Universidade de Brasília Faculdade UnB Gama

Orientação a Objetos

Aula 11 - Polimorfismo e Classes abstratas

Daniel Porto

daniel.porto@unb.br

APRESENTAÇÃO

Polimorfismo

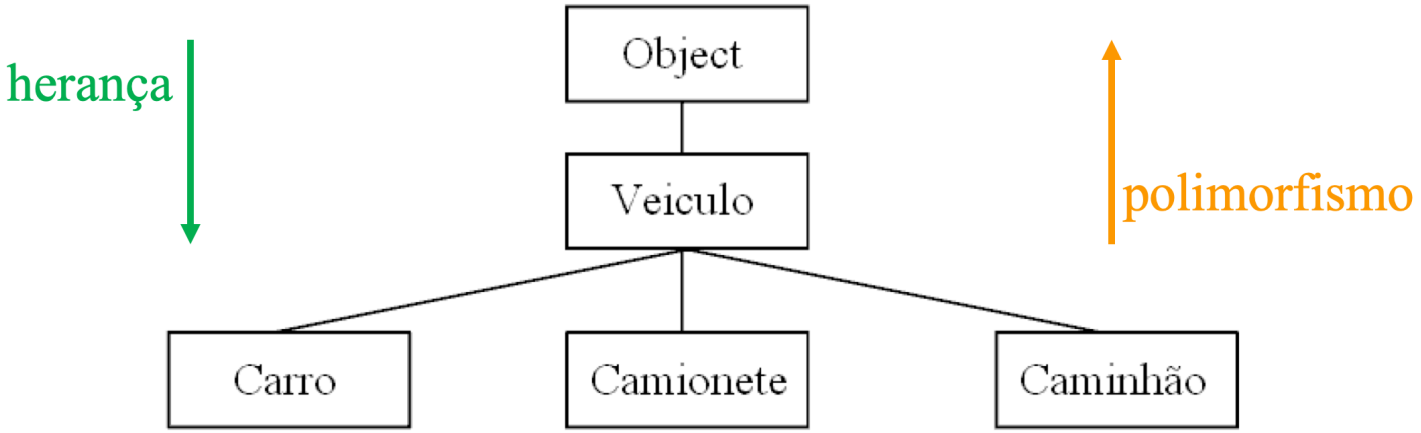
Classes abstratas

Diagrama de Classes

POLIMORFISMO

A propriedade de herança fornece um mecanismo de **especialização** interessante a POO, sendo o processo de **generalização** uma outra propriedade importante na POO, denominada Polimorfismo.

O polimorfismo trabalha na análise inversa a herança, representada em sua hierarquia de classes, indicando que cada subclasse pode assumir as características e funcionalidades de sua superclasse.



POLIMORFISMO

Esta propriedade evidencia que uma classe pode assumir diferentes formas (poli=várias e morfo=fomas), de acordo com suas classes superiores (superclasses).

```
1   /** Síntese
2    *   Conteúdo: estrutura heterogênea de Camionete
3    *     - getCarga(), setCarga(boolean), carrega(),
4    *     - descarrega()
5    */
6   public class Camionete extends Veiculo {
7       private boolean carga;
8       public boolean getCarga() {
9           return carga;
10      }
11      public void setCarga(boolean cargaParametro) {
12          this.carga = cargaParametro;
13      }
14      public void carrega() {
15          carga = true;
16      }
17      public void descarrega() {
18          carga = false;
19      }
20  }
```

POLIMORFISMO

A partir da nova classe Camionete, suponha a necessidade de acompanhar a velocidade de um Carro e de uma Camionete em outra classe (UsaVeiculos).

```
1   /** Síntese
2    *    Objetivo: acompanha velocidade de vários veículos
3    *    Entrada:  sem entrada (só atribuições)
4    *    Saída:    velocidade de cada veículo
5    */
6   public class UsaVeiculos {
7       public static void main(String[] args) {
8           Carro auto1     = new Carro();     // cria 1 carro
9           Camionete auto2 = new Camionete(); // cria 1 camionete
10          auto1.setVelocidade(120);
11          auto2.setVelocidade(70);
12          mostraVelocidade(auto1);  // parâmetro de carro
13          mostraVelocidade(auto2);  // parâmetro de camionete
14      }
15
16      public static void mostraVelocidade(Veiculo auto) {
17          System.out.println("Velocidade = " + auto.getVelocidade());
18      }
19  }
```

POLIMORFISMO

Observe que UsaVeiculos cria 2 objetos diferentes (auto1 é Carro e auto2 é Camionete) e aciona o método mostraVeiculos para mostrar a velocidade destes 2 objetos diferentes.

SOBRECARGA ≠ POLIMORFISMO

Parâmetros enviados para mostraVeiculos são de um mesmo tipo (superclasse), não sendo sobrecarga.

auto1 e auto2 são generalizações de Carro e Camionete, ou seja, os dois são Veiculo.

O método mostraVeiculo pode receber como parâmetro qualquer classe derivada de Veiculo.

Por meio do polimorfismo não foi necessário criar um método para cada tipo de objeto existente nesta classe.

EXERCÍCIO DE FIXAÇÃO

1) Elabore um programa que possibilite uma interação amigável com seus possíveis usuários onde os mesmos possam cadastrar algumas pessoas com nome e data de nascimento e derive de pessoas as características diferentes para familiar ou amigo. Estas duas especializações serão responsáveis somente pela identificação de grau de parentes para uma pessoa do seu convívio familiar ou para indicação da categoria de relacionamento para um amigo: 0-pessoal; 1-profissional; 2-estudantil.

Implemente a correta hierarquia entre os elementos envolvidos na solução deste problema, além da classe CadastraPessoa que possibilitará a criação de uma aplicação executável em Java. Permita um cadastro de até 100 pessoas e quando o usuário não quiser mais cadastrar dados apresente as opções de encerrar o programa ou pesquisar um nome desejado para confirmação deste cadastro. Faça um método para esta pesquisa que implemente e use a propriedade do polimorfismo.

CLASSE ABSTRATA

A possibilidade de derivação de algumas classes pode exigir que certos comportamentos sejam diferentes em suas subclasses, chegando até a não existirem em algumas. No entanto, os métodos implementados devem possuir uma interface de acionamento padrão (assinatura de métodos) para que tal comportamento seja usado adequadamente por suas classes.

Diante destas situações a linguagem Java permite a implementação de classes abstratas, por meio da instrução **abstract**.

```
public abstract class Veiculo {
        :
}
```

CLASSE ABSTRATA

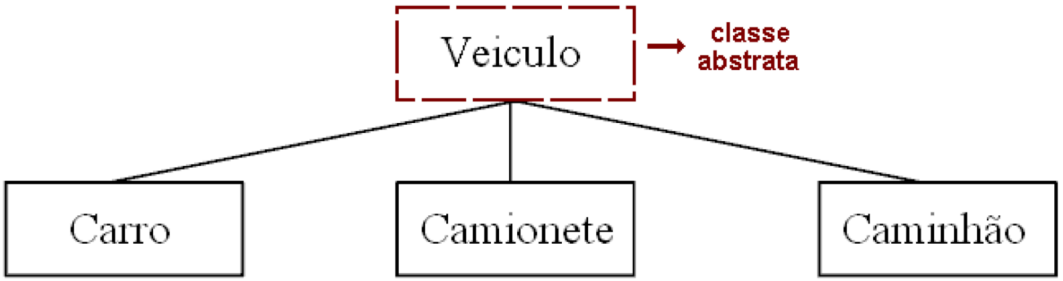
Essa instrução define uma classe, geralmente sinalizando a existência de métodos definidos como abstratos, por meio do uso da instrução abstract em sua assinatura.

```
public abstract class Veiculo {
        public abstract void mover();
                :
}
```

No entanto, é importante ressaltar que a definição de uma classe abstrata não permite a criação de um objeto a partir dela (classe abstrata), podendo ela possuir um, vários ou até mesmo nenhum método abstrato na definição de seus componentes.

CLASSE ABSTRATA

Observe o gráfico que representa a hierarquia de classes que possui Veiculo como uma classe abstrata:



A implementação deste projeto não poderá criar um objeto Veiculo, sendo necessário sua ampliação para criação de uma instancia da mesma.

Observe ainda que podem ser criadas variáveis objeto de uma classe abstrata, tendo essas variáveis que fazer referência a um objeto de uma subclasse não abstrata.

Exemplo: Veiculo auto1 = new Carro();

auto1 é variável do tipo abstrato **Veiculo** que faz referência a uma instância da subclasse **Carro** que não é **abstrata**.

CLASSE ABSTRATA

```java
/** Síntese
 *   Conteúdo: estrutura heterogênea de Veiculo
 *      - frea(),acelera(),getMarca(),getVelocidade()
 *      - setMarca(String),setVelocidade(float), abstract mover()
 */
public abstract class Veiculo {
    private String marca;
    private float velocidade;

    // Encapsulando a classe Veiculo
    public String getMarca() {
        return marca;    }
    public void setMarca(String marca) {
        this.marca = marca;    }
    public float getVelocidade() {
        return velocidade;    }
    public void setVelocidade(float velocidade) {
        this.velocidade = velocidade;    }
    public abstract int mover();   // método abstrato
    public void frea() {
        if (velocidade > 0)
        velocidade--;
    }
    public void acelera() {
        if (velocidade <= 250)
        velocidade++;
    }
}
```

CLASSE ABSTRATA

```java
/** Síntese
 *   Conteúdo: estrutura heterogênea de Carro
 *    - liga(), desliga(), getStatus()
 *    - setStatus(boolean), mover()
 */
import javax.swing.JOptionPane;
public class Carro extends Veiculo {
    private boolean status;

    // Encapsulando a classe Carro
    public boolean getStatus() {
        return status;    }
    public void setStatus(boolean status) {
        this.status = status;    }
    public void liga() {        status = true;
    }
    public void desliga() {
        status = false;    }
    // Implementação obrigatória do método abstrato
    public int mover() {
        final String combustivel = "gasolina";
        return(JOptionPane.showConfirmDialog(null,
        "O veículo está abastecido com " +
        combustivel + "?","Abastecimento",
        JOptionPane.YES_NO_OPTION,
        JOptionPane.QUESTION_MESSAGE));
    }
}
```

CLASSE ABSTRATA

```java
/** Síntese
 *    Objetivo: movimenta um carro
 *    Entrada:  sem entrada (só atribuições)
 *    Saída:    confirma o carro em movimento ou não
 */
public class UsaCarro {
    public static void main(String[] args) {
        Carro auto1     = new Carro();
        auto1.setMarca("FIAT");
        auto1.setVelocidade(0);
        auto1.setStatus(false);

        if(auto1.mover()==0) {  // método abstrato
            auto1.liga();       // método concreto de Veiculo
            auto1.acelera();    // método de Carro
            System.out.println("Carro em movimento");
        }
        else
        System.out.println("Carro precisa de " + "gasolina para se mover.");

        mostraVelocidade(auto1); // parâmetro de carro em método polimórfico
    }
    public static void mostraVelocidade(Veiculo auto) {
        System.out.println("Velocidade = "
                           + auto.getVelocidade()); // método de Veiculo
    }
}
```

CLASSE ABSTRATA

A classe **Veiculo** possui um método abstrato (mover), precisando ser definida como uma classe abstrata.

O método mover() é definido como abstrato em **Veiculo**, onde somente possuirá a assinatura do método com a instrução **abstract**.

Todas as classes não abstratas, derivadas de Veiculo, são obrigadas a implementarem o método mover().

A classe abstrata **Veiculo** possui componentes concretos, mas não pode gerar seus próprios objetos.

Carro é derivada de **Veiculo** e tem que implementar o método mover(), definido como abstrato em **Veiculo**.

CLASSE ABSTRATA

O uso de classes abstratas possibilita a declaração de classes que **definam apenas parte de uma implementação**, deixando para suas subclasses efetuarem a implementação adequada e específica de alguns métodos necessários para algumas especializações, podendo todos os seus métodos também serem abstratos.

A definição de qualquer classe **pode sobrepor métodos** de sua superclasse para declará-los como abstratos, **tornando um método concreto em abstrato** naquele ponto da hierarquia das classes. Esta possibilidade é interessante para as situações que envolvem uma implementação padrão de classe inválida para uma parte da hierarquia de classes.

EXERCÍCIO DE FIXAÇÃO

2) Implemente a classe Veiculo como uma classe abstrata que contem seus métodos abstratos acelera e mover respeitando as seguintes características para as 3 subclasses Carro, Camionete, Caminhão, além da TestaVeiculo que conterá os recursos para criação de uma aplicação executável Java.

a) todo o carro é movido a gasolina e se tiver abastecido deverá ser solicitado ao usuário qual a potência do motor (1.0, 1.4, 1.6, 1.8 ou 2.0). Após estas informações o carro deverá ser ligado e colocado em movimento com a multiplicação de sua potência por 1 (carro se coloca em movimento). Caso o mesmo não tenha sido abastecido não deve ser solicitada a potencia porque o carro permanecerá sem movimento (velocidade zero que será apresentada ao usuário);

EXERCÍCIO DE FIXAÇÃO

b) para camionete deve ser solicitado ao usuário se a potência da mesma é 2.2, 2.8 ou 4.3, sendo ligada após esta mensagem e colocada em movimento com a multiplicação de potência por 1 (camionete se coloca em movimento);

c) no caminhão deve ser solicitado ao usuário se o mesmo esta carregado ou não, onde se ele estiver deverá ser ligado e colocado em movimento com velocidade 1 e senão estiver carregado ele também deverá ser ligado, mas colocado em movimento com velocidade 1.5.

Elabore um programa que verifique qual veículo o usuário gostaria de testar e solicite as informações necessárias para o veículo informado, ligando e o colocando em movimento se for possível pela situação momentânea do veículo escolhido.

DIAGRAMA DE CLASSES

É um diagrama da UML.

Dá uma visão estática do sistema.

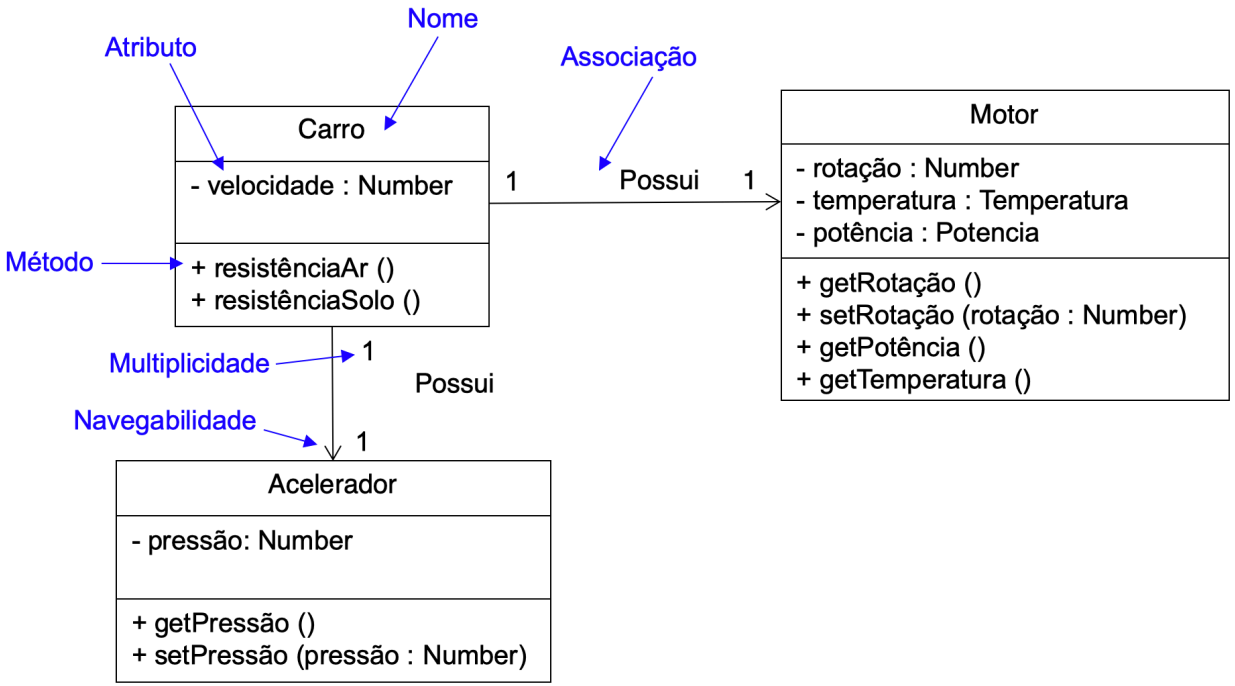
Exibe um conjunto de classes, interfaces e seus relacionamentos.

As classes especificam tanto a estrutura como o comportamento dos objetos.

São os diagramas mais utilizados em sistemas de modelagem orientados a objeto.

O diagrama de classes é composto basicamente por um conjunto de classes relacionadas entre si.

DIAGRAMA DE CLASSES



EXERCÍCIO DE FIXAÇÃO

3) Faça a modelagem (diagrama de classes) de um sistema bancário. Sua modelagem deve incluir pelo menos as classes **ContaCorrente** e **ContaPoupança**.