Os 4 pilares da Programação Orientada e Objetos

Abstração

Na abstração, existem 3 pontos que devem ser levados em consideração, o primeiro ponto é darmos uma identidade ao objeto! Essa identidade deve ser única para não haver conflito dentro do sistema. O segundo ponto fala sobre as características do objeto, no mundo real,

qualquer objeto tem suas características, então na nossa programação não vai ser diferente. Dentro da programação, essas características são nomeadas propriedades, por exemplo, as propriedades de um objeto "Gato" poderia ser "Tamanho", "Cor" e "Idade". Por fim, o terceiro ponto são as ações que o objeto irá realizar, esses eventos são chamados de métodos; esses métodos podem ser "Ascender()", ou até "Latir".

Analogia: Pense na abstração como dirigir um carro, você só precisa saber como dirigir, sem entender todos os detalhes de como o motor funciona. Importância e Benefícios: Reduz a complexibilidade do software e facilita o entendimento e a manutenção do código.

Importância e Benefícios: Reduz a complexidade, permite a reutilização do código e facilita a manutenção e evolução do mesmo.

Questões comuns e respostas: Qual é a diferença entre abstração e encapsulamento?

R: Encapsulamento é sobre esconder os dados internos e proteger o acesso a eles, enquanto a abstração é sobre esconder a complexidade e mostrar apenas a funcionalidade essencial.

Exemplo prático em Java:

```java
// Classe abstrata
abstract class Forma {
    // Método abstrato (não
implementado)
    abstract void desenhar();
}

// Subclasse concreta
class Circulo extends Forma {
    @Override
    void desenhar() {
        System.out.println("Desenhando
um círculo");
    }
}

// Subclasse concreta
class Retangulo extends Forma {
    @Override
    void desenhar() {
        System.out.println("Desenhando
um retângulo");
    }
}

public class Main {
    public static void main(String[]
args) {
        Forma circulo = new Circulo();
        Forma retangulo = new
Retangulo();

        circulo.desenhar();    //
Output: Desenhando um círculo
        retangulo.desenhar();  //
Output: Desenhando um retângulo
    }
}
```

Encapsulamento

O encapsulamento é a principal técnica que define a programação orientada a objetos. Se trata de um dos elementos que adicionam segurança á aplicação em uma programação orientada, pelo fato de esconder as propriedades, criandp uma espécie de caixa preta. O mais usado é o encapsulamento baseado em propriedades privadas, ligadas a métodos especiais chamados getters e setters, que irão retornar e setar o valor da propriedade, respectivamente. Essa atitude evita o acesso direto aa propriedade do objeto, adicionando uma nova camada de segurança á aplicação. Um exemplo da vida real é a televisão, nós não vemos o que acontece internamente nela, apenas o que ela transmite para nós. **Analogia:** Pense no encapsulamento como uma caixa-forte. Apenas que tem a chave pode acessar o conteúdo dela, e mesmo assim, apenas de forma controlada.

Importância e Benefícios: Protege os dados contra acessos não autorizados e facilita a manutenção e modificação do código sem afetar outras partes do programa.

Questões comuns e respostas: Porque devo usar encapsulamento?

R: Para proteger os dados e garantir que apenas métodos autorizados possam alterá-los.

Exemplo prático em Java:

```java
public class ContaBancaria {
    private double saldo; // atributo
encapsulado

    // Construtor
    public ContaBancaria(double
saldoInicial) {
        this.saldo = saldoInicial;
    }

    // Método para depositar dinheiro
    public void depositar(double
quantia) {
        if (quantia > 0) {
            saldo += quantia;
        }
    }

    // Método para sacar dinheiro
    public void sacar(double quantia) {
        if (quantia > 0 && quantia <=
saldo) {
            saldo -= quantia;
        }
    }

    // Método para obter o saldo
    public double getSaldo() {
        return saldo;
    }
}
```

Herança

A herança otimiza a produção da aplicação em tempo e linhas de código. Para explicar, imagine uma família: a criança, por exemplo, está herdando características dos seus pais, os pais, por sua vez, herdam algo dos avós, o que faz com que a criança também o faça. O objeto abaixo herda características de todos os objetos acima dele, seus “ancestrais”. A criança herda diretamente do pai e indiretamente dos avós.

Analogia: Imagine a herança como herdar as características de seus pais, você recebe atributos e comportamentos que já foram definidos anteriormente.

Importância e Benefícios: Promove a reutilização de código e facilita a manutenção e a extensibilidade do software.

Questões comuns e respostas: Posso herdar mais de uma classe em java?

R: Não, Java não suporta herança múltipla diretamente. No entanto, você pode usar interfaces para simular herança múltipla.

Exemplo prático em Java:

```
// Superclasse
public class Veiculo {
    protected String marca;

    public void acelerar() {
        System.out.println("O veículo
está acelerando...");
    }
}

// Subclasse
public class Carro extends Veiculo {
    private int numeroDePortas;

    public Carro(String marca, int
numeroDePortas) {
        this.marca = marca;
        this.numeroDePortas =
numeroDePortas;
    }

    public void mostrarInformacoes() {
        System.out.println("Marca: " +
marca + ", Número de portas: " +
numeroDePortas);
    }
}
```

Polimorfismo

Na natureza, vemos animais que são capazes de alterar sua forma conforme a necessidade, e é dessa ideia que vem o polimorfismo. Como sabemos, os objetos filhos herdam as características de seus ancestrais. O polimorfismo consiste na alteração do funcionamento interno de um método herdado de um objeto pai. Temos o objeto genérico "Eletrodoméstico". Esse objeto possui um método, ou ação, "Ligar()". Temos dois objetos, "Televisão" e "Geladeira", que não isão ser ligados da mesma forma. Assim, precisamos, para cada uma das classes filhas, reescrever o método "Ligar()".

Analogia: Pense no polimorfismo como um artista que consegue tocar vários instrumentos, embora a técnica básica seja a mesma, o som produzido vária de acordo com o instrumento.

Importância e Benefícios: Aumenta a flexibilidade do código e facilita

a manutenção e a extensão do software.

Questões comuns e respostas: O que é polimorfismo em tempo de execução?

R: É quando o método a ser invocado é determinado em tempo de execução com base no objeto atual, como quando usamos métodos sobrescritos.

Exemplo prático em Java:

```
public class Animal {
    public void fazerSom() {
        System.out.println("O animal faz
um som");
    }
}

public class Cachorro extends Animal {
    @Override
    public void fazerSom() {
        System.out.println("O cachorro
late");
    }
}

public class Gato extends Animal {
    @Override
    public void fazerSom() {
        System.out.println("O gato
mia");
    }
}

public class Main {
    public static void main(String[]
args) {
        Animal meuAnimal = new Animal();
        Animal meuCachorro = new
Cachorro();
        Animal meuGato = new Gato();

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