Inatel
Instituto Nacional de Telecomunicações

Programação Orientada a Objetos

Cap.7 - Classes Abstratas

Prof. Me. Renzo P. Mesquita

Objetivos

- Compreender o que são Classes Abstratas e quais suas vantagens;
- Ser capaz de utilizar Classes Abstratas, quando necessário;

Capítulo 7
Classes Abstratas

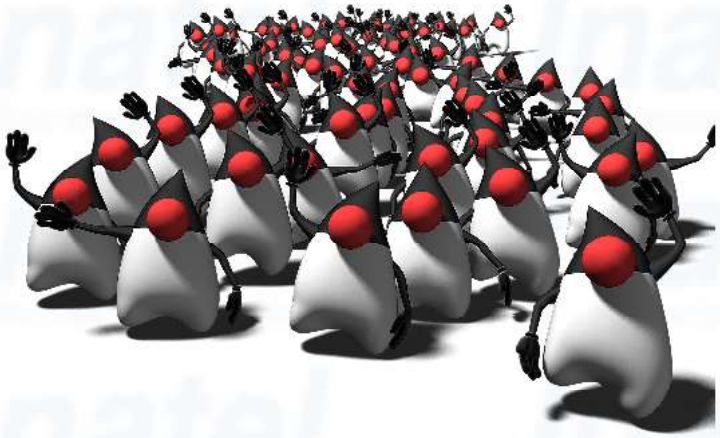
7.1. Introdução;
7.2. Classes Abstratas;
7.3. Métodos Abstratos;
7.4. Mais exemplos de cases com Classes Abstratas;

7.1. Introdução

No Capítulo anterior, vimos um exemplo muito interessante de Classe com um método que permitia somar as bonificações de todos os Funcionários, independente do tipo de cada um.

Relembrando:

```
public class ControleDeBonificacoes {

    private double totalDeBonificacoes = 0;

    public void addBonificacao(Funcionario f)
    {
        totalDeBonificacoes += f.calculaBonificacao();
    }

    public double getTotalDeBonificacoes() {
        return totalDeBonificacoes;
    }
}
```



Utilizamos a Classe Funcionário para poder aplicar o Polimorfismo. Se não fosse ela, precisaríamos criar um método para receber cada um dos tipos de Funcionários, consequentemente, deixando o códgo bem maior.

7.1. Introdução

Em alguns sistemas, pode acontecer de criarmos Classes com apenas o intuito de economizar código e ganharmos Polimorfismo, assim, permitindo criar métodos mais genéricos e que se encaixem em diversas outras Classes.

Como exemplo, podemos citar nossa Classe Funcionário dos exemplos anteriores: ela foi usada de forma "genérica" para enonomizar código nos seus filhos.

Agora a pergunta que não se cala: apesar de usarmos uma variável do tipo Funcionário para referenciar os seus filhos, faz sentido INSTANCIARMOS um Objeto do tipo Funcionário?

7.1. Introdução

Dar "new" em Funcionário pode não fazer sentido, isto é, não queremos criar Objetos do tipo Funcionário, mas sim, Objetos do tipo Gerente ou Diretor ou Engenheiro etc, pois todo Funcionário tem de ser de um tipo específico, não basta ser apenas Funcionário.

Exemplo:

```
// Apesar de não dar erro,
// semanticamente isso abaixo faz sentido? Não! :(
Funcionario f1 = new Funcionario();

// Este faz sentido! :)
Gerente f2 = new Gerente();
// Até este faz sentido! :)
Funcionario f3 = new Gerente();
```

Para forçarmos o programador a criar códigos que fazem mais sentido, veremos o conceito de Classes ABSTRATAS.

7.2. Classes Abstratas

As Classes Abstratas não podem ser instanciadas! Elas são Classes feitas especialmente para servirem de modelo para suas classes filhas. Classes que podem ser instanciadas são chamadas de Concretas.

No nosso exemplo dos Funcionários de uma empresa, é inadimissível que um objeto seja apenas do tipo Funcionário. Por isso, deveremos usar a palavra chave *abstract* no cabeçalho da Classe para impedir que um objeto dela possa ser instanciado.

Exemplo:

```
public abstract class Funcionario {

    protected String nome;
    protected String cpf;
    protected float salario;
    // ...restante da classe
```

Vale ressaltar que ainda poderemos criar variáveis referências do tipo Funcionário, mas não mais instanciar objetos desta Classe. Se tentarmos fazer isso, acontecerá um erro de compilação.

7.2. Classes Abstratas

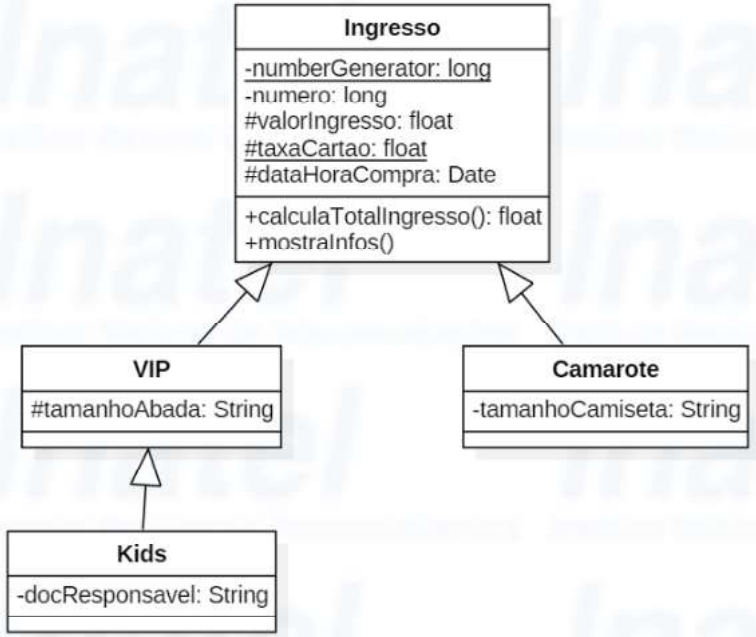
Mas qual a vantagem de uma Classe Abstrata?

Por enquanto, a única diferença é que não podemos, por exemplo, instanciar um Objeto do tipo Funcionário, dando mais consistência ao sistema, pois um Funcionário deve ser de um tipo específico.

Mas então toda Classe Mãe deve ser Abstrata?

De jeito nenhum! Se em um sistema a Classe Mãe fizer sentido, ela também poderá ser Concreta. Por exemplo, no nosso exercício BUTickets, a Classe VIP fazia sentido ser instanciada, logo, ela era Mãe e Concreta.

A decisão se uma Classe Mãe será Abstrata ou Concreta depende do escopo do projeto.



7.3. Métodos Abstratos

No nosso exemplo anterior de Funcionários de uma empresa, se o método calculaBonificação() da Classe Mãe não fosse reescrito pelas Classes Filhas, ele seria herdado da Classe Mãe.

Relembrando:

```
public class Funcionario {

    protected String nome;
    protected String cpf;
    protected float salario;

    public float calculaBonificacao()
    {
        return salario * 0.10f;
    }
    // ...restante da classe
}
```

Se levarmos em consideração que cada funcionário em nosso sistema tem uma regra TOTALMENTE diferente para ser bonificado, faz algum sentido ter esse método na Classe Funcionário?

7.3. Métodos Abstratos

Em uma Classe Abstrata, podemos definir que determinado método será SEMPRE reescrito pelas Classes Filhas e Concretas. Isto é um método Abstrato.

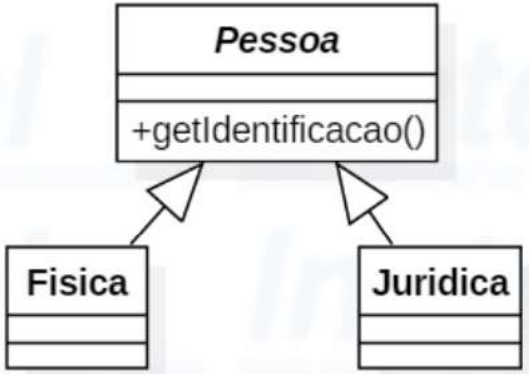
Exemplo:

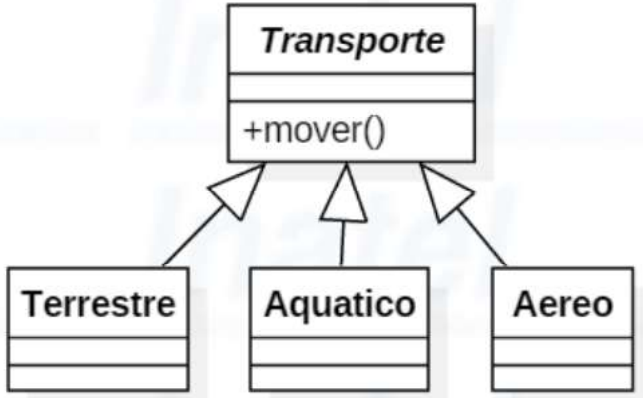
```
public abstract class Funcionario {

    protected String nome;
    protected String cpf;
    protected float salario;

    public abstract float calculaBonificacao();
    // ...restante da classe
```

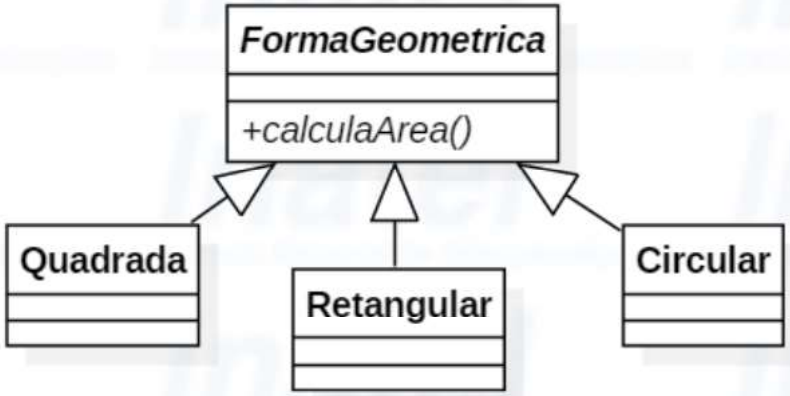
- Repare que um método Abstrato não possui um corpo;
- Agora, sempre que alguém chamar o método calculaBonificacao(), vai executar o corpo da Classe filha em questão;
- Agora, qualquer Classe que estende Funcionário, será OBRIGADA a reescrever este método;
- Um método Abstrato OBRIGA a Classe em que ele se encontra também a ser Abstrata;

7.4. Mais exemplos de cases com Classes Abstratas

Abaixo alguns outros exemplos de Classes e métodos que poderiam se comportar de forma Abstrata em muitos casos:





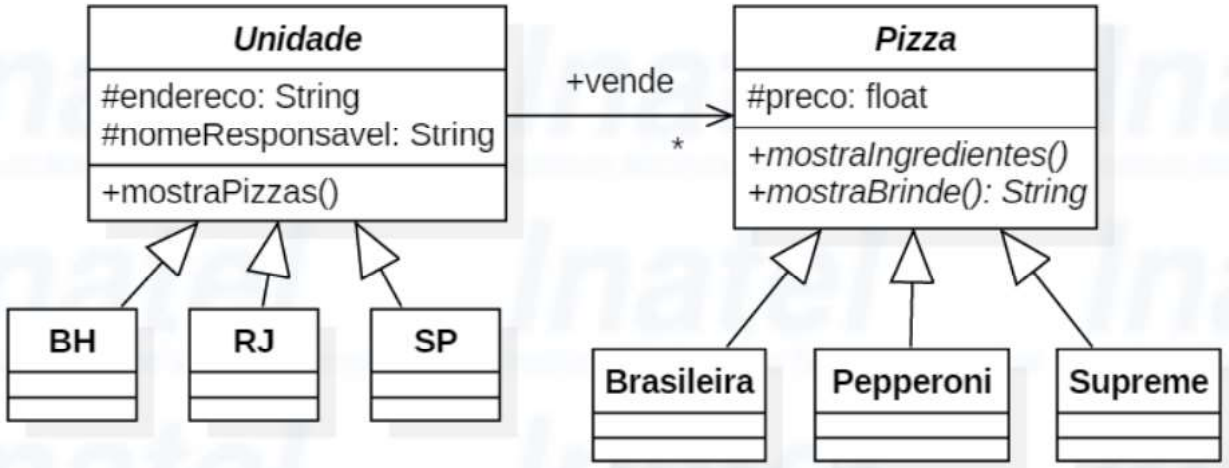


entre infinitas outras..

7.4. Mais exemplos de cases com Classes Abstratas

Exercício 1 - PromocaoPizzaHut

O PizzaHut BH, RJ e SP está com uma promoção muito legal. Cada Pizza comprada vem com um brinde especial. Crie Classes em Java que atendam a seguinte especificação:



Dicas:

Unidade	Pizzas
BH	Pepperoni
	Supreme
RJ	Pepperoni
	Brasileira
SP	Todas

Pizza	Ingredientes	Brinde
Pepperoni	Pepperoni e Mussarela	Caneta
Brasileira	Mussarela, Requeijão, Presunto e Azeitona	Chaveiro
Supreme	Mussarela, Cebola, Pimentão e Azeitona	Caneca

1. Todas as Classes Mães e seus métodos aqui são Abstratos! (Exceto o método mostraPizzas());
2. Preços: Pepperoni (R$15), Brasileira (R$20) e Supreme (R$25);
3. No final, imprima as informações referentes a cada Pizza e as Pizzas que são vendidas por cada unidade;

FIM
DO
CAPÍTULO 7

Próximo Capítulo

Interfaces