Polimorfismo

P. O. O.

Prof. Grace

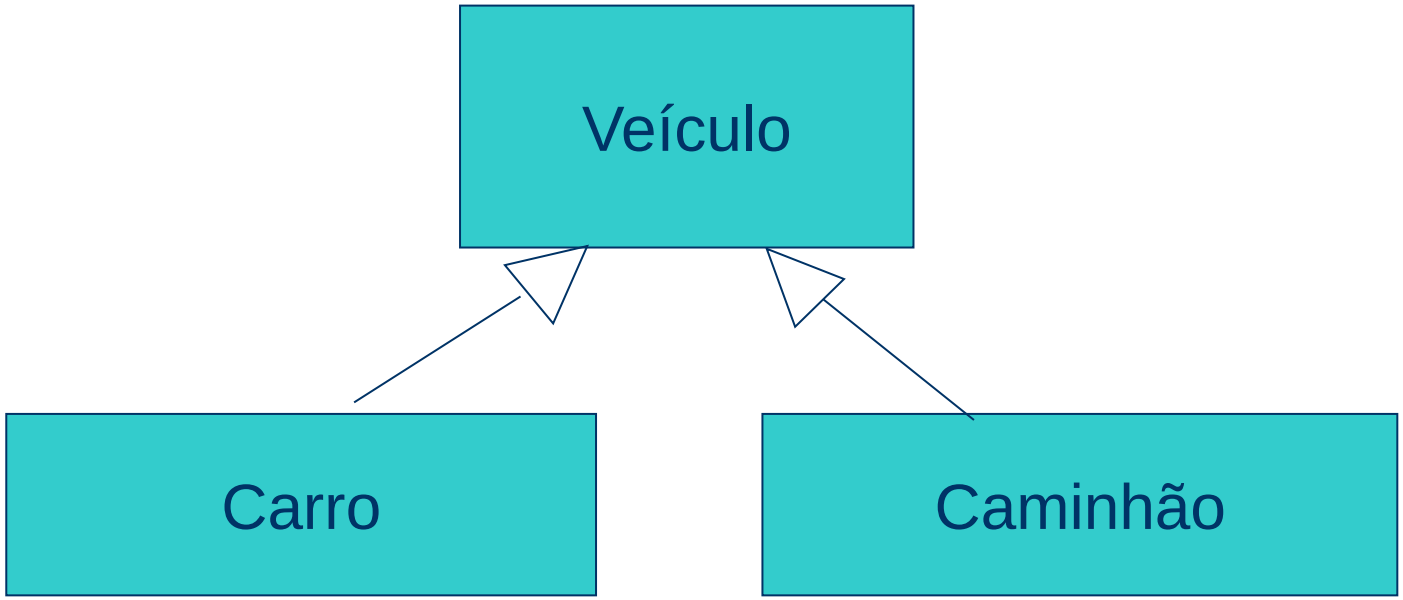
Aula 10

- Revisão Herança
- Polimorfismo
- Atividade

Relembrando Herança

- Cria uma nova classe a partir de uma classe existente:
  - absorvendo os dados e comportamentos da classe existente; e
  - aprimorando-a com novas capacidades.
- Adota um relacionamento hierárquico entre classes
- Permite melhor organização e reuso de código

Exemplo: Veículos



Modelo: Classe Veículo

Veiculo
- modelo: String - placa: String - ano Fabricação: int - valor: double
+ <<constructor>> Veiculo (mod: String; pl: String; ano: int; vlr: double) + setModelo (mod: String) + setPlaca (pl: String) + setAno (ano: int) + setValor (vlr: double) + getModelo: String + getPlaca: String + getAno: int + getValor: double + deprecia (tx: float) + imprime( )

Subclasse Carro

Carro
- numPortas: int - anoModelo: int
+ <<constructor>> carro (mod: String; pl: String; nPortas: int; anoFabr: int; anoMod: int; vlr: double) + setPortas (nPortas: int) + setAnoModelo (ano: int) + getPortas: int + getAnoModelo: int

Teste - Classe Carro

```
1 public class TesteCarro
2 {
3    public static void main(String args[])
4    {  // declaração de objeto da subclasse
5       Carro c;
6
7       // uso do construtor da subclasse
8       c = new Carro("Fiesta", "ABC1678", 2006, 2007, 3, 31000);
9
10      //uso de métodos da superclasse
11      c.imprime();
12      c.deprecia(10);
13
14      System.out.println("Carro depreciado");
15      c.imprime();
16   }
17 }
```

```
Veiculo: Fiesta
Placa: ABC1678
Fabr: 2006
Modelo: 2007
03 Portas
R$ 31000,00

Carro depreciado:
Veiculo: Fiesta
Placa: ABC1678
Fabr: 2006
Modelo: 2007
03 Portas
R$ 27900,00
```

Construtor do Carro

- Chama o construtor da classe Veiculo (superclasse) para garantir integridade dos atributos básicos.

```
public Carro(String modelo, String placa, int anoFabr,
             int anoModelo, int numPortas, double valor){
   super (modelo, placa, anoFabr, valor);
   setPortas(numPortas);
   setAnoModelo(anoModelo);
}
```

Método imprime( ) na Classe Carro

```
public void imprime( ){
   System.out.printf("\nVeiculo: %s\nPlaca: %7s", modelo, placa);
   System.out.printf("\nFabr: %4d\nModelo: %4d", anoFabr,
        anoModelo);
   System.out.printf("\n%02d Portas\nR$ %.2f\n", numPortas, valor);
}
```

OU

```
public void imprime( ){
   super.imprime();
   System.out.printf("\n%02d Portas\n", numPortas);
   System.out.printf("\nAno Modelo: %4d", anoModelo);
   }
```

Atividade para casa
Classe Caminhão

- Subclasse Caminhao estende Veiculo
  - Atributos específicos
    - Capacidade
    - Número de eixos
  - Métodos
    - Construtor
    - Sets e gets
    - Impressão dos dados do caminhão

E o que herança tem a ver com polimorfismo?

Definição de Polimorfismo

Princípio pelo várias classes ***derivadas*** *de uma mesma* ***superclasse*** *podem invocar métodos que têm a* ***mesma identificação*** *(assinatura) mas* ***comportamentos distintos****.*

Por exemplo – Classe Veículo

- Atributos básicos
  - Modelo
  - Placa
  - Ano Fabricação
  - Valor
- Métodos básicos
  - Sets e gets
  - Depreciar valor do veículo
  - Impressão dos dados

Quais ou qual método esperamos que tenha exatamente o mesmo comportamento nas classes carro e caminhão?

Qual deve ser alterado nas classes filhas?

Teste - Classe Carro

```
1  public class TesteCarro
2  {
3      public static void main(String args[])
4      {   // declaração de objeto da subclasse
5          Carro c;
6
7          // uso do construtor da subclasse
8          c = new Carro("Fiesta", "ABC1678", 2006, 2007, 3, 31000);
9
10         //uso de métodos da superclasse
11         c.imprime();
12         c.deprecia(10);
13
14         System.out.println("Carro depreciado");
15         c.imprime();
16     }
17 }
```

Já o **imprime**(), foi reescrito na subclasse... Isto é exemplo de polimorfismo!

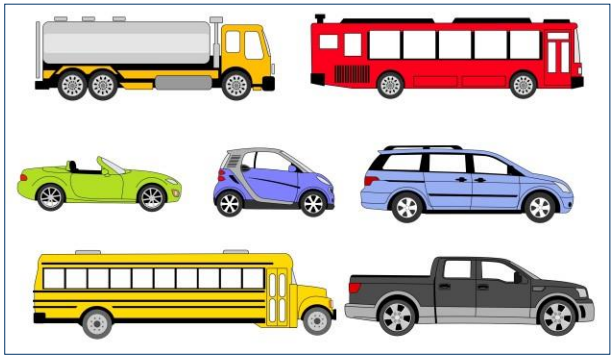
deprecia( ): quando invocado a partir de um objeto Carro, comporta-se exatamente igual a um Veiculo. Método foi herdado.

Reforçando - Polimorfismo

- **Redefinição de um método** da superclasse, ou seja, quando o método é reescrito na subclasse;

- Assim, um **mesmo método** quando invocado por objetos de tipos diferentes apresenta **comportamentos distintos,** apesar de pertencerem a mesma hierarquia (herança).

Polimorfismo



imprime()

imprime()

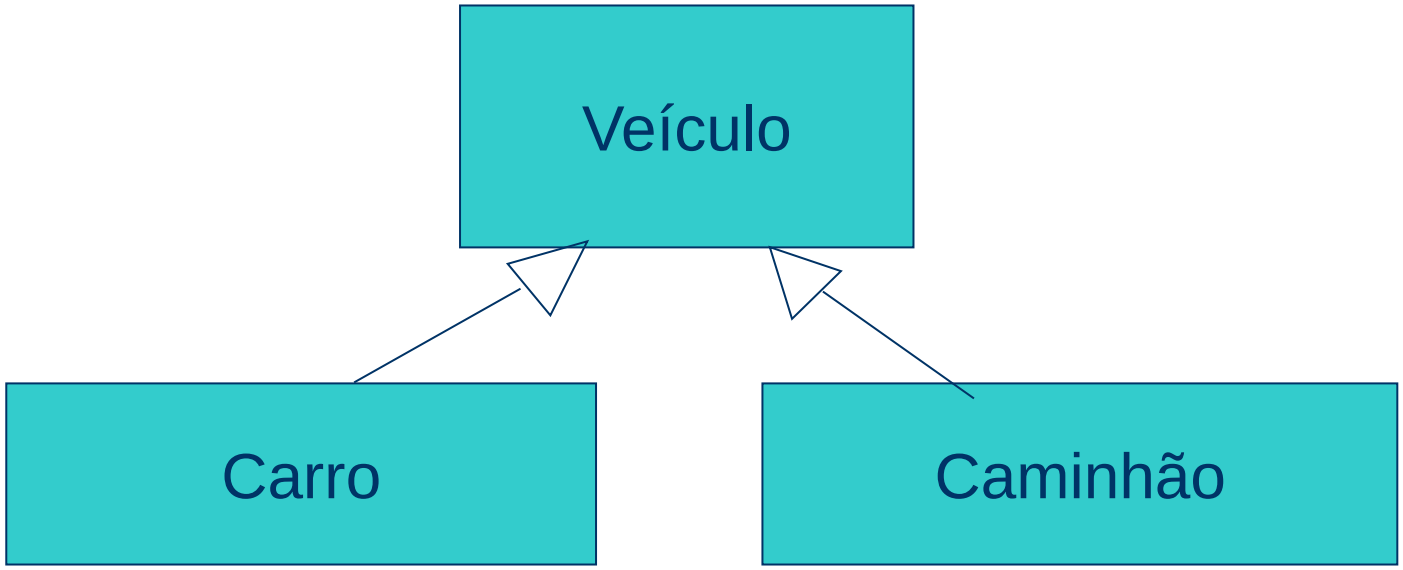
imprime()

- **Redefinição do método imprime()**: provoca diferentes resultados quando o mesmo método é invocado por um objeto do tipo veículo, carro ou caminhão.

Exemplo – Frota de veículos

- Algoritmo
  - Solicita quantidade de veículos (n)
  - Leitura de n veículos:
    - Solicita tipo de veiculo (“v”, “a”, “c”)
    - Solicita dados de acordo com o tipo
    - Armazena em vetor de tamanho **n**
  - Imprime dados de **n** veículos armazenados em cada posição do vetor (polimorfismo)
- Qual o tipo do vetor?
  - Veículo? Carro? Caminhão?

Hierarquia – Veículo



- Podemos afirmar que:
  - Todo Carro “é um” Veículo, mas nem todo Veículo é um carro.
  - Todo Caminhão “é um” Veículo.

Vetor de Veículos

- Como um vetor preparado para armazenar dados de Veiculos consegue guardar dados do tipo Carro ou Caminhão, se eles são “maiores” que veículo??? Ou seja, tem mais atributos e ocupam mais memória?
- Cada posição do vetor guarda a referência para o objeto instanciado em memória (endereço);
- Diferentemente dos tipos primitivos, variáveis associadas a objetos não guardam os objetos, mas sim sua referência (endereçamento em memória).

Atividade - Classe Frota (parte 1 de 3)

```
1 import java.util.Scanner;
2 public class Frota
3 {
4    public static void main(String args[])
5    {
6       Scanner sInput = new Scanner(System.in);
7       Scanner nInput = new Scanner(System.in);
8
9       int tam, i, anoFabr, anoModelo, numPortas, numEixos;
10      double capacidade, valor;
11      String tipo, modelo, placa;
12
13      System.out.println("Digite o tamanho da frota: ");
14      tam = nInput.nextInt();
15      Veiculo frota[] = new Veiculo[tam];
16
17      for(i = 0; i<tam; i++)
18      {
19         System.out.println("Tipo do "+ (i+1) +".o veículo (a - auto/ c - caminhao): ");
20         tipo = sInput.nextLine();
```

Usa classe Scanner
Início da classe Frota

Declaração de Variáveis

Tamanho da Frota

Loop para entrada de dados

Classe Frota (parte 2 de 3)

```
22      if (!tipo.equalsIgnoreCase("a") && !tipo.equalsIgnoreCase("c"))
23              System.out.println("TIPO NÃO PREVISTO");
24
25      System.out.println("Modelo: ");
26      modelo = sInput.nextLine();
27      System.out.println("Placa: ");
28      placa = sInput.nextLine();
29      System.out.println("Ano Fabr.: ");
30      anoFabr = nInput.nextInt();
31      System.out.println("Valor: ");
32      valor = nInput.nextDouble();
33
34      if ( tipo.equalsIgnoreCase("a") )
35      {
36         System.out.println("Ano Modelo: ");
37         anoModelo = nInput.nextInt();
38
39         System.out.println("Portas: ");
40         numPortas = nInput.nextInt();
41
42         frota[i] = new Carro(modelo, placa, anoFabr, anoModelo, numPortas, valor);
43      }
```

Dados comuns a qualquer veiculo

Dados objeto Carro

Classe Frota (parte 3 de 3)

```
44          else if ( tipo.equalsIgnoreCase("c") )
45                {
46                   System.out.println("Capacidade (toneladas): ");
47                   capacidade = nInput.nextDouble();
48                   System.out.println("Num de Eixos: ");
49                   numEixos = nInput.nextInt();
50                   frota[i] = new Caminhao(modelo, placa, anoFabr,
51                                           capacidade, numEixos, valor);
52                }
53                else
54                {
55                   frota[i] = new Veiculo(modelo, placa, anoFabr, valor);
56                }
57       }
58       for(i = 0; i<tam; i++)
59       {
60          System.out.println("Veiculo n.o " + (i+1));
61          if (frota[i]!= null)
62             frota[i].imprime();
63          System.out.println();
64       }
65    }
66 }
```

Caminhão

Tipo genérico: veículo

Polimorfismo
O que vai aparecer
no imprime???

Ligação tardia

- O método pode ser invocado a partir de uma **referência** a um objeto do tipo da **superclasse**, apesar de, na prática, ser uma **instância da subclasse**.
- Neste caso, a **decisão** sobre qual o método que deve ser selecionado, de acordo com o tipo da classe derivada, é tomada em **tempo de execução**, através do mecanismo de **ligação tardia**.
  - Ex: Classe Frota. Diferentemente da classe TesteCarro (não possui ligação tardia).

Atividades para enviar pelo Teams

- Frota de veículos