Classe (programação)

Origem: Wikipédia, a enciclopédia livre.

Em orientação a objetos, uma **classe** é uma descrição que abstrai um conjunto de objetos com características similares. Mais formalmente, é um conceito que encapsula abstrações de dados e procedimentos que descrevem o conteúdo e o comportamento de entidades do mundo real, representadas por objetos.[1] De outra forma, uma classe pode ser definida como uma descrição das propriedades ou estados possíveis de um conjunto de objetos, bem como os comportamentos ou ações aplicáveis a estes mesmos objetos.

Linguagens de programação orientadas a objetos devem possibilitar a implementação de classes. Classes são os elementos primordiais de um diagrama de classes.

Índice

Estrutura da classe

Uma classe comumente define o estado e o comportamento de um objeto implementando atributos e métodos. Os atributos (por vezes referidos como "campos", "membros de dados" ou "propriedades"), indicam as possíveis informações armazenadas por um objeto de uma classe, representando o estado de cada objeto. Os métodos (por vezes referidos como "operações" ou serviços) são procedimentos que formam os comportamentos e ações oferecidos por objetos de uma classe, sendo responsáveis por alterar o estado ou fornecer informações sobre um objeto.

Outros elementos associados a uma classe são:

- Construtor e destrutor - métodos especiais que definem o comportamento do objeto de uma classe no momento da sua criação e destruição. Em algumas linguagens, como em C++, um método destrutor é utilizado para liberar recursos do sistema (como memória)[2], já em outras, como em Java e C#, isto é realizado de modo automático pelo coletor de lixo.
- Propriedade - define o acesso a um estado do objeto.
- Evento - define um ponto em que o objeto pode chamar outros procedimentos de acordo com seu comportamento e estado interno.

Encapsulamento

No paradigma de orientação a objetos, é possível encapsular o estado de um objeto. Em termos práticos, isso se realiza limitando o acesso a atributos de uma classe exclusivamente através de seus métodos. Para isso, as linguagens orientadas a objeto oferecem limitadores de acesso para cada membro de uma classe.

Tipicamente os limitadores de acesso são:

- público (*public*) - o membro pode ser acessado por qualquer classe. Os membros públicos de uma classe definem sua interface
- protegido (*protected*) - o membro pode ser acessado apenas pela própria classe e suas sub-classes
- privado (*private*) - o membro pode ser acessado apenas pela própria classe

Cada linguagem de programação pode possuir limitadores de acesso próprios. Por exemplo, em Java, o nível de acesso padrão de um membro permite que qualquer classe de seu pacote (*package*) possa ser acessado. Em C#, o limitador de acesso interno (*internal*) permite que o membro seja acessado por qualquer classe do *Assembly* (isto é, da biblioteca ou executável).

No exemplo abaixo, implementado em Java, a classe *Pessoa* permite o acesso ao atributo *nome* somente através dos métodos *setNome* e *getNome*.

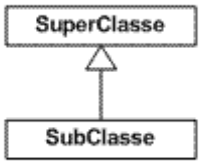
```java
public class Pessoa {
    private String nome;

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }
}
```

Outro exemplo em C++:

```cpp
#include <string>

class Pessoa
{
    private:
        std::string nome;

    public:
        std::string getNome();
        void setNome(std::string nome);
};

// Definição dos métodos

std::string Pessoa::getNome()
{
    return this->nome;
}

void Pessoa::setNome(std::string nome)
{
    this->nome = nome;
}
```

Herança

A herança é um relacionamento pelo qual uma classe, chamada de sub-classe, herda todos comportamentos e estados possíveis de outra classe, chamada de super-classe ou classe base. É permitido que a sub-classe estenda os comportamentos e estados possíveis da super-classe (por isso este relacionamento também é chamado de extensão). Essa extensão ocorre adicionando novos membros a sub-classe, como novos métodos e atributos.



Representação de herança entre classes em UML

É também possível que a sub-classe altere os comportamentos e estados possíveis da super-classe. Neste caso, a sub-classe sobrescreve membros da super-classe, tipicamente métodos.

Quando uma classe herda de mais de uma super-classe, ocorre uma herança múltipla. Esta técnica é possível em C++ e em Python, mas não é possível em Java e C#, no entanto estas linguagens permitem múltipla tipagem através do uso de interfaces.

Polimorfismo

Na programação orientada a objetos, o **polimorfismo** permite que referências de tipos de classes mais abstratas representem o comportamento das classes concretas que referenciam. Assim, um mesmo método pode apresentar várias formas, de acordo com seu contexto. O polimorfismo é importante pois permite que a semântica de uma interface seja efetivamente separada da implementação que a representa. O termo polimorfismo é originário do grego e significa 'muitas formas' (poli = muitas, morphos = formas).

Associação

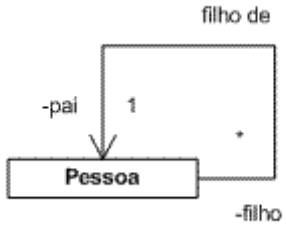
Uma associação é um vínculo que permite que objetos de uma ou mais classes se relacionem. Através destes vínculos é possível que um objeto convoque comportamentos e estados de outros objetos.

As associações podem ser:

- unárias - quando a associação ocorre entre objetos de uma mesma classe.
- binárias - quando a associação ocorre entre dois objetos de classes distintas.
- múltiplas - quando a associação ocorre entre mais de dois objetos de classes distintas.

Cada associação possui características de:

- cardinalidade ou multiplicidade - determina quantos objetos no sistema são possíveis em cada vértice da associação.
- navegação - se é possível para cada objeto acessar outro objeto da mesma associação.



Exemplo de associação unária em UML

No exemplo de associação unária acima, cada pessoa tem um único pai (cardinalidade 1) e qualquer número de filhos (cardinalidade *). De acordo com a seta de navegação, só é possível navegar para o pai de cada pessoa. Desta forma cada objeto da classe Pessoa consegue acessar seu objeto pai, mas não consegue acessar seus objetos filhos.

Agregação

Tipo de relacionamento com características todo-parte, onde existe um grau de acoplamento entre o todo e as partes menos intenso, podendo haver certo grau de independência entre eles.

Composição

Tipo de relacionamento com características todo-parte, onde existe um alto grau de coesão entre o todo e as partes, com total grau de dependência entre eles (todo e as partes). Desta forma, se o todo não existir, as partes também não existirão.

Um exemplo de composição é a mão:

Uma mão é composta por dedos. Os dedos compõem a mão.

Não há lógica em existir um dedo sem mão, porém pode-se ter uma mão sem um ou mais dedos

Classes abstratas e concretas

Uma classe abstrata é desenvolvida para representar entidades e conceitos abstratos. A classe abstrata é sempre uma superclasse que não possui instâncias. Ela define um modelo (*template*) para uma funcionalidade e fornece uma implementação incompleta - a parte genérica dessa funcionalidade - que é compartilhada por um grupo de classes derivadas. Cada uma das classes derivadas completa a funcionalidade da classe abstrata adicionando um comportamento específico.

Uma classe abstrata normalmente possui métodos abstratos. Esses métodos são implementados nas suas classes derivadas concretas com o objetivo de definir o comportamento específico. O método abstrato define apenas a assinatura do método e, portanto, não contém código.

Por outro lado, as classes concretas implementam todos os seus métodos e permitem a criação de instâncias. Uma classe concreta não possui métodos abstratos e, geralmente, quando utilizadas neste contexto, são classes derivadas de uma classe abstrata.

Ver também

- Método
- Atributo
- Construtor
- Destrutor
- Arquitetura de dados
- Administração de dados
- Modelagem de dados

Referências

1. Roger Pressman, Bruce Maxim (2016). *Engenharia de Software* 8 ed. [S.l.]: McGraw Hill Brasil. ISBN 9788580555349
2. http://www.learncpp.com/cpp-tutorial/86-destructors/ *Destructors – Learn C++*
