
UNIVERSIDADE DO SUL DE SANTA CATARINA

Aqui o seu futuro acontece

# Tópicos avançados em Programação

Revisão Orientação à objetos

http://dl.dropbox.com/u/3025380/prog2/aula1.pdf

flavio.ceci@unisul.br

# Orientação a objetos

- O **objeto** é uma **abstração** de conjunto de **coisas do mundo** real. Pode ser uma **máquina**, uma **organização**, um **carro**, uma **passagem de ônibus** ou **negócio**;
- A OO **modela objetos** do mundo real, estudando e criando **classes** a partir de suas **características** como cor, nome, tamanho, etc. (Lima, 2005).

# Orientação a objetos

- ***Domínio da Aplicação*** – modelo obtido dentro de um contexto de negócio após o estudo e observação da realidade.

- ***Abstração*** – processo de separar mentalmente os objetos observados e estudados da realidade.

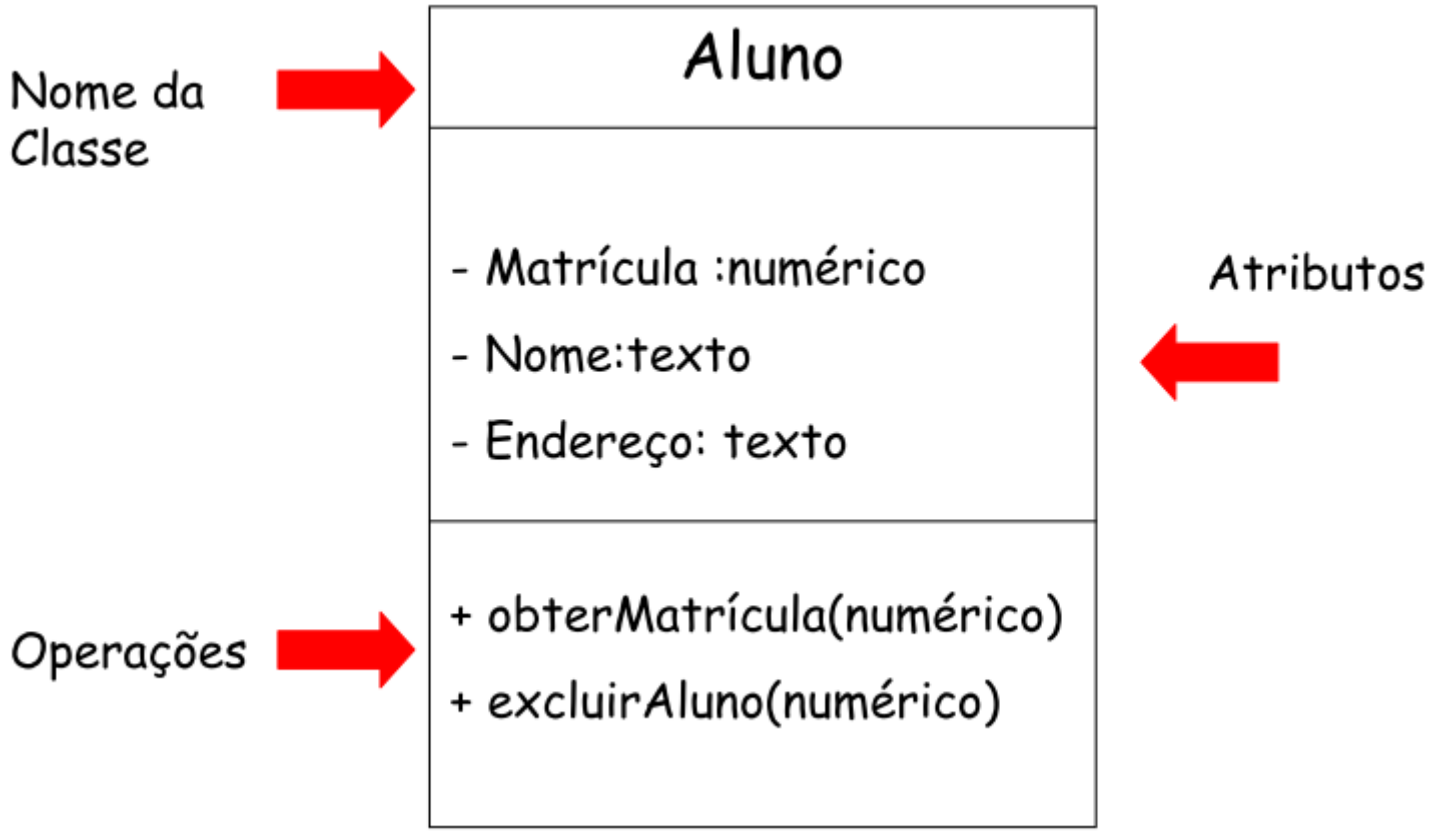
# Orientação a objetos

- ***Classe*** – pode ser vista como a **descrição de um tipo de objeto**, com propriedades semelhantes (atributos), o mesmo comportamento (operações), os mesmos relacionamentos com outros objetos e a mesma semântica.

- A classe agrupa objetos com características e comportamentos comuns!

# Orientação a objetos

- Por exemplo: a classe aluno de uma escola, apresenta um conjunto de alunos que apresentam as mesmas informações.



# Orientação a objetos

- **Instância**:
  - A classe por sua vez deve descrever as propriedades e comportamentos daquele objeto. Uma classe descreve um grupo de objetos;
  - Cada objeto do mundo real pertencente a uma classe é denominado instância da classe;
  - João Filomeno é então uma instância da classe Aluno.

# Orientação a objetos

- **Atributos**:
  - O atributo é a descrição dos dados armazenados pelos objetos de uma classe;
  - O atributo de uma classe está associado a um conjunto de valores que o atributo pode assumir;

# Orientação a objetos

- **Atributos**:
  - Atributos não tem comportamento. Cada valor de um atributo é particular para um dado objeto;
  - Uma classe pode ter qualquer número de atributos ou mesmo nenhum atributo.

# Orientação a objeto

- São Atributos para a classe Aluno:
  - Matricula;
  - Nome;
  - Curso;
  - Idade;
  - Sexo...
- Os atributos **são sempre individuais** e cada objeto da classe possui seus próprios atributos.

# Orientação a objetos

- **Operações**:
  - As operações implementam serviços que podem ser solicitados por algum objeto da classe para modificar o comportamento.
  - Todos os objetos da classe vão compartilhar destas operações.

# Visibilidade

# Visibilidade

- Publico (+) – Qualquer classificador externo com visibilidade para que determinado classificador seja capaz de usar a característica.
- Protegido (#) – Qualquer descendente do classificador é capaz de usar a característica;
- Privado (-) – Somente o próprio classificador é capaz de usar a característica;
- Pacote (~) – Somente classificadores declarados no mesmo pacote podem usar a característica.

# Encapsulamento e Método construtor

# Encapsulamento

- No paradigma OO, **o acesso aos dados dos objetos** deve ocorrer **somente pelos métodos dos próprios objetos** (membro de dados privados);

- Garantindo a **manipulação adequada** dos mesmos e utilização das regras de negócio implementadas;

# Encapsulamento

- Também é conhecido com **ocultação de informação**;
- Deste modo os objetos definem **quais serviços** estão **acessíveis** a outros **objetos**;
- Um padrão proposto pele mercado é associado a utilização de métodos de acesso:
  - setters;
  - getters.

# Métodos **setter**

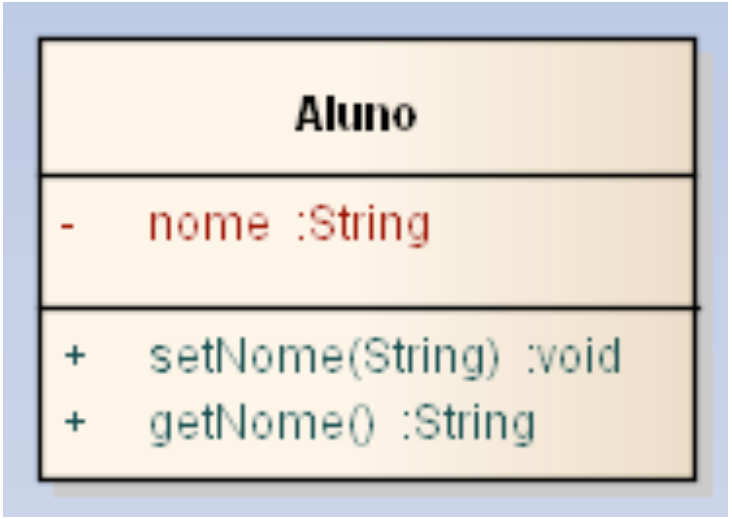
- Todos os métodos utilizados para alterar/ inserir valores nos membros de dados;
- No padrão proposto, esses métodos deve começar com a palavra set seguida do nome da propriedade.
- Por exemplos:
  - Atributo: int idade;
  - Método: setIdade(int);

# Métodos **getter**

- Todos os métodos utilizados para recuperar valores encapsulados nos objetos;
- No padrão proposto, esses métodos devem começar com a palavra get, seguida pelo nome da propriedade recuperada;
- Por exemplo:
  - Atributo: int idade;
  - Método: getIdade(): int;

# Exemplo

```
public class Aluno {

    private String nome;

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getNome() {
        return nome;
    }
}
```



# Vantagens de uso

- Garantir que as **regras de negócio** relacionadas aos dados **sejam realmente utilizadas**;
  - Impedindo que alterações inapropriadas nos atributos sejam realizadas;

- Garantir o correto acesso às informações.

# Métodos construtores

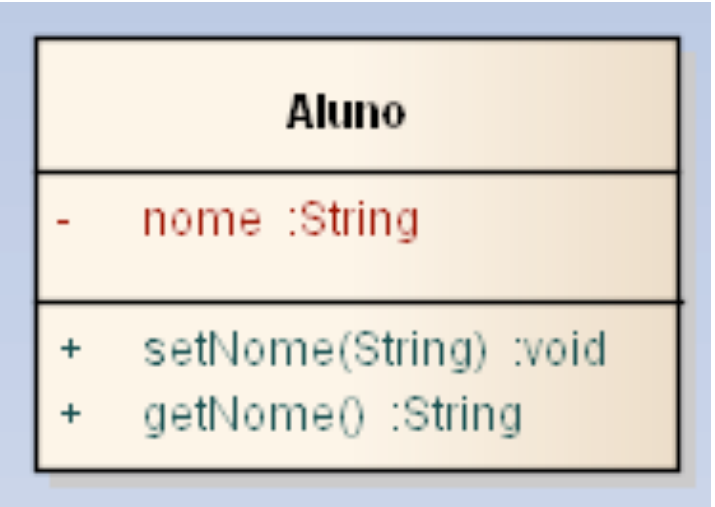
# Métodos construtores

- Os métodos construtores, como o próprio nome sugere, tem como finalidade a construção de objetos;
- Pode-se pensar como um **contrato** para a **construção do objeto**;
- O método construtor pode ser de duas formas:
  - Com argumento; e
  - Sem argumento.

# Construtor **sem** argumento

- Um construtor sem argumento sempre é criado em uma classe;
- Ele inicia todos os atributos de classe com os seus valores default (nulo ou zero);
- No exemplo abaixo o atributo nome será nulo.

```
Aluno aluno = new Aluno();
```



# Construtor **com** argumento

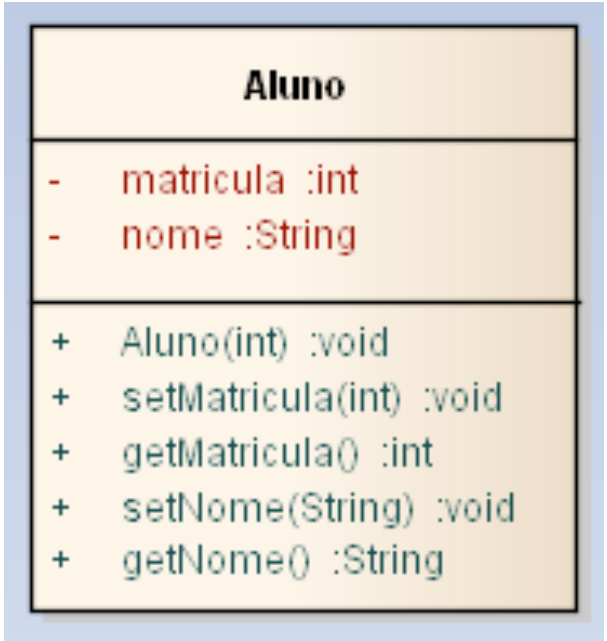
- A possibilidade da **utilização** de um **método construtor com argumento**, traz um **segurança** para o desenvolvedor que concebe o objeto;
- Quando um objeto possui um construtor com argumento **ele está dizendo** a seguinte coisa para quem for manipular esse objeto:
  - Para utilizar esse objeto precisas preencher os seguintes atributos, atributos esses que estão sendo pedidos como parâmetros no construtor.

# Construtor **com** argumento

```
public class Aluno {

    private int matricula;

    private String nome;

    public Aluno(int matricula) {
        this.setMatricula(matricula);
    }

    public void setMatricula(int matricula) {
        this.matricula;
    }

    public int getMatricula() {
        return matricula;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getNome() {
        return nome;
    }
}
```

Criação do objeto Aluno:

```
Aluno aluno = new Aluno(98413);
```



# Exercício 1

# Exercício 1

- A partir do cenário apresentado a seguir, modele as classes principais do sistema solicitado.
- Nesse momento não se preocupe com a modelagem dos relacionamentos entre as classes.
  - Identifique:
    - Classes;
    - Atributos; e
    - Métodos.

# Exercício 1

- A Unisul solicitou para os alunos do curso de Sistemas de Informação da Unisul, a criação de um sistema para controle e cadastro dos seus alunos.

- Cada aluno pode ter vinculo apenas com um curso.

# Exercício 1

- Requisitos do sistema:
  - O sistema deve permitir o cadastro de alunos;
  - O sistema deve permitir o cadastro de cursos;
  - Todo aluno deve ter vinculo com um curso;
  - O sistema deve armazenar essas informações em memória.

# Exercício 1

- Características do Aluno de Graduação:
  - Nome;
  - Matricula;
  - Curso;
  - Tipo (Graduação ou Pós);
  - Telefone.

# Exercício 1

- Características do Curso:
  - Nome do curso;
  - Nível (Graduação; Especialização; Mestrado; Doutorado);
  - Unidade.

# Exercício 1

- O sistema deve possuir um menu para que se possa interagir com as operações;
- As operações suportadas são:
  - Localizar um aluno;
  - Localizar um aluno por tipo(Graduação ou Pós)
  - Listar alunos (apresenta as informações do curso associado);
  - Listar alunos por curso (primeiramente deve-se busca um curso por nome, depois apresentar os alunos vinculados);
  - Excluir um aluno;

# Exercício 1

- As operações suportadas são:
  - Excluir um curso (e todos os alunos vinculados ao curso em questão);
    - Exclusão por nome;
  - Incluir um aluno;
  - Incluir um curso;
    - Deve-se verificar se o curso já existe ou não (utilizando como base o seu nome);
  - Vincular um curso à um aluno.
    - Antes de cadastrar um novo curso deve-se verificar se ele já não existe.

# Exercício 1

- Desenhe sua proposta de solução:

# Relacionamento entre Objetos

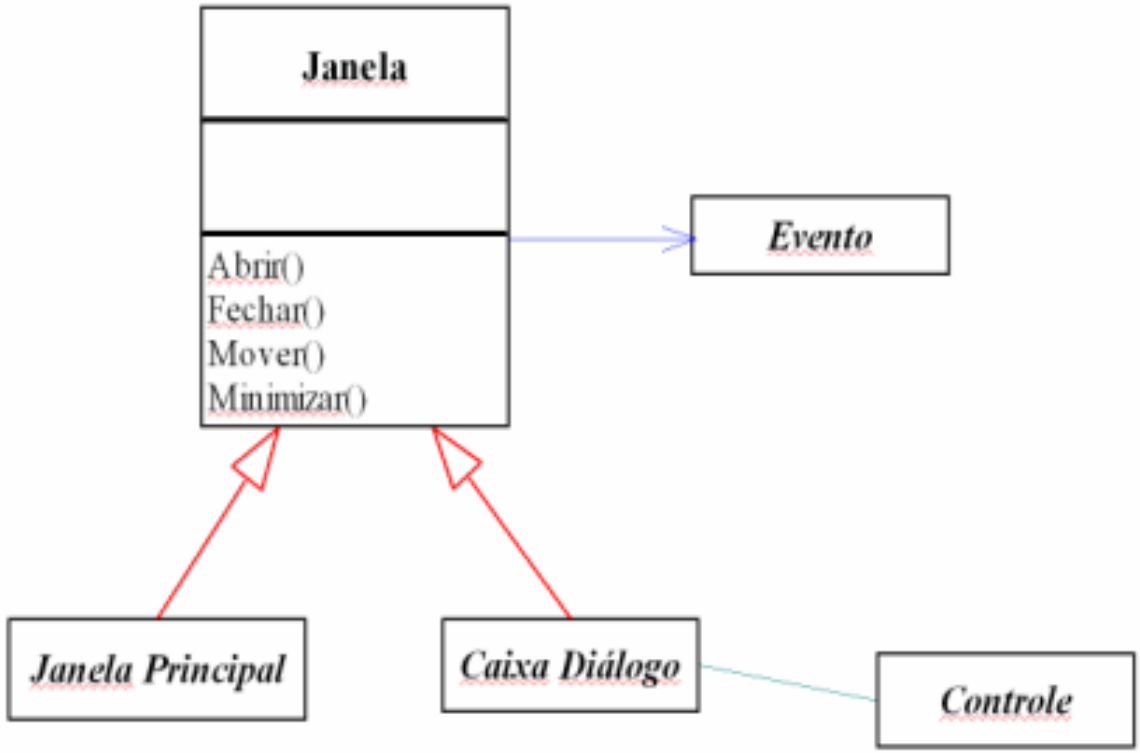
# Relacionamento entre Objetos

- Um relacionamento representa a interação entre as classes e objetos, eles apóiam o refinamento das classes.

- Existem diferentes tipos de relacionamentos possíveis entre as classes identificadas, três deles são os mais importantes as associações, as dependências e as generalizações.

# Relacionamento entre Objetos

- dependência
- generalização
- associação



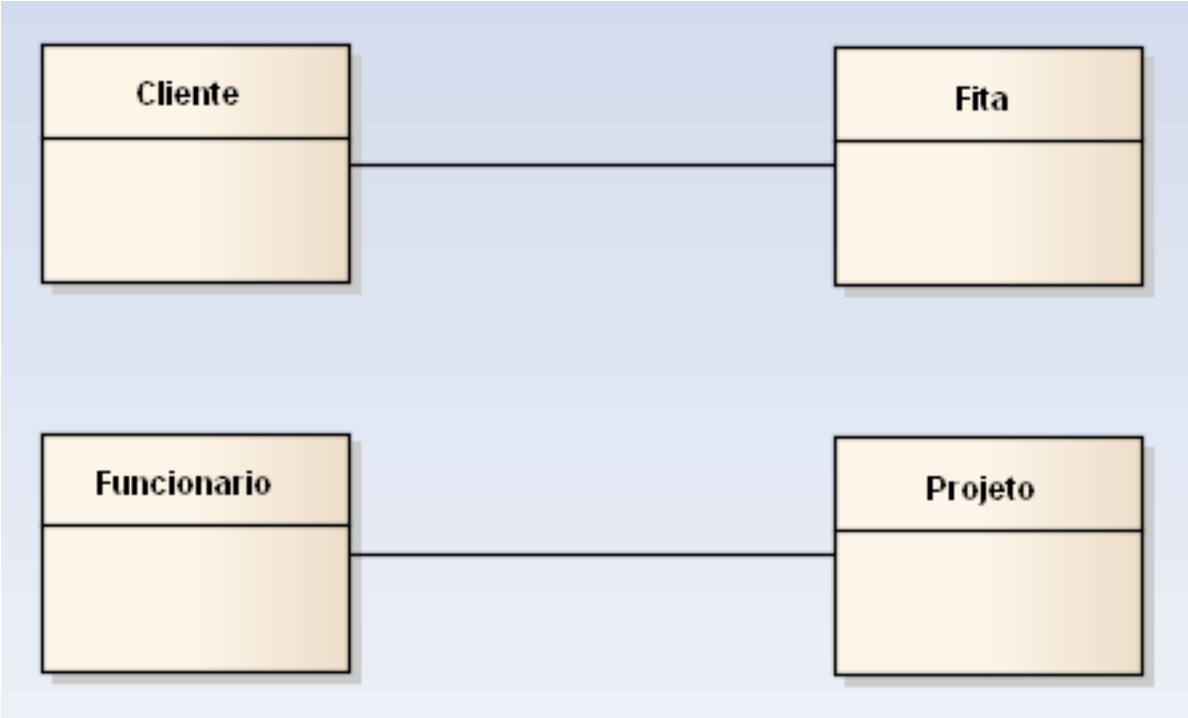
# Relacionamento de Associação

- **Associação** é uma relação que descreve um **conjunto de vínculos** entre elementos de modelo.

- Quando uma determinada **instância** de uma das classes **origina ou se associa** a **uma ou mais instâncias da outra classe** você pode dizer que temos um relacionamento de **associação** (Furlan, 1992).

# Relacionamento de Associação

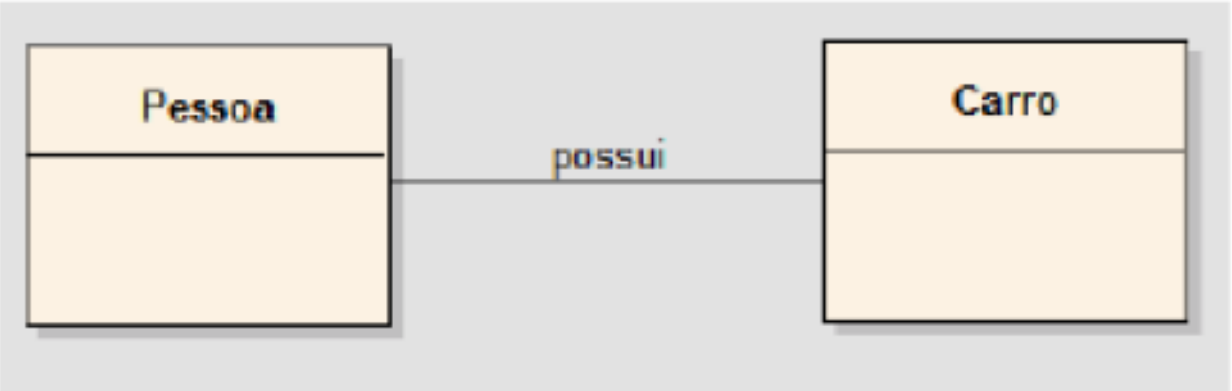
Veja: no domínio de uma vídeo locadora o cliente loca fitas, este é um relacionamento de associação. A associação entre Funcionário e Projetos significa que os **objetos da classe *Funcionário*** estão **conectados** aos **objetos** da classe ***Projeto***.



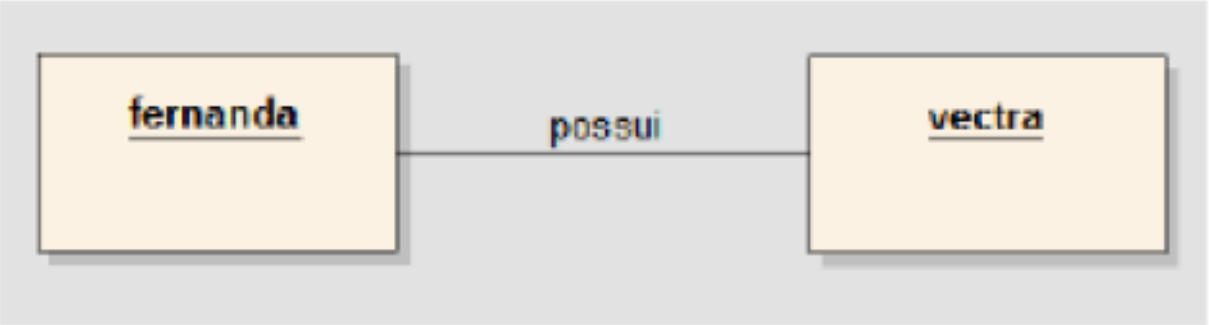
# Associações e Links

- Associação:
  - Refere-se a um relacionamento representado por uma linha num Diagrama de Classes.
  - A linha pode ter um nome lógico que descreve o relacionamento.
- Link:
  - Refere-se a um relacionamento entre dois objetos mostrado num diagrama de objetos.

# Associações e Links




Uma Associação entre duas classes.




Um Link entre dois objetos.

# Multiplicidade

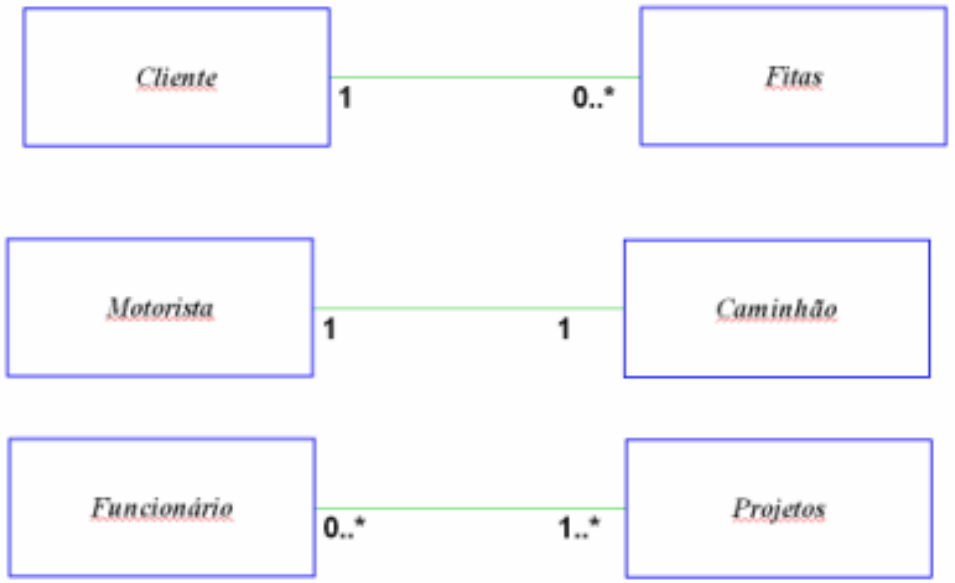
- Quando você fala em associação é possível **representar a quantidade de objetos** aos quais o outro objeto esta associado.
- Um exemplo prático:
  - Um projeto existe sem que seja alocado um funcionário para este projeto?
  - Quantos projetos podem ser alocados para cada funcionário?
  - Quantos funcionários podem ser alocados para cada projeto?

# Multiplicidade

| Nome | Simbologia |
|---|---|
| Apenas um | 1 |
| Zero ou muitos | 0...* |
| Um ou muitos | 1...* |
| Zero ou um | 0...1 |
| Intervalo específico | $1_i..1_s$ |

# Exemplos de multiplicidade

- Um cliente pode alocar nenhuma ou várias fitas, mas uma fita pode estar locada por apenas um cliente.
- Em uma empresa de transporte um motorista dirige apenas um caminhão, e cada caminhão pode ser dirigido por apenas um motorista.
- No terceiro exemplo um funcionário pode estar alocado a vários projetos, por outro lado um projeto possui vários funcionários.



# Relacionamento de Agregação

# Agregação

- A agregação é um caso particular da associação.
- A agregação indica que uma das classes do relacionamento **é uma parte**, ou está **contida** em outra classe.
- Mas as duas classes estão no **mesmo nível**, ou seja não existe uma classe mais importante do que a outra

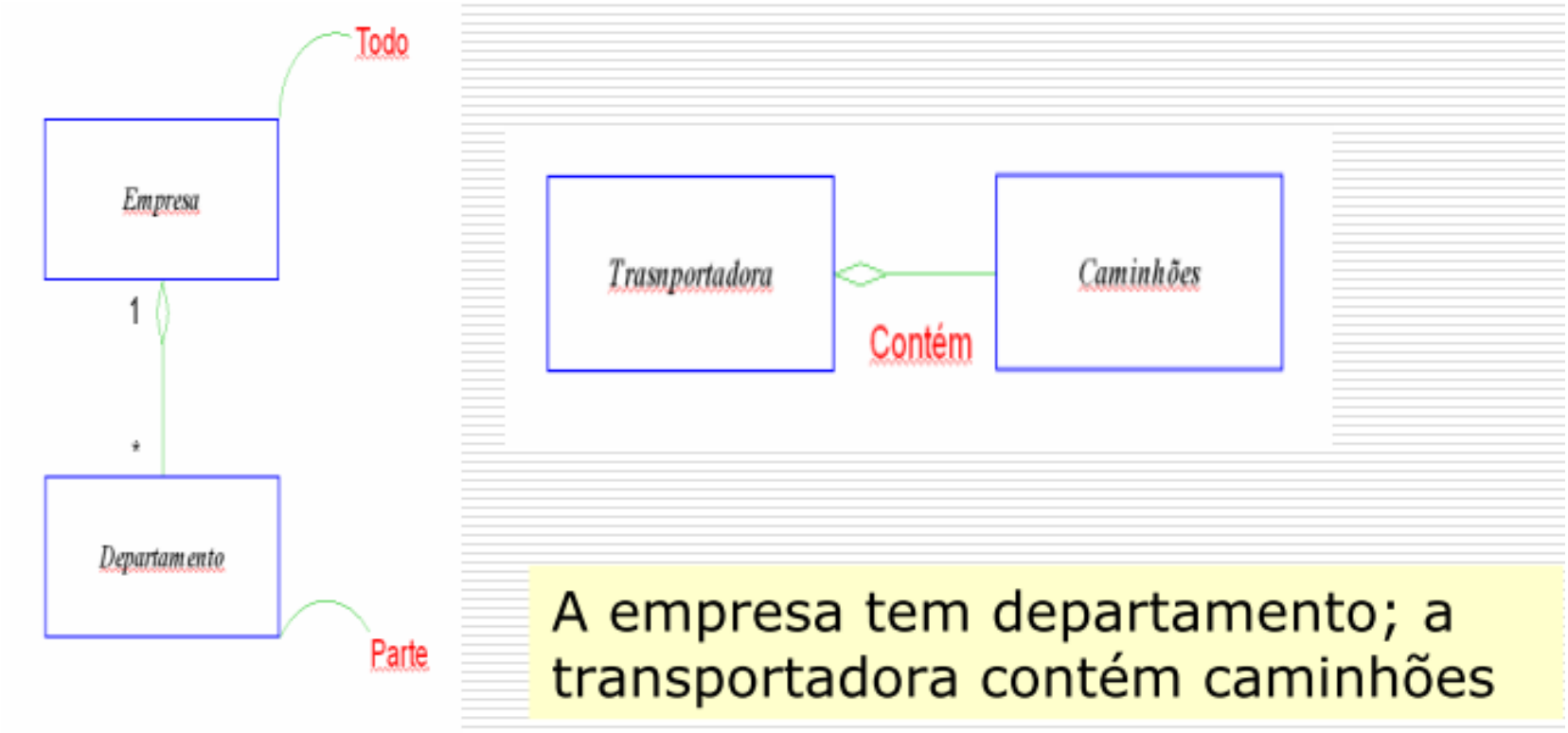
# Agregação

- As palavras chaves usadas para identificar uma agregação são: "***consiste em***", "***contém***" ou "***é parte de***".

- Outra dica importante é que **as partes não morrem** obrigatoriamente **com o todo,** e uma mesma parte pode estar em mais de um "todo".

# Agregação

- Graficamente você vai representar a associação de agregação por uma linha e um diamante aberto na extremidade.



# Relacionamento de Composição

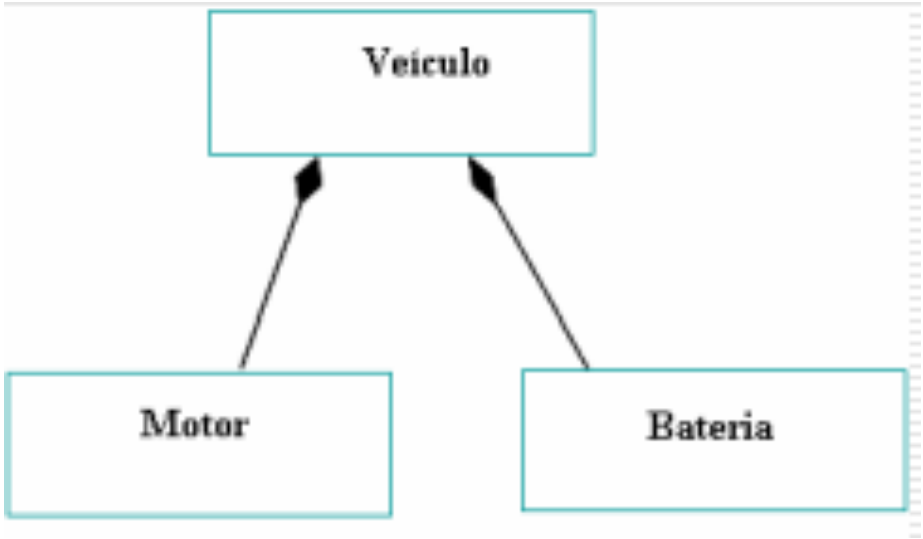
# Composição

- É um tipo especial de agregação onde a multiplicidade do lado "***todo***" **é sempre 1**.
- As partes **vivem** e **morrem** obrigatoriamente **com o todo**.
- Uma mesma parte não pode estar em mais de um "todo".
- Os objetos da classe parte não existem de forma independente da classe todo.

# Composição

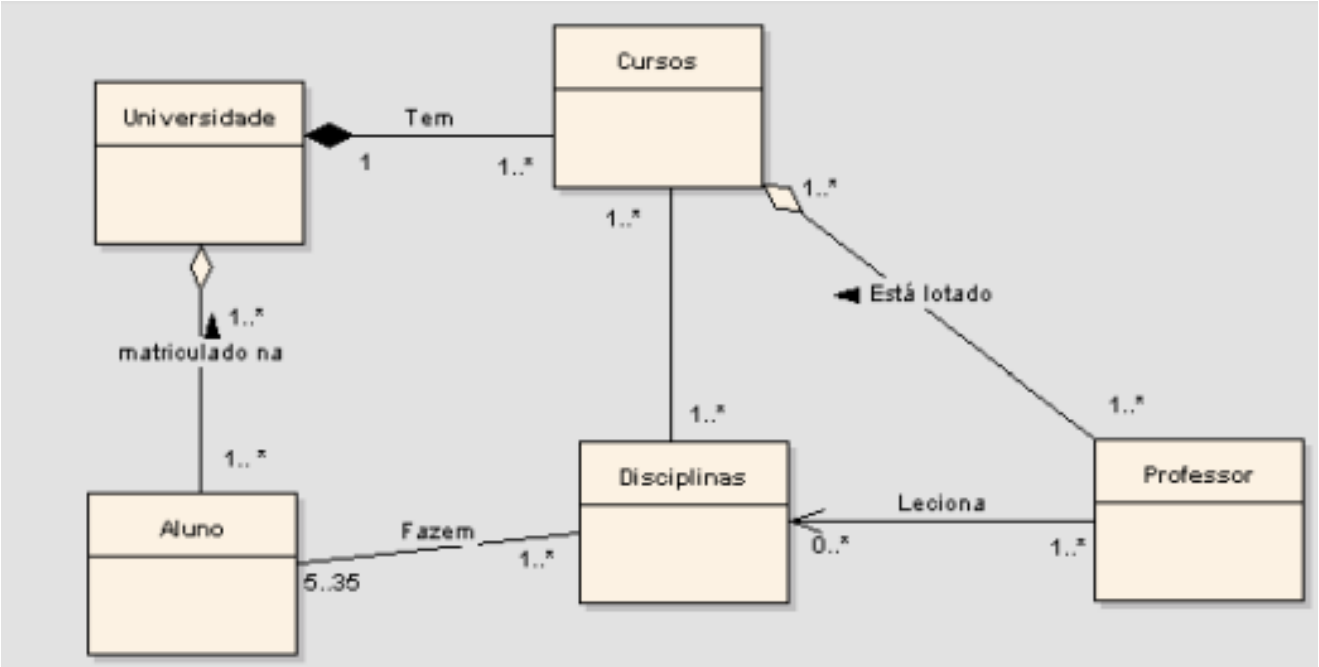
- A composição é um **tipo forte de associação**, onde um objeto agregado é composto de vários objetos componentes (Rumbaugh, 1994).

A classe “motor” e “bateria” neste exemplo não existem de forma independente da classe veículo. Elas fazem parte do todo “Veículo”.



# Composição

- Na representação das classes a associação de composição exprime que o “item de pedido” não existe sem o “Pedido”, ou seja , o “item de pedido” não existe de forma independente no sistema.



# Revisão sobre relacionamentos

# Associação

Pode incluir relacionamentos mais fortes como Agregação e Composição

- O termo associação tem sido usado até agora para descrever o relacionamento entre duas classes num diagrama de classes.
- Uma associação, na verdade, representa um tipo de relacionamento entre duas classes e descreve a extensão pelas quais elas dependem uma da outra.

Frases de validação (relacionamentos)

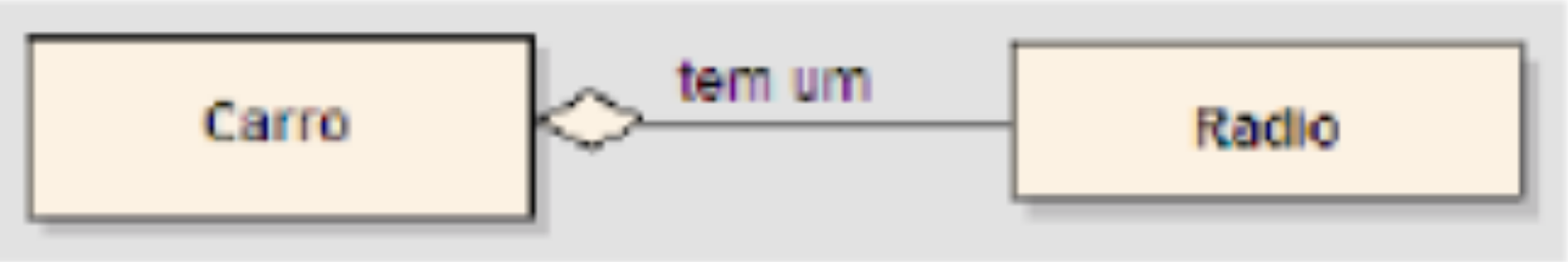
- Herança: Um labrador *é um* cão.
- Associação: Um gerente *supervisiona* um contratado.
- Agregação: Um carro *tem um* rádio.
- Composição: Um carro *sempre contém um* motor.

# Código na associação

```
public class Carro {

  private Pessoa pessoa;
  //outros atributos

  public setPessoa(Pessoa pessoa)
  {
    this.pessoa = pessoa;
    //mais codigo
  }

}

// em outro lugar do código

Carro carro = new Carro(2000,6,"Ford","Scorpio");
Pessoa pessoa = new Pessoa("Homer","Simpson");
carro.setPessoa(pessoa);

//fim do código
```

# Relembrando agregação

- Forma de associação
- Ênfase mais forte em como os dois objetos relacionarão dentro do sistema;
- Caracterizado por um relacionamento “tem um”;
- Conceito de todo/parte;
- Um carro tem um rádio para ser completo.

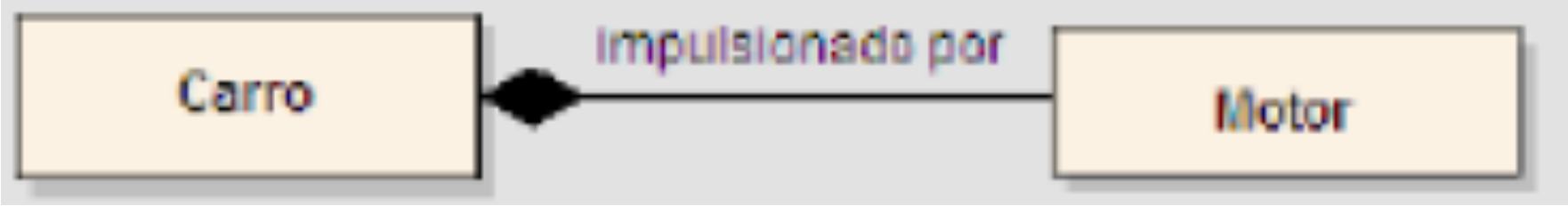


# Código na agregação

```
public class Carro {

  private Motor motor;
  //outros atributos

  public Carro(Motor motor, String marca, String modelo)
  {
    this.motor = motor;
    //mais codigo
  }

}

// em outro lugar do código
Motor motor = new Motor(2000,6);
Carro carro = new Carro(motor,"Ford","Scorpio");
//fim do código
```

# Relembrando composição

- A linha da associação na extremidade do “composto” é marcada por uma losango preto.
- Um carro sempre contém um motor.
- Usar composição ao invés de associação ou agregação depende de quão forte é o relacionamento.



# Código composição

```
public class Carro {

  private Motor motor;

  //outros atributos

  public Carro(int tamanhoMotor, int cilindros, String marca, String modelo)
  {
    motor = new Motor(tamanhoMotor, cilindros);
    //mais codigo
  }

}

// em outro lugar do código
Carro carro = new Carro(2000,6,"Ford","Scorpio");
//fim do código
```

# Exercício 2

# Exercício 2

- Utilizando como base o cenário e as classes já levantadas no exercício 1, modele as relações entre as classes.

- Verifique se todas as classes modeladas são suficiente para suportar o sistema e a sua extensibilidade.

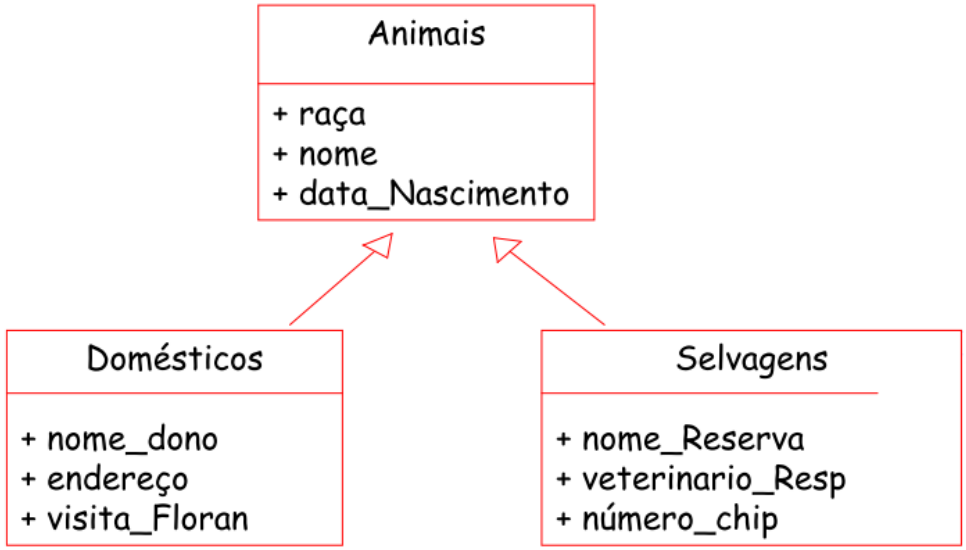
# Exercício 2

- Desenhe a sua proposta de solução:

# Herança e Polimorfismo

# Orientação a objetos

- Herança:
  - O mecanismo de herança permite que uma classe seja criada a partir de outra classe (superclasse). A nova classe (subclasse) herda todas as suas características.



# Orientação a objetos

- Herança:
  - Auxilia no **reaproveitamento** de código;
  - Cria uma nova classe a partir de uma classe existente:
    - Absorvendo os dados e comportamentos da classe existente e aprimorando-a com novas capacidades.
  - A subclasse estende a superclasse:
    - Subclasse: grupo mais especializado de objetos, seus comportamentos podem ser herdados da superclasse e personalizados.

# Orientação a objetos

- Herança (hierarquia de classes):
  - **Superclasse direta**: herdada explicitamente (um nível acima na hierarquia);
  - **Superclasse indireta**: herdada de dois ou mais níveis acima na hierarquia;
  - **Herança única**: herdade de uma superclasse;
  - **Herança múltipla**: herdada de múltiplas superclasses.
    - O JAVA não suporta herança múltipla!!!

# Orientação a objetos

- Herança:
  - Frequentemente, um objeto de uma classe também "**é um**" objeto de uma outra classe.

    - Exemplo: Em geometria, um retângulo **é um** quadrilátero.
      - A classe Retângulo herda da classe Quadrilátero;
      - Quadrilátero: superclasse;
      - Retângulo: subclasse.

# Orientação a objetos

- Herança:
  - A superclasse em geral representa um conjunto maior de objetos do que as subclsses.

    - Exemplo:
      - Superclasse: Veículo
        - » Carros, caminhões barcos, bicicleta...
      - Subclasse: Carro
        - » Subconjunto mais específico e menor de veículos.

# Orientação a objetos

- Herança (Hierarquia de herança):
  - Relacionamento de herança: **estrutura de hierarquias do tipo árvore**.

  - Cada classe torna-se:
    - **Super**classe: que **fornece** membros a outras classes;

  Ou

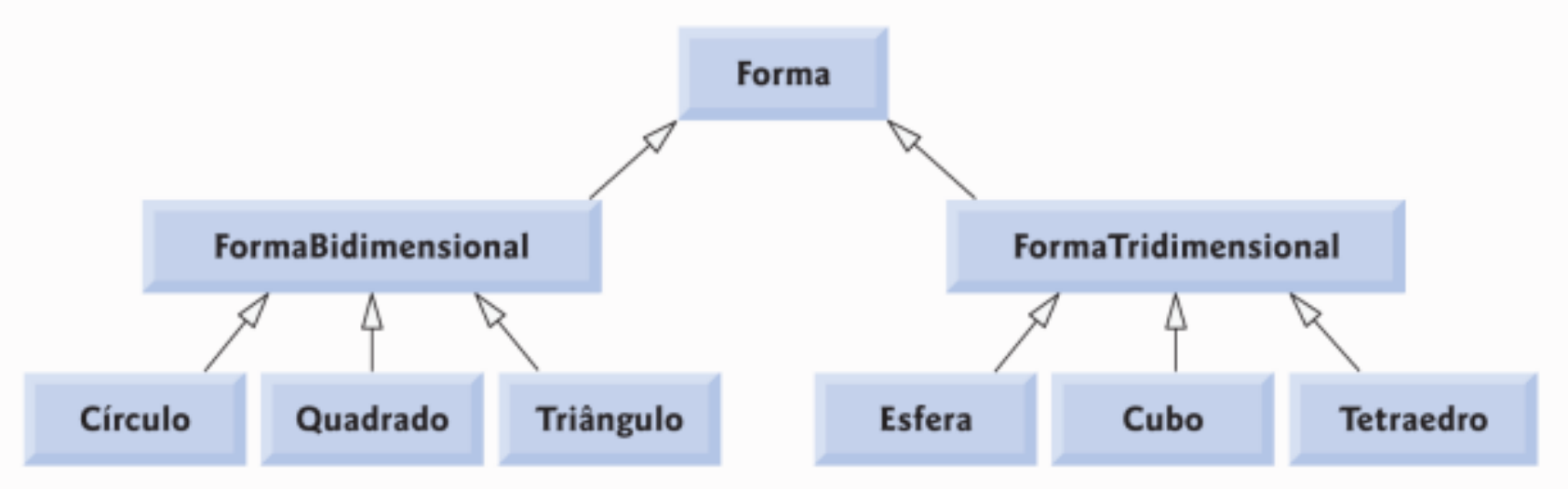
    - **Sub**classe: que **herda** membros de outras classes.

# Orientação a objetos

- Herança (Hierarquia de herança):

# Orientação a objetos

- Herança (Hierarquia de herança):



# Classes abstratas e Interfaces

# Classes Abstratas

- São classes que contêm **funcionalidades incompletas** e não podem ser instanciadas dentro de um sistema.

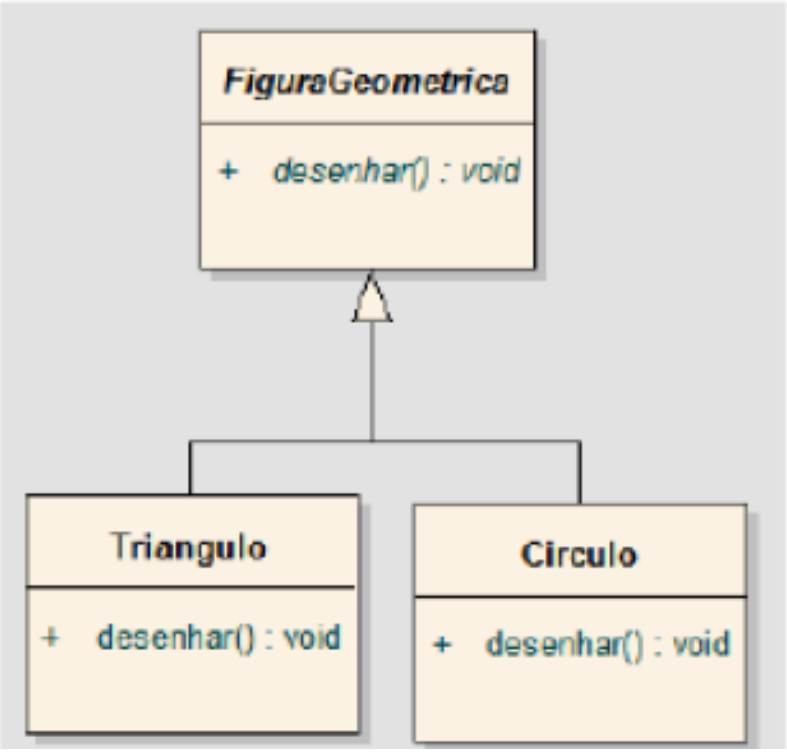
- Uma classe que **estende** uma **classe abstrata** herda todos os seus métodos (**incluindo os abstratos**)

- Quando uma **subclasse reescreve um método que ela herdou**, é dito que ela **sobrescreveu** o método.

# Classes Abstratas

- **Qualquer classe que herda um método abstrato, também é considerado abstrata**, a menos que o método seja reescrito dentro da subclasse e totalmente implementado.

# Classes Abstratas



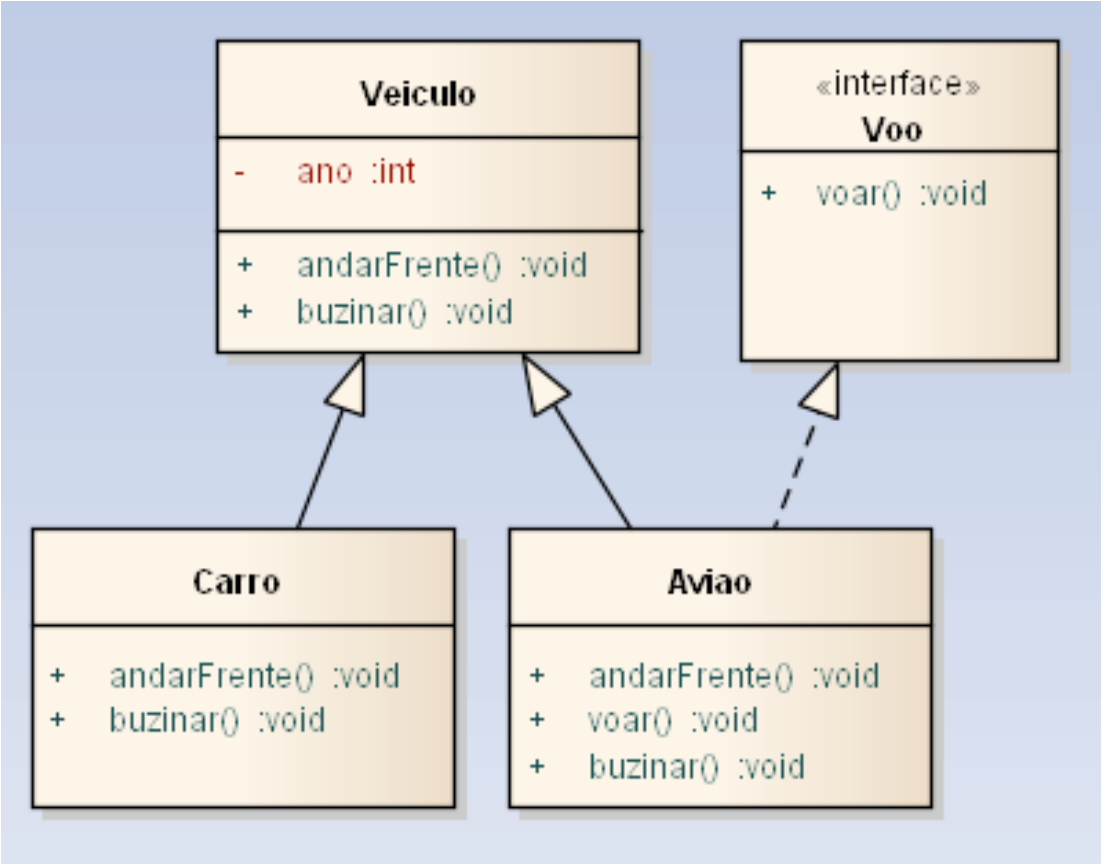

- Os nomes das classes abstratas são escritos em *itálico*
- Quaisquer métodos abstratos daquela classe também são escritos em *itálico*
- Ou a palavra "abstract" é colocada entre chaves após o nome do método (isto é {abstract})
- Chaves após o nome do método são chamadas de lista de propriedades do método.

# Interfaces

- Uma interface é uma **coleção de declarações** de métodos sem dados e sem corpo;
- São apresentadas **apenas as assinaturas dos métodos**, de modo que a classe que vir a implementar a interface em questão tenha que respeitar tais métodos.
- Funciona como um **contrato** que **obriga** a classe que a implementar **desenvolver tais métodos**.

# Interfaces

- Considerando que uma interface é um contrato, vamos pensar na seguinte situação:



# Polimorfismo

# Orientação a objetos

- Polimorfismo:
  - É a propriedade segundo a qual **uma operação** pode se **comportar** de **modos diversos** em **classes diferentes**.

Observe uma classe Veículos, uma bicicleta ou um carro fazem parte da classe mas em sub-classes diferentes.

Ambos os objetos herdam a operação *acelerar()*, mas para o objeto Carro a implementação da operação seria diferente pois no carro isto é possível pela aceleração e na bicicleta pelo aumento de pedaladas!

# Orientação a objetos

- Polimorfismo:
  - A mesma invocação pode produzir "**muitas formas**" de resultados.

- Interfaces:
  - Implementadas pelas classes a fim de atribuir funcionalidades comuns a classes possivelmente não-relacionadas.

# Orientação a objetos

- Polimorfismo:
  - Quando um programa **invoca um método** por meio de uma **variável de superclasse**, a **versão correta de subclasse do método é chamada** com base no tipo da referência armazenada na variável da superclasse.
  - Facilita a adição de novas classes a um sistema com o mínimo de modificações no código do sistema.

# Orientação a objetos

- Polimorfismo:
  - Com o polimorfismo, o **mesmo nome** e assinatura de método **podem ser utilizados** para fazer com que **diferentes ações** ocorram, **dependendo** do **tipo de objeto** em que o método é invocado.
  - O polimorfismo permite que programadores tratem de generalidades e deixem que o ambiente de tempo de execução trate as especificidades.

# Orientação a objetos

- Polimorfismo:
  - Os **programadores** podem instruir **objetos a se comportarem de maneiras apropriadas** para esses objetos, **sem** nem mesmo **conhecer os tipos dos objetos** (contanto que os objetos pertençam à mesma hierarquia de herança).
  - O polimorfismo promove **extensibilidade**: O software que invoca o comportamento polimórfico é independente dos tipos de objeto para os quais as mensagens são enviadas.

# Orientação a objetos

- Polimorfismo:
  - **Novos tipos de objetos** que podem **responder a chamadas de método existentes** podem ser incorporados a um sistema **sem exigir modificações no sistema básico**.
  - Somente o código de cliente que instancia os novos objetos deve ser modificado para, assim, acomodar os novos tipos.

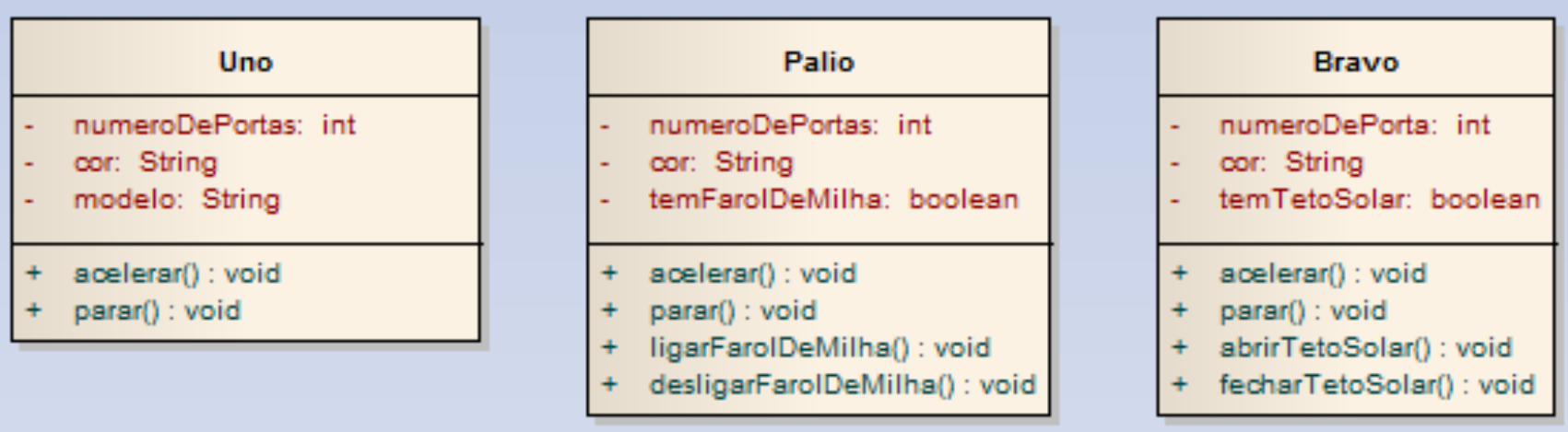
# Orientação a objetos

- Polimorfismo (comportamento):
  - Uma referência de superclasse pode ter por alvo um objeto de subclasse:
    - **Isso é possível porque um objeto de subclasse também é um objeto de superclasse**;
    - Ao invocar um método a partir dessa referência, o tipo do objeto referenciado real, não o tipo da referência, determina qual método é chamado.
  - Uma referência de subclasse pode ter por alvo um objeto de superclasse somente se o objeto sofrer **downcasting**

# Exercício 3

# Exercício 3

- Remodele o diagrama abaixo utilizando os conceitos de orientação a objeto para que a solução seja mais robusta, flexível e para obter o melhor aproveitamento de código, mais detalhes a seguir:



# Exercício 3

- Alguns pontos importantes para a nova solução dessa modelagem:

  – O carro Uno acelera com características de um motor 1.0, já o carro Palio com características do motor 1.4 e o carro Bravo com a de motor 1.8.

  – O sistema deve estar preparado para suportar novos modelos de carros, utilizando motores já conhecidos ou suportar novos motores, que terão reflexo direto na aceleração do carro.

# Exercício 3

- Desenhe a sua proposta de solução:

# Exercício 4

# Exercício 4

- Desenvolva uma proposta de solução construindo um diagrama de classes, que atenda ao cenário apresentado a seguir.

- O modelo proposto tem que seguir as noções da orientação à objetos, deixando a solução robusta e mais extensível.

# Exercício 4

- Uma universidade solicitou que se desenvolva um sistema para controlar todo o seu processo de matricula.

- Todo aluno deve informar algumas características, como por exemplo o seu nome, matricula, telefone, sexo e endereço.

# Exercício 4

- Existem 3 tipos de alunos:
  - Graduação: esse tipo de aluno deve apresentar também a unidade em que o mesmo foi matriculado;
  - Mestrado: Deve apresenta quem é o seu orientador, que é um professor do curso que ele se propôs a fazer e a sua carga horária.
  - Doutorado: Seguem as mesmas características do mestrado.

# Exercício 4

- Cada aluno pode se matricular em disciplinas que possuam turmas em aberto, pode-se abrir mais de uma turma de uma disciplina;

- Uma turma contém as seguintes informações: um código identificador, um professor vinculado, um semestre, um ano, uma sala, uma unidade, um curso que ela pertence, uma disciplina vinculada e um período.

# Exercício 4

- Uma disciplina deve conter o seu nome, um código identificador, uma relação com o curso de origem, sua carga horária, e o nível correspondente;

- Um curso deve apresentar o seu nome, a área de conhecimento que faz parte, o nível, e informar o seu coordenado, que é um professor.

# Exercício 4

- Um professor deve informar o seu nome, sua titulação máxima, sua matricula, a sua carga horária máxima e seu vínculo com a instituição: (titular ou horista).

# Exercício 4

- Desenhe a sua proposta de solução: