Universidade Presbiteriana Mackenzie

Estrutura de Dados II

Prof. Dr. Jean M. Laine

Sobrecarga de Métodos e Polimorfismo

Atividade Prática: Sistema de Batalha Pokémon

Descrição do Problema:

Neste exercício, vocês irão modelar um simples sistema de batalha Pokémon, onde diferentes tipos de Pokémon podem lutar entre si. A ideia é praticar conceitos de orientação a objetos e criar uma hierarquia de classes utilizando conceitos de herança, sobrecarga de métodos, e polimorfismo.

Conceitos Básicos

Sobrecarga	Exemplo
A **sobrecarga de métodos** (ou *method overloading*) em orientação a objetos ocorre quando uma classe tem mais de um método com o mesmo nome, mas com diferentes assinaturas. A assinatura de um método inclui o nome do método, o número de parâmetros, o tipo de parâmetros, e a ordem desses parâmetros. A sobrecarga permite que você crie métodos que realizam funções similares, mas que aceitam diferentes tipos ou quantidades de argumentos. **Como funciona a sobrecarga de métodos?** • **Diferentes tipos de parâmetros:** Você pode ter métodos com o mesmo nome, mas que aceitam diferentes tipos de argumentos. • **Diferentes números de parâmetros:** Os métodos podem ter o mesmo nome, mas um número diferente de parâmetros. • **Diferente ordem de parâmetros:** Quando os parâmetros são de tipos diferentes, a ordem dos parâmetros pode variar.	``` class Calculadora { // Método para somar dois inteiros public int somar(int a, int b) { return a + b; } // Método para somar dois números de ponto flutuante public double somar(double a, double b) { return a + b; } } ```

Polimorfismo	Exemplo
O *polimorfismo* permite que o mesmo método tenha diferentes comportamentos, dependendo do objeto que o chama. **Polimorfismo de Sobrecarga (Polimorfismo Estático):** Como discutido anteriormente, é quando vários métodos têm o mesmo nome, mas diferentes assinaturas (número, tipo ou ordem dos parâmetros). O compilador determina qual método será invocado durante a compilação. **Polimorfismo de Substituição (Polimorfismo Dinâmico):** Este ocorre quando um método em uma classe base é substituído (ou "**overridden**") por um método com o mesmo nome e assinatura em uma classe derivada. A decisão sobre qual método será chamado é feita em tempo de execução, com base no tipo do objeto que está chamando o método.	

```java
// Classe base Animal
class Animal {
    // Método genérico
    public void fazerSom() {
        System.out.println("O animal faz um som.");
    }
}

// Classe derivada Cachorro
class Cachorro extends Animal {
    // Substitui o método fazerSom da classe base
    @Override
    public void fazerSom() {
        System.out.println("O cachorro late.");
    }
}

// Classe derivada Gato
class Gato extends Animal {
    // Substitui o método fazerSom da classe base
    @Override
    public void fazerSom() {
        System.out.println("O gato mia.");
    }
}

// Programa principal
public class TestePolimorfismo {
    public static void main(String[] args) {
        // Criação de objetos
        Animal meuAnimal = new Animal(); // Instância de Animal
        Animal meuCachorro = new Cachorro(); // Instância de Cachorro
        Animal meuGato = new Gato(); // Instância de Gato

        // Chamadas polimórficas
        meuAnimal.fazerSom();   // Saída: O animal faz um som.
        meuCachorro.fazerSom(); // Saída: O cachorro late.
        meuGato.fazerSom();     // Saída: O gato mia.
    }
}
```

Objetivos de Aprendizado:

- Praticar programação orientada a objetos, com a criação e uso de classes e subclasses em Java.
- Compreender e aplicar herança e polimorfismo/sobrecarga de métodos.
- Implementar um sistema simples que simule batalhas entre diferentes tipos de Pokémon.

Especificações:

1. **Superclasse: Pokemon**
    - **Atributos:**
        - nome (String): O nome do Pokémon.
        - nivel (int): O nível do Pokémon.
        - hp (int): A vida do Pokémon.
    - **Métodos:**
        - atacar(): Método abstrato que será implementado pelas subclasses. Este método deve receber como parâmetro um outro Pokémon e definir como o ataque afeta os atributos do adversário.
        - receberDano(int dano): Método que reduz o valor de hp do Pokémon.

- recuperarHp(): Método para curar o Pokémon, recuperando seu hp (pode ser *sobrecarregado* para diferentes níveis de cura).
    - recuperarHp(): método para cura básica que incrementa o hp em algumas unidades;
    - recuperarHp(int quantidade): método para cura que que incrementa o hp de acordo com a quantidade recebida por argumento.
- estaVivo(): Método que verifica se o Pokémon está ou não vivo.

2. **Subclasses de Pokemon:**
    - **PokemonAgua**:
        - Especialidade: Ataques de água.
        - Método atacar() deve causar dano extra, igual a 15 unidades, a Pokémons do tipo PokemonFogo. Caso contrário, causa dano de 10 unidades.
    - **PokemonFogo**:
        - Especialidade: Ataques de fogo.
        - Método atacar() deve causar dano extra, igual a 15 unidades, a Pokémons do tipo PokemonPlanta. Caso contrário, causa dano de 10 unidades.
    - **PokemonPlanta**:
        - Especialidade: Ataques de planta.
        - Método atacar() deve causar dano extra, igual a 15 unidades, a Pokémons do tipo PokemonAgua. Caso contrário, causa dano de 10 unidades.
3. **Main Class:**
    - Crie uma classe com o método Main para iniciar uma batalha entre dois Pokémons de diferentes tipos;
    - Instancie os Pokémons com alguns atributos aleatórios: nome, nível e hp;
    - A batalha deve continuar até que o hp de um dos Pokémons chegue a zero;
    - Utilize polimorfismo para realizar os ataques de acordo com o tipo do Pokémon.
    - Demonstre o uso de sobrecarga ao curar (aleatoriamente) o Pokémon com diferentes quantidades de hp durante a batalha.
    - Após a batalha, exiba o vencedor.