INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO TRIÂNGULO MINEIRO

Barbara Ramos Alves
Carlos Eduardo Rangel Lima
Diego Henrique Martins Diniz
Milena Arantes Bertolini
William Alves

Relacionamento de Classes - Programação Orientada a Objetos 1

Uberlândia
2023

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
TRIÂNGULO MINEIRO

Introdução:
Vamos explorar conceitos fundamentais da programação orientada a objetos em Java: o relacionamento entre classes. Entendendo como as classes se relacionam, facilitará a criação de sistemas mais robustos, flexíveis e eficientes.

Conceitos Básicos:
Classes:
Em Java, uma classe é um modelo para criar objetos. Ela define atributos e métodos que representam o comportamento e as características dos objetos.

Objetos:
Os objetos são instâncias de classes. Eles representam entidades do mundo real e são criados a partir de classes.

Relacionamento entre Classes:
Refere-se à maneira como as classes interagem umas com as outras. Existem diferentes formas de relacionamento: herança, composição, agregação, associação, dependência e generalização.

Importância do Relacionamento entre Classes e Benefícios:

- Reusabilidade de Código: Relacionamentos bem projetados facilitam a reutilização de classes em diferentes partes do código.
- Manutenção Simplificada: Alterações em uma classe não afetam diretamente outras classes se o relacionamento for adequado.
- Organização Eficiente: Permite a criação de sistemas mais organizados e modulares.

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
TRIÂNGULO MINEIRO

Herança:
Herança é um dos pilares da programação orientada a objetos. Permite que uma classe herde atributos e métodos de outra, promovendo a reutilização de código. Estabelecendo uma relação "é um".
Relacionamento: Relação do tipo "é um".

Exemplo:
1 - Uma classe "Círculo" pode herdar da classe "Figura"

2 -

```
class Animal {
    // Atributos e métodos comuns a todos os animais
}

class Cachorro extends Animal {
    // Atributos e métodos específicos para um cachorro
}
```

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
TRIÂNGULO MINEIRO

Composição:
Composição é um tipo de relacionamento onde uma classe contém objetos de outras classes como parte de sua estrutura.
Relacionamento: Uma parte faz parte de um todo.
Exemplo: Uma classe "Carro" é composta por objetos das classes "Motor", "Rodas",

```
class Carro {
    Motor motor; // Classe Motor é parte do Carro
}
```

Agregação:
Agregação é uma forma mais fraca de composição, onde uma classe contém objetos de outras classes, mas a vida útil dos objetos não é necessariamente controlada pela classe que os contém.
Relacionamento: Uma parte é associada a um todo, mas pode existir independentemente.

Exemplo:
1- Uma classe "Time" pode ter uma agregação com a classe "Jogador".

2 -

```
class Universidade {
    List<Aluno> alunos; // Classe Aluno está agregada à Universidade
}
```

Associação:
Associação é uma relação mais genérica entre classes, indicando que elas estão conectadas de alguma forma.
Relacionamento: Uma classe está associada a outra de alguma forma.

Exemplo:
1 - Uma classe "Aluno" está associada a uma classe "Professor".

2 -

```
class Pessoa {

}

class Departamento {
    List<Pessoa> membros; // Associação entre Departamento e Pessoa
}
```

Dependência:
Indica que uma classe depende de outra, mas sem uma relação direta de composição, agregação ou associação.
Relacionamento: Uma classe depende de outra para realizar uma ação.
Exemplo: Uma classe "Voo" pode depender de uma classe "Aeroporto" para obter informações.

```java
// Classe Aeroporto
public class Aeroporto {
    public String obterInformacoes() {
        return "Informações do Aeroporto";
    }
}

// Classe Voo depende de Aeroporto
public class Voo {
    private Aeroporto aeroporto;

    // Dependência é estabelecida através do construtor
    public Voo(Aeroporto aeroporto) {
        this.aeroporto = aeroporto;
    }

    public void obterInformacoesAeroporto() {
        String informacoes = aeroporto.obterInformacoes();
        System.out.println("Informações do Voo: " + informacoes);
    }
}
```

Generalização:

Relacionamento onde uma classe mais geral (superclasse) é especializada por uma classe mais específica (subclasse).
Relacionamento: Uma classe mais específica é um tipo especializado de uma classe mais geral.
Exemplo: Uma classe "Animal" pode ser generalizada por classes mais específicas como "Mamífero" e "Réptil".

```java
// Classe Animal (superclasse)
public class Animal {
    public void emitirSom() {
        System.out.println(x:"Som do Animal");
    }
}
```

```java
// Classe Mamífero (subclasse de Animal)
public class Mamifero extends Animal {
    @Override
    public void emitirSom() {
        System.out.println(x:"Som do Mamífero");
    }
}
```

```java
// Classe Réptil (subclasse de Animal)
public class Reptil extends Animal {
    @Override
    public void emitirSom() {
        System.out.println(x:"Som do Réptil");
    }
}
```

Conclusão:
Em síntese, a compreensão dos relacionamentos entre classes é essencial para o desenvolvimento de software eficiente e sustentável. A aplicação adequada desses conceitos permite a construção de alicerces sólidos para sistemas mais poderosos e flexíveis.