PEDRO HENRIQUE VALENTIN GONTIJO

Linguagens e Técnicas de programação III

Fundamentos da Programação Orientada a Objeto

Introdução a POO

Nos desenvolvimentos de sistemas, existem alguns fatores importantes como: o entendimento do código, fácil manutenção, reaproveitamento entre outros. Para isso,a Programação Orientada a Objeto, também conhecida como POO, tem a intenção de ajudar nesses fatores, dando tempo e agilidade no desenvolvimento de um sistema para o programador.
A Programação Orientada a Objetos foi criada por Alan Kay, autor da linguagem Smalltalk. Ele dizia que o computador ideal deveria funcionar como um organismo vivo, isto é, cada célula se relaciona com outras a fim de alcançar um objetivo, mas cada uma funciona de forma autônoma. As células poderiam também reagrupar-se para resolver outro problema, ou desempenhar outras funções.

Conceitos

1. Classes

Define os atributos e métodos comuns que serão compartilhados por um objeto. A POO utiliza classe para codificar os objetos com características e comportamentos semelhantes. Usamos as classes para construir objetos, o que é chamado de *instanciação*. E os objetos consistem na essência da programação orientada a objetos (ou *OOP*, do inglês *Object-Oriented Programming*). Falando intuitivamente, as classes consistem de uma maneira de organizar um conjunto de dados, e designar todos os métodos necessários para usar ou alterar esses dados.

2. Objetos

Coisa material ou abstrata que pode ser percebida pelos sentidos e descrita por meio das suas características, comportamento e estado atual. São usados para representar entidades ou coisas do mundo real. São utilizados para fazer uma abstração do mundo real para solucionar um determinado problema.

3. Métodos

É um membro da classe que implementa uma ação que pode ser executada por um objeto ou pela classe. Pode ter uma lista de parâmetros, que representa os valores ou referências de variáveis que são passadas para o método é um tipo de retorno, que indica o que é retornado pelo método.

4. Atributos

Os atributos são as propriedades de um objeto, também são conhecidos como variáveis ou campos. Essas propriedades definem o estado de um objeto, fazendo com que esses valores possam sofrer alterações.

5. Abstração

É utilizada para a definição de entidades do mundo real. Sendo onde são criadas as classes. Essas entidades são consideradas tudo que é real, tendo como considerar as suas características e ações.

6. Encapsulamento

É a técnica utilizada para esconder uma ideia, ou seja, não expor detalhes internos para o usuário, tornando partes do sistema mais independentes possível.Em um processo de encapsulamento os atributos das classes são do tipo PRIVATE. Para acessar esses tipos de modificadores, é necessário criar métodos SETTERS e GETTERS. Os métodos setters servem para alterar a informação de uma propriedade de um objeto. E os métodos getters para retornar o valor dessa propriedade.

7. Herança

Na Programação Orientada a Objetos o significado de herança tem o mesmo significado para o mundo real. Assim como um filho pode herdar alguma característica do pai, na Orientação a Objetos é permitido que uma classe herda atributos e métodos da outra, tendo apenas uma restrição para a herança. Os modificadores de acessos das classes, métodos e atributos só

podem estar com visibilidade PUBLIC e PROTECTED para que sejam herdados.

Uma das grandes vantagens de usar o recurso da herança é na reutilização do código. Esse reaproveitamento pode ser acionado quando se identifica que o atributo ou método de uma classe será igual para as outras. Para efetuar uma herança de uma classe é utilizada a palavra reservada chamada EXTENDS.

8. Polimorfismo

Polimorfismo significa "muitas formas", é o termo definido em linguagens orientadas a objeto, como por exemplo que permite ao desenvolvedor usar o mesmo elemento de formas diferentes. Polimorfismo denota uma situação na qual um objeto pode se comportar de maneiras diferentes ao receber uma mensagem. No Polimorfismo temos dois tipos:
Polimorfismo estático ou sobrecarga,polimorfismo dinâmico ou sobreposição.

O Polimorfismo Estático se dá quando temos a mesma operação implementada várias vezes na mesma classe. A escolha de qual operação será chamada depende da assinatura dos métodos sobrecarregados.

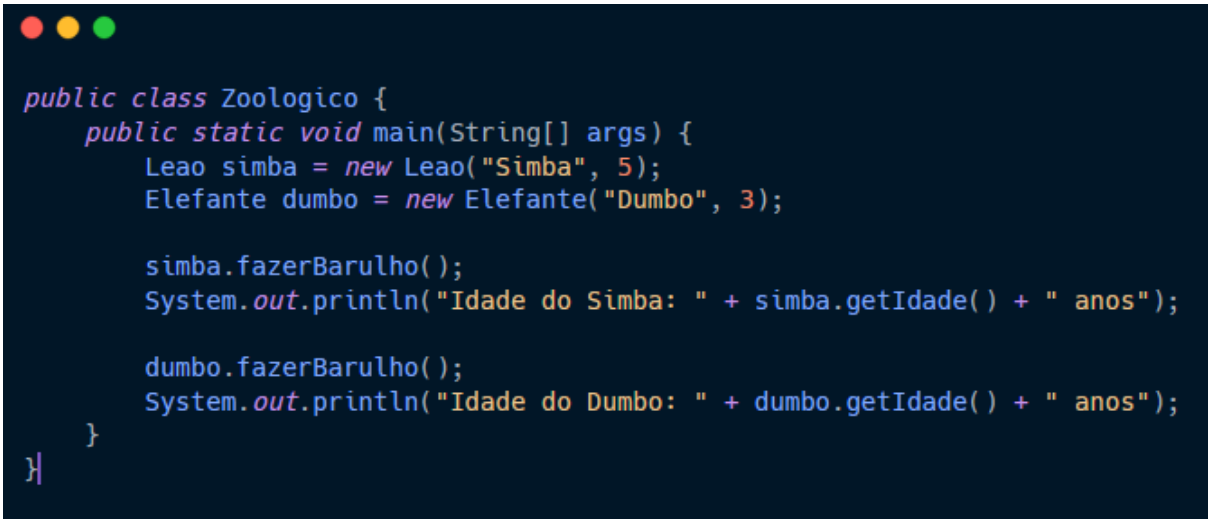
O Polimorfismo Dinâmico acontece na herança, quando a subclasse sobrepõe o método original. Agora o método escolhido se dá em tempo de execução e não mais em tempo de compilação. A escolha de qual método será chamado depende do tipo do objeto que recebe a mensagem.

Conclusão

Utilizar POO pode ser mais confiável, pois o isolamento entre partes pode gerar um software mais seguro, ao alterar uma parte nenhuma outra é afetada.
Ao dividir um projeto em parte, elas podem ser desenvolvidas em paralelo, criando assim uma interação em equipe e um desenvolvimento mais rápido.
A manutenção é mais fácil, sendo assim, uma pequena modificação pode beneficiar todas as partes que utilizarem um objeto.
O software não é instável, ele pode se expandir para se manter atualizado, ele pode ser reutilizado em outros sistemas com poucas alterações e por fim, mais natural de se entender, focando mais na funcionalidade do que nos detalhes da implementação.

```java
class Animal {
    private String nome;
    private int idade;

    public Animal(String nome, int idade) {
        this.nome = nome;
        this.idade = idade;
    }

    public String getNome() {
        return nome;
    }

    public int getIdade() {
        return idade;
    }

    public void fazerBarulho() {
        System.out.println("O esta fazendo barulho");
    }
}
```

```java
class Leao extends Animal {
    public Leao(String nome, int idade) {
        super(nome, idade);
    }

    @Override
    public void fazerBarulho() {
        System.out.println("O leão está rugindo.");
    }
}
```



```java
public class Zoologico {
    public static void main(String[] args) {
        Leao simba = new Leao("Simba", 5);
        Elefante dumbo = new Elefante("Dumbo", 3);

        simba.fazerBarulho();
        System.out.println("Idade do Simba: " + simba.getIdade() + " anos");

        dumbo.fazerBarulho();
        System.out.println("Idade do Dumbo: " + dumbo.getIdade() + " anos");
    }
}
```

- **Classes: As classes : `Animal`; `Leao` e `Elefante` representam diferentes tipos de animais no zoológico.**
- **Objetos: As instâncias `simba` e `dumbo` são objetos da classe `Leao` e `Elefante`, respectivamente.**
- **Métodos: O método `fazerBarulho()` é um método em comum para todos os animais, mas é sobreposto nas subclasses para ter comportamentos específicos.**
- **Atributos: As classes possuem atributos como `nome` e `idade` para representar características dos animais.**
- **Abstração: A classe abstrata `Animal` serve como uma abstração geral para os animais, enquanto as subclasses fornecem detalhes específicos.**
- **Encapsulamento: Os atributos da classe `Animal` são encapsulados com modificadores `private`, e os métodos `getNome()` e `getIdade()` são usados para acessar esses atributos de forma controlada.**
- **Herança: As subclasses `Leao` e `Elefante` herdam da classe base `Animal`, compartilhando atributos e métodos comuns.**
- **Polimorfismo: O polimorfismo é exemplificado quando chamamos o método `fazerBarulho()` em objetos das subclasses. O comportamento é determinado pelo tipo de objeto em tempo de execução.**