

# Orientação a Objetos

- Roteiro
  - Paradigma de construção de software
  - Orientação a Objetos
  - Conceitos de Orientação a Objetos (OO)
  - Classe, Objeto e Mensagem
  - Os pilares da Orientação a Objetos (OO)
  - Reuso de Implementação

# Orientação a Objetos

- **Paradigma**
- Paradigma é a filosofia adotada na construção de software:
  - **Imperativo ou Procedural** (C, Fortran, etc...);
  - **Lógico** (Prolog, etc...);
  - **Funcional** (Lisp, OCAML, etc...);
  - **Orientado a Objetos** (Java, C++, SmallTalk, etc... );
  - **Orientado a Aspectos** (AspectJ, AspectC++, etc...).

# Orientação a Objetos

- **Paradigma Orientado a Objetos**
  - Sugere a **diminuição** da distância entre a **modelagem computacional** e o **mundo real**.
  - Surgiu na tentativa de solucionar **problemas complexos** existentes através do desenvolvimento de softwares **menos complexos, confiáveis e com baixo custo** de desenvolvimento e manutenção.

# Orientação a Objetos

- **Paradigma Orientado a Objetos**
  - Permite que **objetos** do **mundo real** sejam mapeados em Objetos no computador, pressupondo que o mundo é **composto por objetos**.
  - Os **sistemas** são modelados como um **conjunto de objetos** que **interagem entre si**.

# Orientação a Objetos

- **Por que programar Orientado a Objetos?**
  - Permite **alta reutilização de código**;
  - **Reduz tempo de manutenção** de código;
  - **Reduz complexidade** através da melhoria do grau de abstração do sistema;
  - **Aumenta qualidade e produtividade** → oferece maiores facilidades ao desenvolvedor;
  - Adoção (aceitação) comercial **crescente**.

# Orientação a Objetos

- **Armadilhas da Orientação a Objetos**
  - Pensar no paradigma OO simplesmente como **uma linguagem**;
  - Aversão a reutilização;
  - Pensar na OO como uma **solução para tudo**;
  - Programação Egoísta:
    - É preciso documentar!

# Orientação a Objetos

- **Orientação a Objetos (OO)**

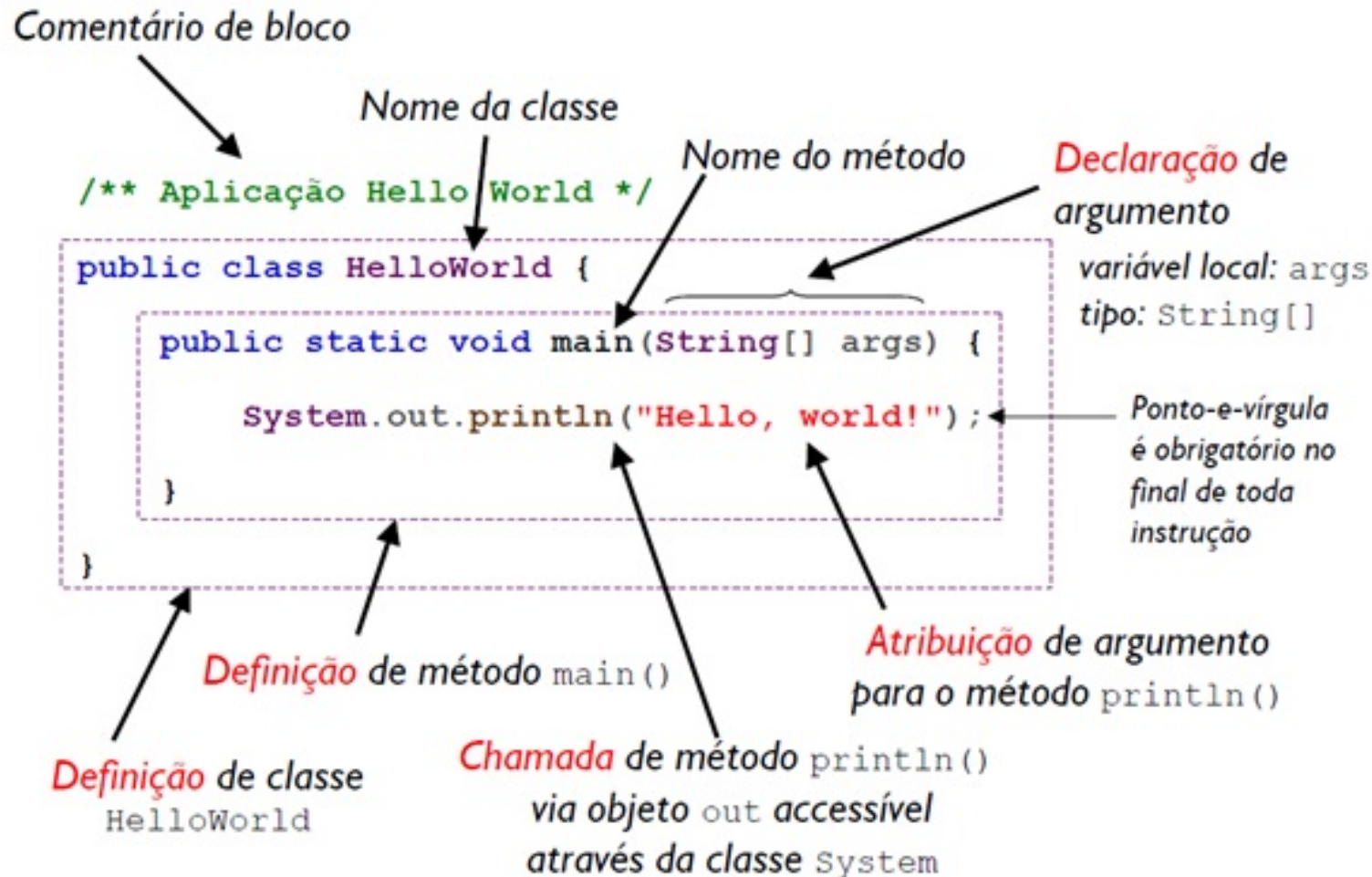
- É uma forma de **entender e representar sistemas complexos** como estruturas hierárquicas de objetos que se **relacionam**.

# Orientação a Objetos

- **Conceitos da Orientação a Objetos**
  - Classe
  - Objeto (Instância)
  - Mensagem
  - Encapsulamento
  - Herança
  - Polimorfismo
  - ...



# Orientação a Objetos



# Orientação a Objetos

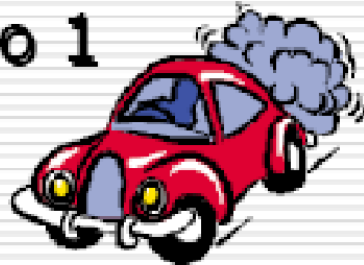
- **Classe**

- A classe é a implementação de tipo abstrato de dados (TAD) no paradigma orientado a objetos.
- Uma classe Java é um molde para a criação de objetos. A classe define as propriedades (atributos) e os comportamentos (métodos).
- Além disso, uma classe Java define como produzir (instanciar) objetos a partir dela.

# Orientação a Objetos

- **Classe**

**Objeto 1**



**Objeto 2**



**Classe**  
**Automovel**

numeroPortas  
cor  
fabricante  
ano  
placa

# Orientação a Objetos

- Classe



# Orientação a Objetos

- **Classe**

```
1 package aula8;
2
3 public class Alunos {
4
5     private String nome=" ";
6
7
8     public void setNome(String nome) {
9         this.nome = nome;
10    }
11
12    public String getNome() {
13        return nome;
14    }
15 }
```

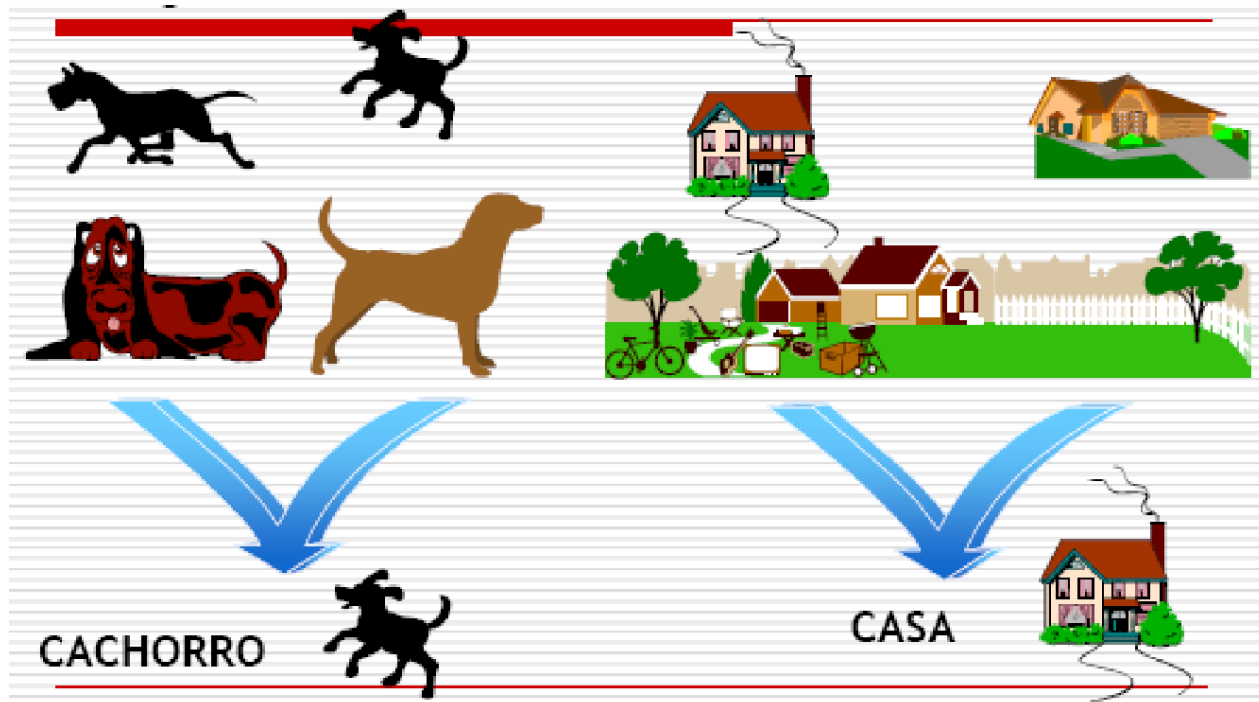
# Orientação a Objetos

- **Objeto**

- Um objeto é uma construção de software que encapsula **estado** e **comportamento**, através respectivamente de **propriedades** (atributos) e **operações** (métodos);
- **Estado de um Objeto**: composto por suas **propriedades** e seus respectivos **valores**;
- **Comportamento**: a maneira como o objeto reage quando o seu estado é **alterado** ou quando uma mensagem é **recebida**.

# Orientação a Objetos

- Objeto



# Orientação a Objetos

- Objeto

☐ Um objeto possui operações:



- ☐ Ligar;
- ☐ Desligar;
- ☐ Travar portas;
- ☐ Acelerar;
- ☐ Frear...



# Orientação a Objetos

- Objeto

```
1 package br.com.xstream.teste;
2
3 public class Carro {
4     private String marca;
5     private String ano;
6     private Motor motor;
7
8     public String getMarca() {..
11    public void setMarca(String marca) {..
14    public String getAno() {..
17    public void setAno(String ano) {..
20    public Motor getMotor() {..
23    public void setMotor(Motor motor) {..
26 }
```

# Orientação a Objetos

- **Objeto / Instancia**

**Carro car = new Carro();**

**car.marca = “Fiat”;**

**car.ano = “2011”;**

**car.LigarMotor();**

**car.DesligarMotor();**

# Orientação a Objetos

- **Mensagens**

- Mecanismo através do qual os objetos se **comunicam**, invocando as **operações desejadas**;
- Um objeto (**Emissor**) envia uma mensagem a outro (**Receptor**) que executará uma tarefa.

# Orientação a Objetos

- **Os pilares da OO**

- Os pilares da OO são mecanismos fundamentais que garantem a filosofia de Orientação a Objetos. São eles:

- Encapsulamento;
    - Herança;
    - Polimorfismo.

# Orientação a Objetos

- **Os pilares da OO**

- Os pilares da OO são mecanismos fundamentais que garantem a filosofia de Orientação a Objetos. São eles:

- Encapsulamento;
    - Herança;
    - Polimorfismo.

# Orientação a Objetos

- **Encapsulamento**

- **Resumindo:** “Não mostre as cartas de seu baralho”
- **Objetivos:**
  - Ocultar do mundo externo ao objeto os detalhes de implementação e restringir o acesso às propriedades e aos métodos;
  - Permitir a criação de programas com **menos erros e mais clareza.**

# Orientação a Objetos

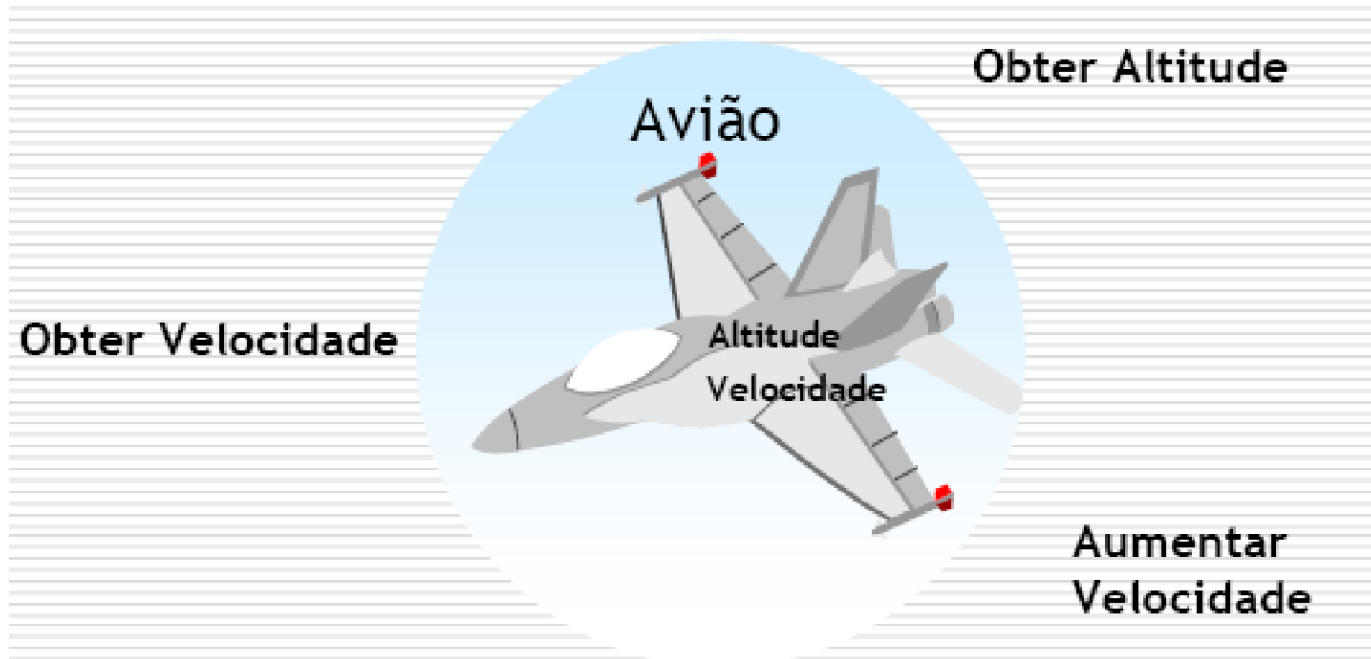
- **Encapsulamento**

- Vantagens:

- Segurança no acesso ao objeto;
    - Melhor consistência no estado interno, pois evita **alterações incorretas** nos valores das propriedades.

# Orientação a Objetos

- **Encapsulamento**





# Orientação a Objetos

- Encapsulamento

```
1  package javaapplication7;
2
3  public class Exemplo {
4      private String texto;
5      Exemplo(String texto){
6          this.texto = texto;
7      }
8
9      public void setTexto(String texto){
10         this.texto = texto;
11     }
12
13     public String getTexto(){
14         return this.texto;
15     }
16 }
```

# Orientação a Objetos

- **Herança**

- **Resumindo:** “Filho de peixe, peixe é”.
- Permite definir **novas classes** (subclasses) a partir de uma classe já existente (superclasse).
- A subclasse herda as **propriedades comuns** da superclasse e pode ainda **adicionar novos métodos** ou **reescrever métodos** herdados.

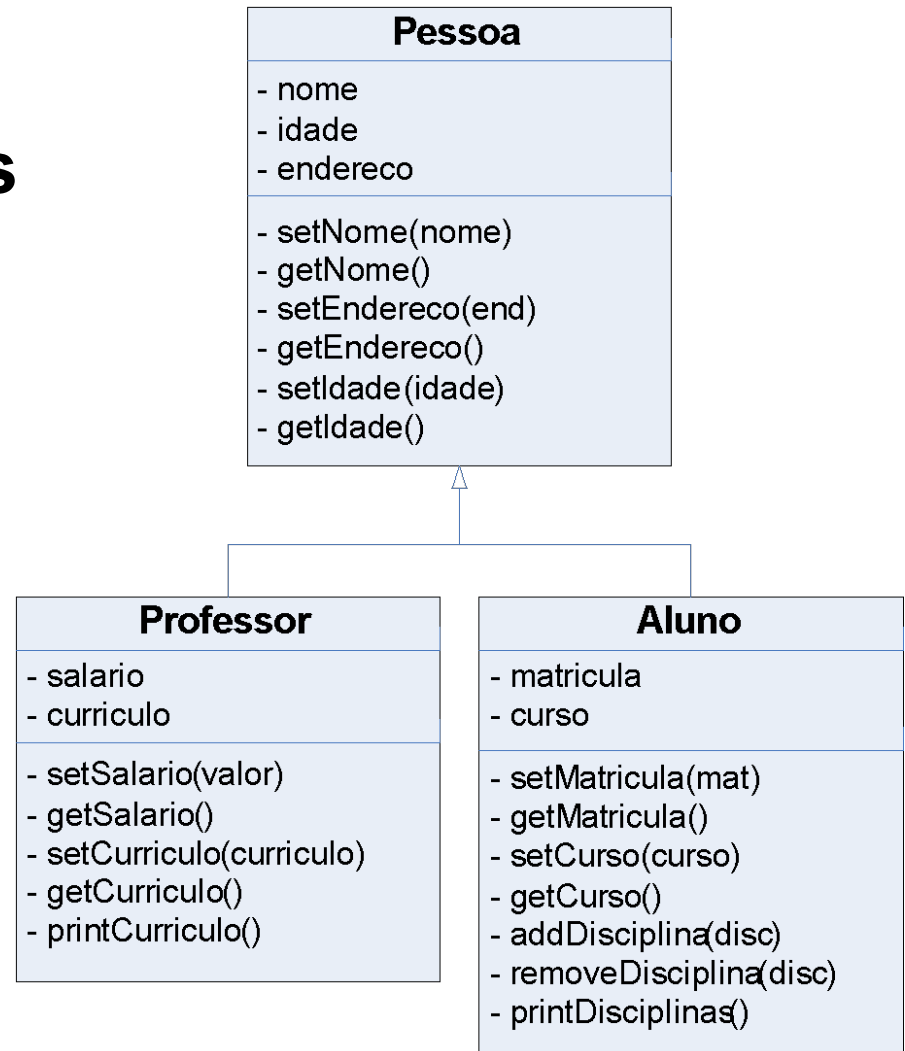
# Orientação a Objetos

- **Herança**

- **Objetivo:** evitar que classes que possuam atributos ou métodos **semelhantes** sejam **repetidamente criados**.

# Orientação a Objetos

- **Herança**
  - Pode ser: **Simples**



# Orientação a Objetos

```
class Pessoa {  
    String      nome;  
    int         idade;  
  
    void definirNome(String valor) {  
        nome = valor;  
    }  
  
    String retornarNome() {  
        return nome;  
    }  
  
    void definirIdade(int valor) {  
        idade = valor;  
    }  
  
    int retornarIdade() {  
        return idade;  
    }  
}
```

```
class Aluno extends Pessoa {  
    String      curso;  
  
    void definirCurso(String valor) {  
        curso = valor;  
    }  
  
    String retornarCurso() {  
        return curso;  
    }  
}
```

# Orientação a Objetos

- **Polimorfismo**

- **Resumindo:** “Vamos nos adaptar”.
- Permite a um método ter **várias implementações** as quais são selecionadas com base na **quantidade de parâmetros e seus tipos** que é passado para a invocação do método.

Janela ( )



Janela ( 1 x 2 , 2 )



Janela ( 1 x 2 , 2, Azul )



# Orientação a Objetos

- **Tipos de Polimorfismo**

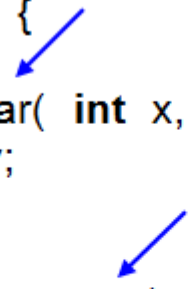
- São dois os tipos de Polimorfismo:

- Sobrescrita ou Redefinição de métodos (**Override**);
    - Sobrecarga de métodos (**Overload**).

# Orientação a Objetos

- **Exemplo de Sobrecarga (Overload)**
  - Permite a existência de vários métodos de mesmo **nome**, porém com **assinaturas** levemente diferentes (número, tipo e qtd de parâmetros).

```
public class Soma {  
    public int somar( int x, int y) {  
        return x+y;  
    }  
  
    public double somar( double x, double y) {  
        return x+y;  
    }  
}
```





# Orientação a Objetos

- Exemplo de Sobrescrita (Override)
  - Permite a existência de vários métodos com **assinaturas idênticas**, porém com implementações distintas.

# Orientação a Objetos

- **Reuso de Implementação**
  - Separação **interface-implementação** → **maior reuso**
    - Reuso depende de **bom planejamento** durante a etapa de **modelagem**.
  - Uma vez criada uma classe, ela deve representar uma **unidade de código útil** para que seja **reutilizável**.

# Orientação a Objetos

- **Formas de uso e reuso de classes**

- Uso e reuso de **objetos** criados pela classe: **mais flexível**

- **Composição:** a “é parte essencial de” b 

- **Agregação:** a “é parte de” b 

- **Associação:** a “é usado por” b 

- Reuso da interface da classe: **pouco flexível**

- **Herança:** b “é um tipo de” a (substituição útil, extensão)

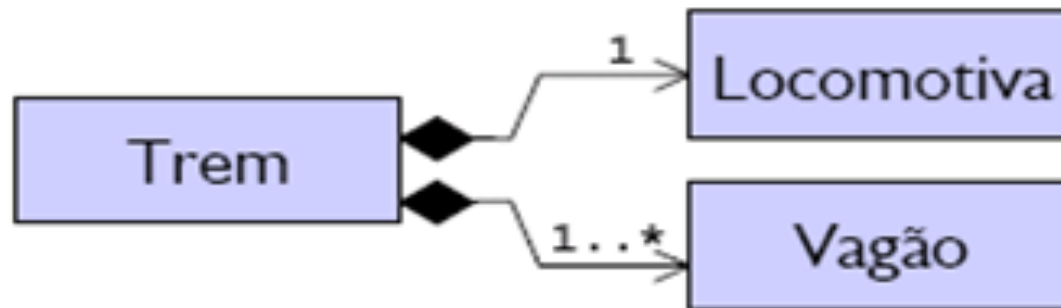


# Orientação a Objetos

- **Exemplo:**
  - Crie classes com atributos e métodos para:
    - Automóvel;
    - Pássaro;
    - Computador;
    - Animal;
    - Pessoa;

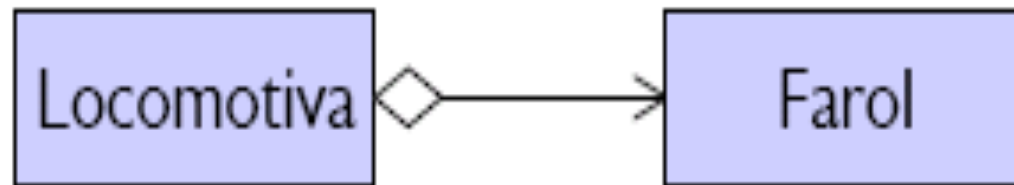
# Orientação a Objetos

- **Composição, Agregação e Associação**
  - **Composição** (tem-um): um trem é formado por locomotiva e vagões.



# Orientação a Objetos

- **Composição, Agregação e Associação**
  - **Agregação:** uma locomotiva tem farol (mas não vai deixar de ser uma locomotiva se não o tiver).



# Orientação a Objetos

- **Composição, Agregação e Associação**
  - **Associação:** um trem usa uma estrada de ferro (não faz parte do trem, mas ele depende dela).

