Monitoria POO

Polimorfismo, Sobrecarga, Sobreposição de métodos e Interface

Monitor: Johnes Thomas

 /johnesthomas

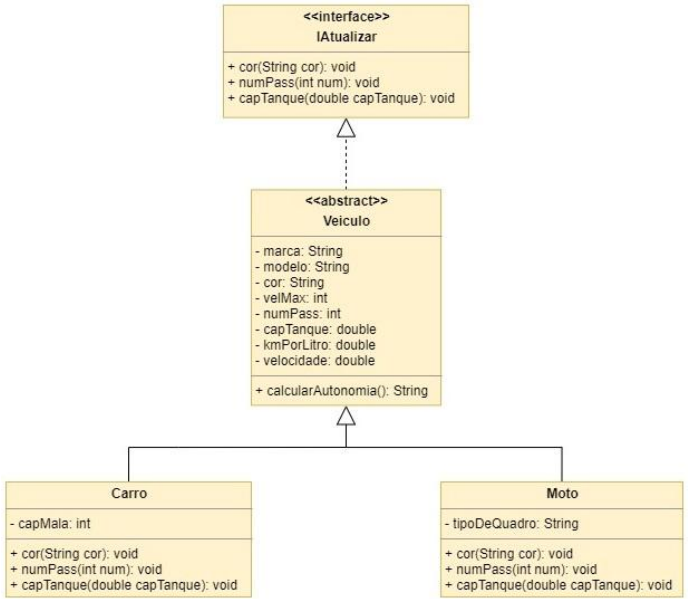
 /johnesthomas

2020 - ANO DO FOCO

GRUPO TIRADENTES
Inspirando o futuro

Introdução

Vamos conhecer um pouco de Polimorfismo, Sobreposição de Métodos, Sobrecarga e Interface. Para isso vamos implementar o diagrama de classe ao lado. Esses mecanismos são utilizados para otimização do código.



Polimorfismo

Um dos pilares da POO (Programação Orientada a Objetos) é o Polimorfismo. Ele é um recurso que permite que um elemento seja utilizado de diferentes formas, ou seja, uma subclasse pode usar o mesmo método da superclasse de maneira diferente.

```
public String calcularAutonomia() {
    double autonomia = getCapTanque() * getKmPorLitro();
    return Double.toString(autonomia);
}
```

Método criado antes da classe **Veiculo** ser definida como classe abstrata, e por isso tem corpo, e não só uma assinatura.

```
24    @Override
25    public String calcularAutonomia() {
26        double autonomia = (super.getCapTanque() * super.getKmPorLitro());
27        return "Autonomia da Moto: " + Double.toString(autonomia);
28
29    }
```

```
23    @Override
24    public String calcularAutonomia() {
25        double autonomia = (super.getCapTanque() * super.getKmPorLitro());
26        return "Autonomia do Carro: " + Double.toString(autonomia);
27
28    }
```

Sobrecarga

É um dos tipos específicos de polimorfismo.

Sobrecarga é um recurso que a classe oferece que permitir realizar mais de uma operação com o mesmo nome. A sobrecarga é muito utilizada nos construtores das classes, podendo criar diversos tipos de variações da mesma classe logo na instância do objeto. Podemos observar na imagem ao lado.

```java
public class Veiculo {

    private String marca;
    private String modelo;
    private String cor;
    private int velMax; // velocidade maxima
    private int numPass; // numero de passageiros
    private double capTanque; // capacidade do tanque de combustivel
    private double kmPorLitro;

    public Veiculo () {
    }

    public Veiculo(String marca, String modelo, String cor, int velMax, int numPass, double capTanque, double kmPorLitro) {
        this.marca = marca;
        this.modelo = modelo;
        this.cor = cor;
        this.velMax = velMax;
        this.numPass = numPass;
        this.capTanque = capTanque;
        this.kmPorLitro = kmPorLitro;
    }

    public Veiculo(String marca, String modelo, String cor, int numPass, double capTanque, double kmPorLitro) {
        super();
        this.marca = marca;
        this.modelo = modelo;
        this.cor = cor;
        this.numPass = numPass;
        this.capTanque = capTanque;
        this.kmPorLitro = kmPorLitro;
        this.velMax = 200;
    }

    public Veiculo(String marca, String modelo, int numPass, double capTanque, double kmPorLitro) {
        super();
        this.marca = marca;
        this.modelo = modelo;
        this.numPass = numPass;
        this.capTanque = capTanque;
        this.kmPorLitro = kmPorLitro;
        this.cor = "Branco";
        this.velMax = 180;
    }
```

Obs.: Classe principal antes das alterações !

Sobreposição de Métodos

É um conceito do polimorfismo que nos permite reescrever um método.

Podemos reescrever o mesmo método da superclasse nas subclasses. Os métodos permanecem com o mesmo nome, tipo de retorno e a mesma quantidade de parâmetro do método principal. Os métodos sobrepostos serão implementados com especificações da superclasse, podendo adicionar algo a mais ou não.

```java
public String calcularAutonomia() {
    double autonomia = getCapTanque() * getKmPorLitro();
    return Double.toString(autonomia);
}
```

```java
package polimorfismo;

public class Carro extends Veiculo {
    private int capMala;

    public Carro() {
    }

    public Carro(String marca, String modelo, String cor, int velMax, int numPass, double capTanque, double kmPorLitro,
            int capMala) {
        super(marca, modelo, cor, velMax, numPass, capTanque, kmPorLitro);
        this.capMala = capMala;
    }

    public int getCapMala() {
        return capMala;
    }

    public void setCapMala(int capMala) {
        this.capMala = capMala;
    }

    @Override
    public String calcularAutonomia() {
        double autonomia = (super.getCapTanque() * super.getKmPorLitro());
        return "Autonomia do Carro: " + Double.toString(autonomia);

    }
}
```

```java
package polimorfismo;

public class Moto extends Veiculo {

    private String tipoDeQuadro;

    public Moto() {
    }

    public Moto(String marca, String modelo, String cor, int velMax, int numPass, double capTanque, double kmPorLitro,
            String tipoDeQuadro) {
        super(marca, modelo, cor, velMax, numPass, capTanque, kmPorLitro);
        this.tipoDeQuadro = tipoDeQuadro;
    }

    public String getTipoDeQuadro() {
        return tipoDeQuadro;
    }

    public void setTipoDeQuadro(String tipoDeQuadro) {
        this.tipoDeQuadro = tipoDeQuadro;
    }

    @Override
    public String calcularAutonomia() {
        double autonomia = (super.getCapTanque() * super.getKmPorLitro());
        return "Autonomia da Moto: " + Double.toString(autonomia);

    }
}
```

Interface

O objetivo de criar um Interface é fazer com que a classe que a implementa seja obrigada a usar seus métodos. De outra forma, ela é definida como um “contrato”. Os métodos criados na Interface não tem corpo, apenas assinatura. Podemos implementar mais de uma interface numa classe segue um exemplo: **public abstract class NomeDaClasse implements Interface1, Interface2, InterfaceN { }**

```
public interface IAtualizar {

    void cor(String cor);
    void numPass(int num);
    void capTanque(double capTanque);

}
```

Interface criada !

```
public abstract class Veiculo implements IAtualizar{

    private String marca;
    private String modelo;
    private String cor;
    private int velMax; // velocidade maxima
    private int numPass; // numero de passageiros
    private double capTanque; // capacidade do tanque de combustivel
    private double kmPorLitro;
    private double velocidade;
```

Interface implementada !

Bora praticar !

Referências

https://www.devmedia.com.br/uso-de-polimorfismo-em-java/26140#:~:text=Polimorfismo%20significa%20%22muitas%20formas%22%2C,diferentes%20ao%20receber%20uma%20mensagem -

https://www.devmedia.com.br/sobrecarga-e-sobreposicao-de-metodos-em-orientacao-a-objetos/33066

https://www.devmedia.com.br/java-interface-aprenda-a-usar-corretamente/28798

https://www.caelum.com.br/apostila-java-orientacao-objetos/interfaces