Classes abstratas, polimorfismo e interfaces

Prof. Ítalo Assis

Ajude a melhorar este material =]

Encontrou um erro? Tem uma sugestão?

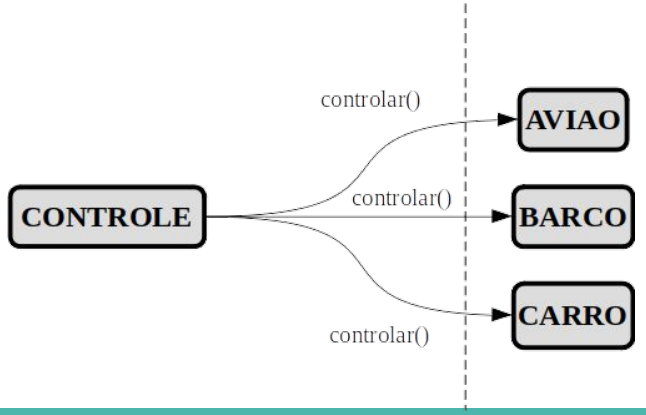
Envie e-mail para italo.assis@ufersa.edu.br

Agenda

- Motivação
- Classes abstratas
- Polimorfismo
  - Polimorfismo de inclusão
  - Polimorfismo de sobrecarga
  - Polimorfismo de sobreposição
- Interfaces

Motivação

- Você é engenheiro de uma empresa especializada em controlar veículos remotamente;
- A empresa deseja criar um sistema capaz de controlar diferentes tipos de veículos (barcos, carros, aviões...);
- Seguindo o paradigma de orientação a objeto, **como você modelaria esse problema?**
- Vamos dividir o problema em dois:
  - Modelagem dos veículos;
  - Modelagem do controle.



Prática: implementar o projeto do slide

Motivação

- Modelagem dos veículos - 1ª tentativa: **uma classe para cada entidade**

BARCO
capacidade profundidade cadBarco
moverFrente()

CARRO
capacidade qtdRodas placa
moverFrente()

AVIAO
capacidade cadANAC cadAviao
moverFrente()

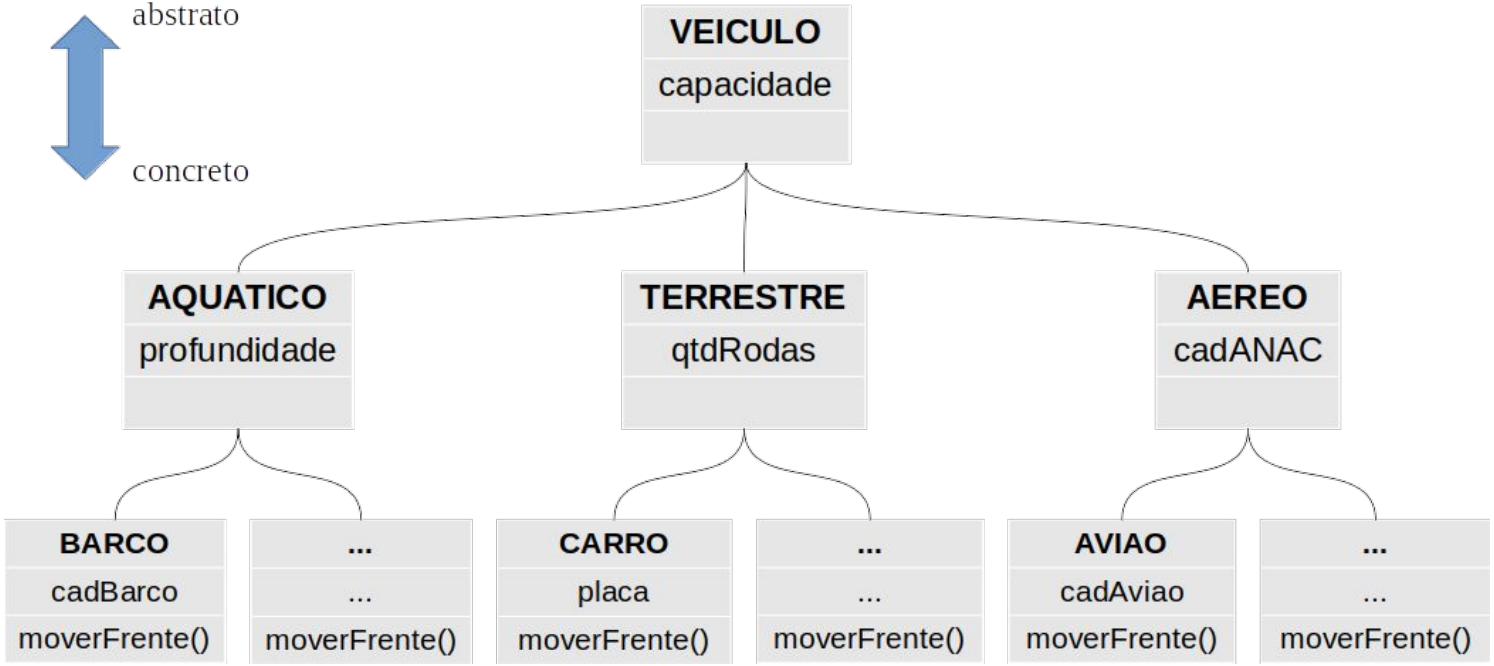
Motivação

- Concluímos a modelagem dos veículos! =)
- **Mas essa é mesmo a melhor forma?**
- Se a empresa determinar que todos os veículos devem ter capacidade mínima igual a 10, o que precisaria ser mudado?
    - Validar os valores inseridos para o atributo capacidade através dos construtores e métodos *set* de cada classe.
- A não utilização de herança reduz o reuso de código.

Prática: implementar o projeto do slide

Motivação

- Modelagem dos veículos - 2ª tentativa: **uso de hieraquia de classe**



Classes abstratas

- Às vezes é útil declarar classes - chamadas **classes abstratas** - para as quais você não pretende criar objetos;
- O propósito de uma classe abstrata é fornecer uma superclasse apropriada a partir da qual outras classes podem herdar e assim compartilhar um modelo comum;
- Uma classe abstrata não pode ser utilizadas para instanciar objetos.

Classes abstratas

- Você cria uma classe abstrata declarando-a com a palavra-chave *abstract*;
  - *public abstract class NomeDaClasse{}*
- Uma classe abstrata normalmente contém um ou mais métodos abstratos;
- Um método abstrato é um método que não fornece implementação e tem palavra-chave *abstract* na sua declaração;
  - *public abstract void nomeDoMetodo();*
- Uma classe abstrata pode ser derivada por outras classes que, por sua vez, deverão implementar seus métodos abstratos ou declarar-se também abstratas;

Classes abstratas

- Métodos e classes *final*
  - Vimos que as variáveis declaradas *final* não podem ser modificadas depois de serem inicializadas
  - Os métodos *final* não podem ser sobrescritos
  - Classes *final* não podem ser superclasses
- Operador *instanceof*
  - Determina se um objeto específico é uma instância de uma classe
  - *if (empregado instanceof Empregado) { //... }*
- Prática: Vamos reformular nosso projeto utilizando classes abstratas?

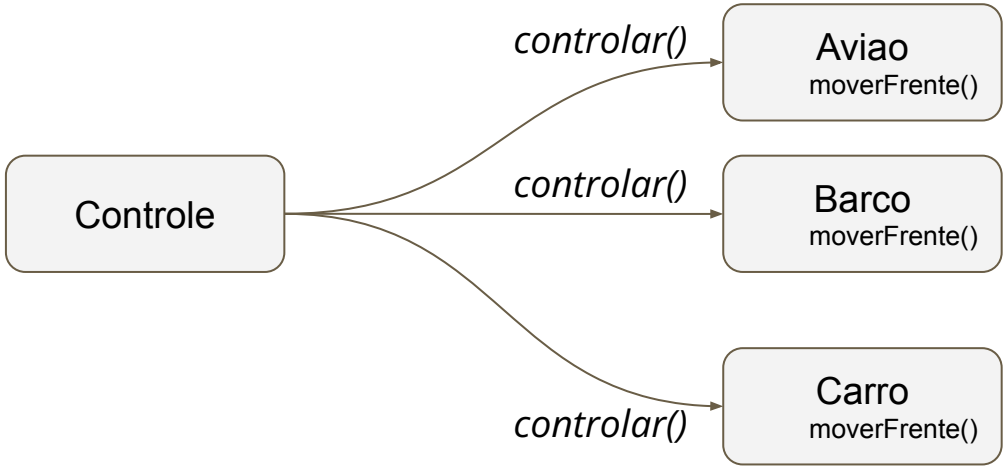
Motivação

- Você é engenheiro de uma empresa especializada em controlar veículos remotamente;
- A empresa deseja criar um sistema capaz de controlar diferentes tipos de veículos (barcos, carros, aviões...);
- Seguindo o paradigma de orientação a objeto, como você modelaria esse problema?
- Vamos dividir o problema em dois:
  - Modelagem dos veículos; ✔
  - **Modelagem do controle**.

Prática: implementar o projeto do slide

Motivação

- Modelagem do controle - 1ª tentativa: uma classe com um método para cada classe concreta;

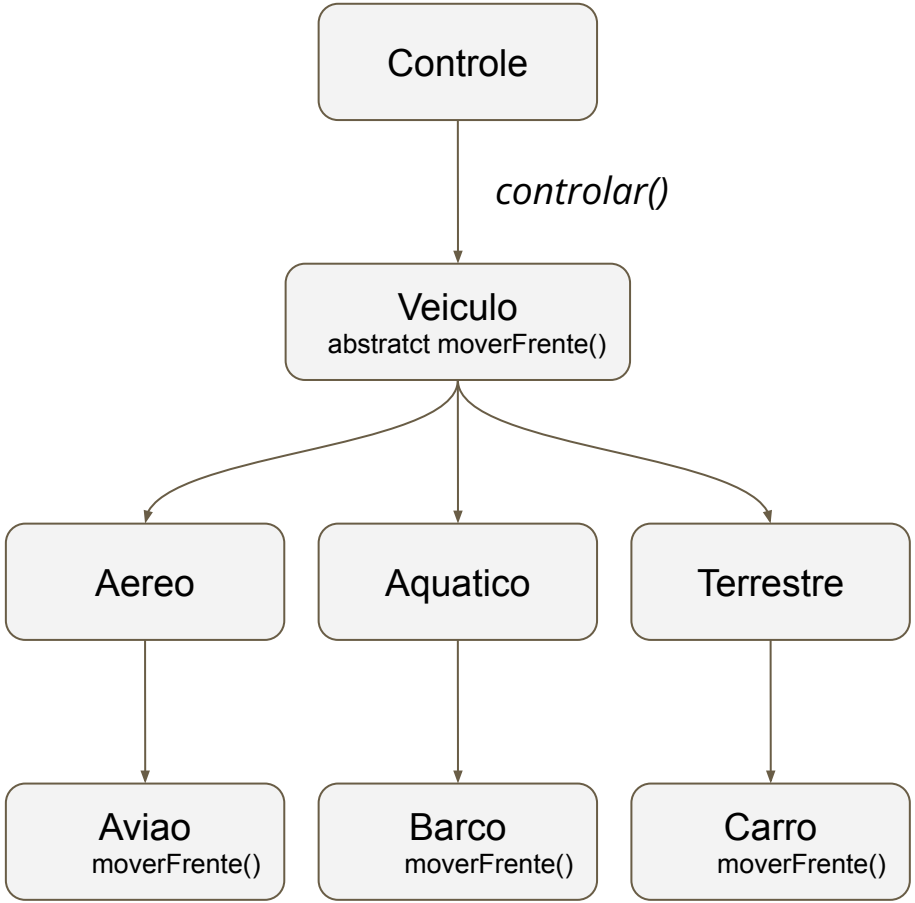


Polimorfismo

- O **Polimorfismo** permite que diferentes objetos (avião, barco, carro) respondam a uma mesma mensagem (*controlar*) de formas diferentes (*moverFrente*);
- Está relacionado à possibilidade de se usar o mesmo nome para métodos diferentes e à capacidade que o programa tem em discernir, dentre os métodos homônimos, aquele que deve ser executado;
- Pode ser classificado como polimorfismo:
  - de inclusão;
  - de sobrecarga;
  - ou de sobreposição.

Polimorfismo de inclusão

- **Substituição da classe mãe por seus descendentes**;
- Assinatura do método contém parâmetro da classe mãe (Veiculo);
- Na chamada do método podemos passar instâncias das classes filhas (Aviao, Barco, Carro);
- Caso o método *controlar* chame o método *moverFrente* de Veiculo, cada objeto irá buscar a implementação mais próxima desse método na árvore de hierarquia.



Prática: implementar o projeto do slide

Motivação

- Você é engenheiro de uma empresa especializada em controlar veículos remotamente;
- A empresa deseja criar um sistema capaz de controlar diferentes tipos de veículos (barcos, carros, aviões...);
- Seguindo o paradigma de orientação a objeto, como você modelaria esse problema?
- Vamos dividir o problema em dois:
  - Modelagem dos veículos; ✔
  - Modelagem do controle. ✔
- **Mas e os polimorfismos de sobrecarga e sobreposição?**

Polimorfismo de sobrecarga

- Ocorre quando definimos **múltiplas funções com o mesmo nome** porém tipos/quantidades diferentes de parâmetros;
- Técnica comum para construtores;
- Prática: criar um segundo construtor para classe Carro

Polimorfismo de sobreposição

- Ocorre quando redefinimos um método da classe mãe em uma classe filha;
- O **método da classe filha se sobrepõe**;
- Prática:
  - Todas as classes em Java herdam da classe *Object*;
  - A classe *Object* possui uma implementação do método *toString*;
  - Verique a impressão de um objeto do tipo *Carro* com e sem implementação de método *toString* na classe *Carro*.

Interface

- Outra forma de obter abstração em Java é com interfaces.
  - Assim como as classes abstratas, as interfaces não podem ser usadas para criar objetos
- Uma **interface** é como uma "**classe completamente abstrata**" utilizada para agrupar métodos relacionados **sem implementação**.
- A interface também pode conter constantes.
- Funciona como uma espécie de contrato
  - Uma classe concreta que “assina” o contrato é obrigada a implementar todos os métodos da interface.

```
interface Animal {
  public void somDoAnimal();
  public void dormir();
}
```

Interface

- Os métodos da interface devem ser "implementados" por outra classe com a palavra-chave ***implements*** (em vez de *extends*).
- O corpo do método da interface é fornecido pela classe concreta que a implementa.

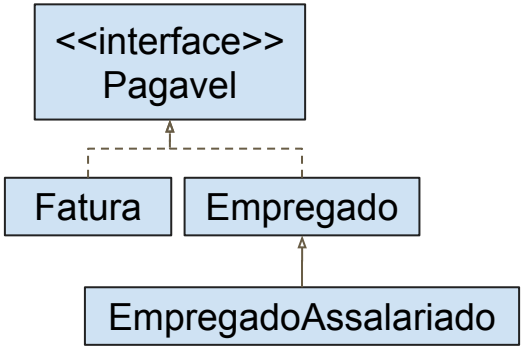
```
interface Animal {
  public void somDoAnimal();
  public void dormir();
}

class Gato implements Animal {
  public void somDoAnimal() {
    System.out.println("miau");
  }
  public void dormir() {
    System.out.println("Zzz");
  }
}

class Main {
  public static void main(String[] args) {
    Gato meuGato = new Gato();
    meuGato.somDoAnimal();
    meuGato.dormir();
  }
}
```

Interface

- Segurança: interfaces ocultam detalhes e mostram apenas as características importantes de um objeto.
- Java não suporta herança múltipla (uma classe só pode herdar de uma superclasse). No entanto, isso pode ser alcançado com interfaces, porque a classe pode implementar várias interfaces.
    - *class CarroAnfibio implements Terrestre, Aquatico { ... }*
- Uma interface é muitas vezes usada quando classes que não estão relacionadas por uma hierarquia de classes precisam compartilhar métodos e constantes comuns. Isso permite que objetos de classes não relacionadas sejam processados polimorficamente.

Prática

- Construa um aplicativo que pode determinar os pagamentos para funcionários e também faturas segundo o modelo
  - *Pagavel* deve conter um método para obter o valor do pagamento
  - *Fatura* deve conter atributos para o preço do item e a quantidade do item
  - *Empregado* deve ser uma classe abstrata e ter os atributos nome e carteira de trabalho
  - *EmpregadoAssalariado* deve ter um atributo para o valor do seu salário
  - *Fatura* e *EmpregadoAssalariado* são classes concretas
  - Na classe executável, escreva um método que recebe um objeto *Pagavel* e imprime o custo do pagamento



Os códigos relacionados a esta aula estão disponíveis em

https://github.com/italoaug/Programacao-Orientada-a-Objetos/blob/main/codigos/abstrata-polimorfismo-interface

Referências

DEITEL, Paul J; DEITEL, Harvey M. **Java: como programar**. 8.ed. São Paulo: Pearson Education do Brasil, 2010. xxix, 1144 p. ISBN: 9788576055631.

SANTOS, Rafael. **Introdução à programação orientada a objetos usando Java**. Rio de Janeiro: Editora Campus, 2003.

INSTITUTO METRÓPOLE DIGITAL. **Programação Orientada a Objetos**. 2015. Disponível em: https://materialpublic.imd.ufrn.br/curso/disciplina/5/8 . Acesso em: 21 out. 2020. 2ª Edição. ISBN: 978-85-7064-002-4.

BATISTA, Rogério da Silva; MORAES, Rafael Araújo de. **Introdução à Programação Orientada a Objetos**. 2013. Disponível em: http://proedu.rnp.br/bitstream/handle/123456789/611/Intro_Progr_OrientadaObjetos_PB_CAPA_FICHA_ISBN_20130813.pdf . Acesso em: 21 out. 2020.

W3SCHOOLS. **Java Interface**. Disponível em: https://www.w3schools.com/java/java_interface.asp. Acesso em: 28 out. 2021.