Bing

Referência: https://www.freecodecamp.org/portuguese/news/principios-de-programacao-orientada-a-objetos-em-java-conceitos-de-poo-para-iniciantes/

Relacionamentos entre classes são formas de representar como as classes se relacionam umas com as outras, ou seja, como elas compartilham dados e comportamentos. Existem dois tipos principais de relacionamentos: É UM e TEM UM.

- Um relacionamento É UM indica que uma classe é uma especialização ou uma implementação de outra classe. Isso significa que ela herda ou assume os atributos e métodos da classe mais geral, e pode adicionar ou modificar alguns deles. Por exemplo, a classe Cachorro é uma especialização da classe Animal, pois ela tem todos os atributos e métodos de um Animal, mas também tem alguns específicos de um Cachorro, como latir ou abanar o rabo. Em Java, usamos a palavra-chave `extends` para indicar um relacionamento É UM entre classes. Veja o código abaixo:

```
// Classe mais geral
public class Animal {
    // Atributos comuns a todos os animais
    private String nome;
    private int idade;
    private double peso;

    // Métodos comuns a todos os animais
    public void comer() {
        System.out.println(nome + " está comendo");
    }

    public void dormir() {
        System.out.println(nome + " está dormindo");
    }
}

// Classe mais específica
public class Cachorro extends Animal { // Cachorro É UM Animal
    // Atributos específicos de um cachorro
    private String raca;
    private boolean castrado;

    // Métodos específicos de um cachorro
    public void latir() {
        System.out.println(nome + " está latindo");
    }

    public void abanarRabo() {
        System.out.println(nome + " está abanando o rabo");
    }
```

```
}
```

- Um relacionamento TEM UM indica que uma classe possui uma referência para uma instância de outra classe, ou seja, ela usa ou contém um objeto de outra classe como parte de seus dados. Por exemplo, a classe Carro possui uma referência para uma instância da classe Motor, pois ela usa um Motor como parte de seu funcionamento. Em Java, usamos a composição de objetos para indicar um relacionamento TEM UM entre classes. Veja o código abaixo:

```
// Classe que representa um motor
public class Motor {
    // Atributos de um motor
    private int potencia;
    private int cilindros;
    private boolean ligado;

    // Métodos de um motor
    public void ligar() {
        ligado = true;
        System.out.println("Motor ligado");
    }

    public void desligar() {
        ligado = false;
        System.out.println("Motor desligado");
    }
}

// Classe que representa um carro
public class Carro {
    // Atributos de um carro
    private String modelo;
    private String cor;
    private int ano;
    private Motor motor; // Carro TEM UM Motor

    // Métodos de um carro
    public void acelerar() {
        System.out.println("Carro acelerando");
        motor.ligar(); // Usa o método do motor
    }

    public void frear() {
        System.out.println("Carro freando");
        motor.desligar(); // Usa o método do motor
    }
}
```

Existem dois tipos de relacionamentos TEM UM: agregação e composição. A diferença entre eles é que na agregação, as classes podem existir independentemente uma da outra, enquanto que na composição, uma classe depende da existência da outra.

- Na agregação, a classe A possui uma referência para uma instância da classe B, mas a classe B pode ser usada em outros contextos ou por outras classes. Por exemplo, a classe Estudante possui uma referência para uma instância da classe ContactInfo, que contém as informações de contato do estudante. Mas a classe ContactInfo pode ser usada em outros lugares, como por exemplo, por uma classe Funcionario de uma empresa. Assim, Estudante pode existir sem ContactInfo e ContactInfo pode existir sem Estudante. Veja o código abaixo:

```
// Classe que representa as informações de contato
class ContactInfo {
    // Atributos das informações de contato
    private String endereco;
    private String email;
    private int telefone;
}

// Classe que representa um estudante
public class Estudante {
    // Atributos de um estudante
    private String nome;
    private int idade;
    private int serie;
    private ContactInfo contactInfo; // Estudante TEM UM ContactInfo

    // Método de um estudante
    public void estudar() {
        System.out.println("Estudar");
    }
}

// Classe que representa um funcionário
public class Funcionario {
    // Atributos de um funcionário
    private String nome;
    private int salario;
    private String cargo;
    private ContactInfo contactInfo; // Funcionario TEM UM
ContactInfo

    // Método de um funcionário
    public void trabalhar() {
        System.out.println("Trabalhar");
    }
}
```

- Na composição, a classe A possui uma referência para uma instância da classe B, mas a classe B não pode existir sem a classe A. Por exemplo, a classe Estudante possui uma referência para uma instância da classe StudentId, que contém as informações de identificação do estudante. Mas a classe StudentId não pode existir sem a classe Estudante, pois ela é específica para esse contexto. Assim, Estudante pode existir sem

StudentId, mas StudentId não pode existir sem Estudante. Veja o código abaixo:

```
// Classe que representa as informações de identificação do
estudante
class StudentId {
    // Atributos das informações de identificação
    private String idNumber; //A-123456789
    private String bloodGroup;
    private String accountNumber;
}

// Classe que representa um estudante
public class Estudante {
    // Atributos de um estudante
    private String nome;
    private int idade;
    private int serie;
    private StudentId studentId; // Estudante TEM UM StudentId

    // Método de um estudante
    public void estudar() {
        System.out.println("Estudar");
    }
}
```

https://www.freecodecamp.org/portuguese/news/principios-de-programacao-orientada-a-objetos-em-java-conceitos-de-poo-para-iniciantes/