métodos para construir um tipo específico de personagem.

O CriadorDePersonagens age como o diretor, que recebe um construtor e o utiliza para criar o personagem passo a passo. Ele possui métodos para definir o construtor e para criar o personagem.

No exemplo de uso, primeiro criamos um construtor de guerreiro, configuramos o criador com esse construtor e o utilizamos para criar um guerreiro. Em seguida, repetimos o processo com um construtor de mago. No final, imprimimos os personagens criados.

Este exemplo ilustra como o padrão de projeto Builder permite a criação flexível de objetos complexos, como diferentes tipos de personagens, separando o processo de construção da representação final do objeto. Isso torna a construção flexível e permite a criação de diferentes tipos de objetos usando a mesma estrutura de construção.

Consequências

- Vantagens
    - Reuso de estruturas
    - Construção flexível
    - Simplificação
- Desvantagens
    - A complexidade geral do código aumenta uma vez que o padrão exige criar múltiplas classes novas.

Repositório: ***https://github.com/ibanez-junior-utfpr/as27s-2023-1***

Pasta: CCH-1