



Introdução a Projetos de Software

Programação III

Prof. Edson Mota, PhD, MSc, PMP

Quem sou eu?

Sistema FIEB



PELO FUTURO DA INOVAÇÃO

- Edson Mota, PhD, MSc, PMP®, ITIL®

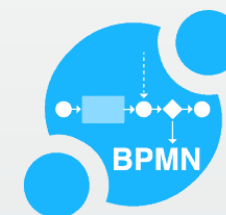
- Graduação em Computação
- MBA em Gestão da Informação e Business Intelligence
- Especialização em Metodologia do Ensino Superior
- Mestre em Sistemas e Computação
- Doutor em Ciência da Computação pela UFBA

- Certificações Profissionais:

- *Project Management Professional (PMP)®*
- *Information Technology Infrastructure Library (ITIL)®*
- *Business Process Model and Notation (BPMN)®*

- Atuação Profissional

- **Tech Leader** em Projetos de Pesquisa e Desenvolvimento do CIMATEC
- Além disso, atuei com **consultor** por mais de 15 anos pela SPHERA TECNOLOGIA
 - Experiência no gerenciamento de projetos com entregas no Brasil e exterior
 - Principalmente em **projetos de software** e elaboração de ambientes **Business Intelligence**.



Algumas produções acadêmicas

- Edson Mota, PhD, MSc, PMP[®], ITIL[®]
- Autor de artigos nacionais e internacionais no campo do gerenciamento de projetos; Internet das coisas; Redes veiculares; Sistemas Distribuídos e Blockchain.
- Autor do Livro: **Gestão De Stakeholders - Uma Abordagem Teórico-prática Utilizando A Tecnologia Da Informação Como Suporte Na Gestão De Stakeholders.**



Contato:

edsonmottac@gmail.com



Objetivo

Ampliar a experiência em projetos de software.

Competências Específicas

- Ampliar a experiência em projetos de sistemas orientados a objetos utilizando componentes de software e conceitos de reuso e coesão;
- Desenvolver experiência em projetos de sistemas orientados a objetos sobre a plataforma C#;
- Desenvolvimento de softwares com a utilização de linguagens de programação orientadas a objeto;
- Construir aplicações utilizando Windows Forms.

O que você entende por
Desenvolvimento de
Software?



Desenvolvimento de Software

- O **desenvolvimento de software** refere-se a um conjunto de atividades da computação dedicadas ao processo de criação, design, implantação e suporte de software.

Software, pode ser entendido como o conjunto de instruções ou programas que dizem ao computador o que fazer.



Como ocorre o processo de Desenvolvimento de Software?



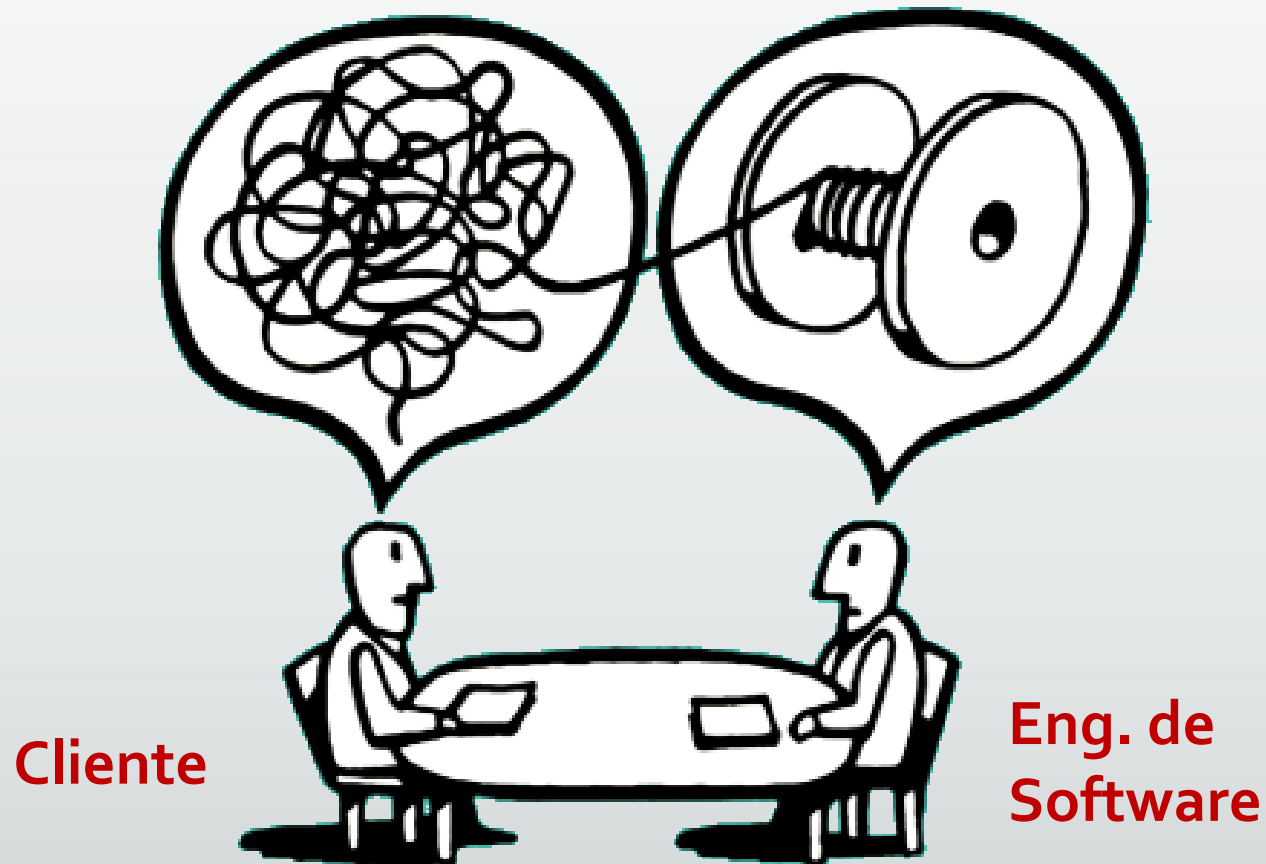
Panorama Geral

- Envolve, em geral, etapas como:
 - Análise de requisitos
 - Projeto
 - Desenvolvimento
 - Testes
 - Implantação



Análise de Requisitos

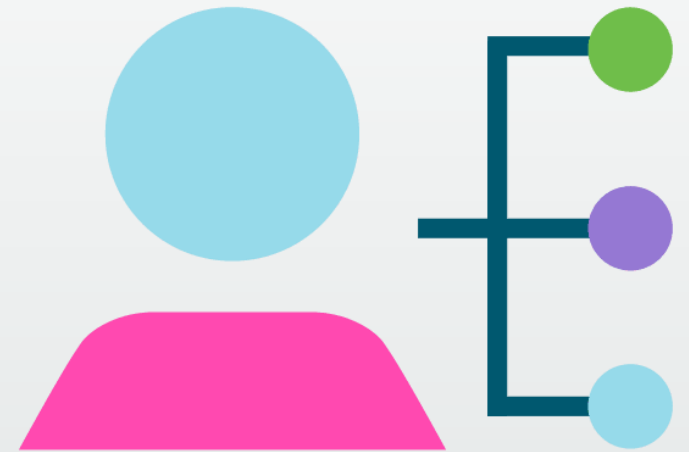
- Trata-se de **extrair** as necessidades dos clientes.



Comunicação é
fundamental

Design / Projeto

- Envolve decisões **arquiteturais** do projeto.
 - Fundamentadas nos requisitos do projeto.
- Estão entre as decisões de projeto:
 - Definições acerca da estrutura do software;
 - Divisão de responsabilidades;
 - Definição de classe e relacionamentos (modelo);
 - Produção de diagramas (casos de uso, arquitetura, etc);
 - Entre outros...



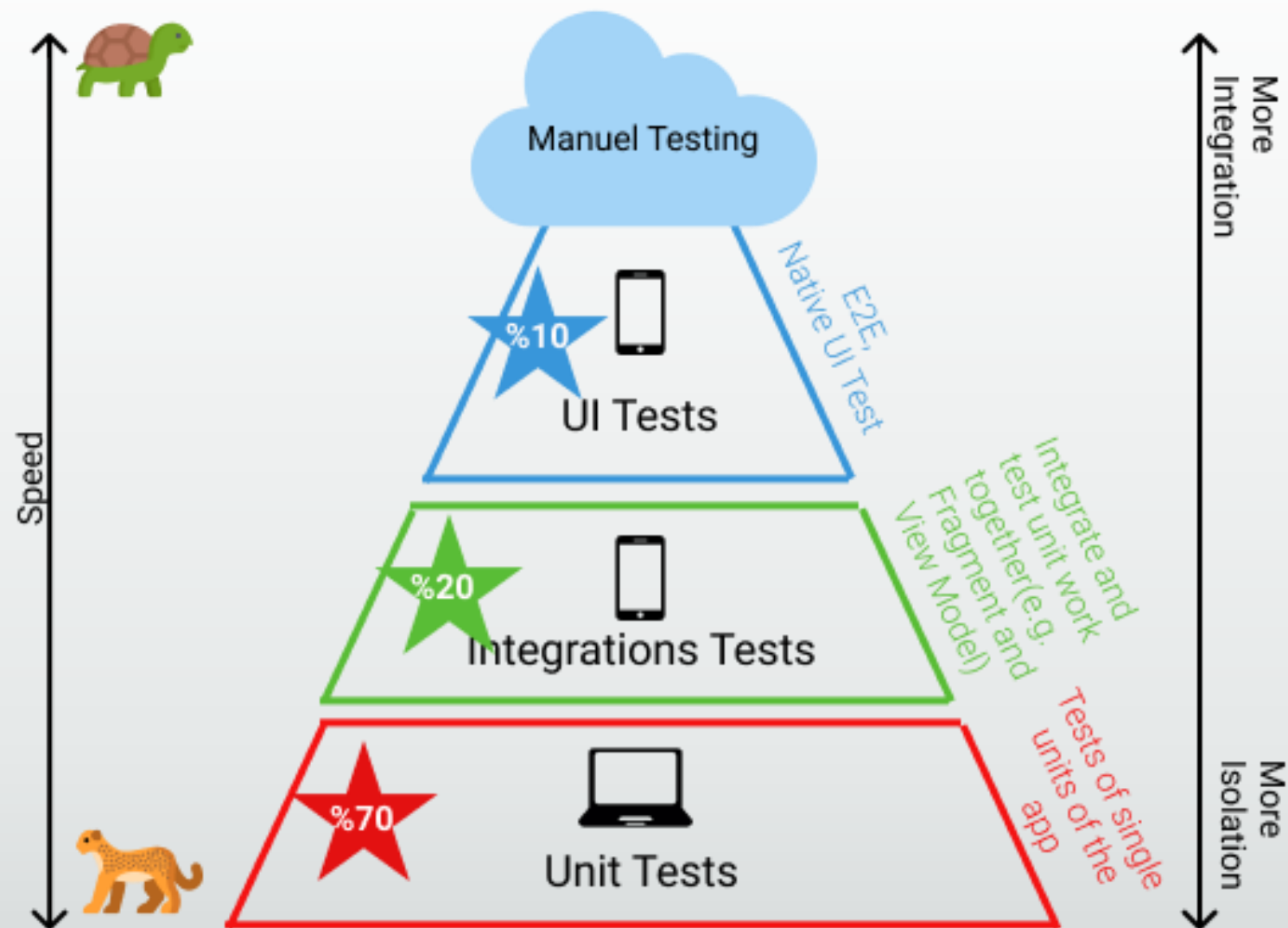
Desenvolvimento / Implementação

- Consiste na **materialização** do software por meio da **codificação** dos seus módulos e funcionalidades.
 - Tudo isso em concordância com as definições de projeto e seus requisitos.
- Envolve:
 - Codificação;
 - Escolhas de Padrões de Projeto;
 - Definições de Protocolos de Comunicação;
 - Estrutura e Persistência de Dados;
 - Entre outros.



Testes

Consistem em executar as funcionalidades de uma aplicação, utilizando um conjunto de dados conhecido, e analisar se os resultados são esperados.



Deployment / Implantação

- Consiste no processo de disponibilizar a aplicação em um certo ambiente para utilização de usuários e demais stakeholders.
- A tarefa de “*Deployar*”, como é normalmente referenciada no vocabulário de TI, tem se aproximado cada vez mais das equipes de desenvolvimento.
- Esse movimento tem culminado em um novo leque de atividades conhecidas como **DevOps**.

- **Infraestrutura sobre código!**



Por que precisamos de um
processo de Desenvolvimento
de Software?

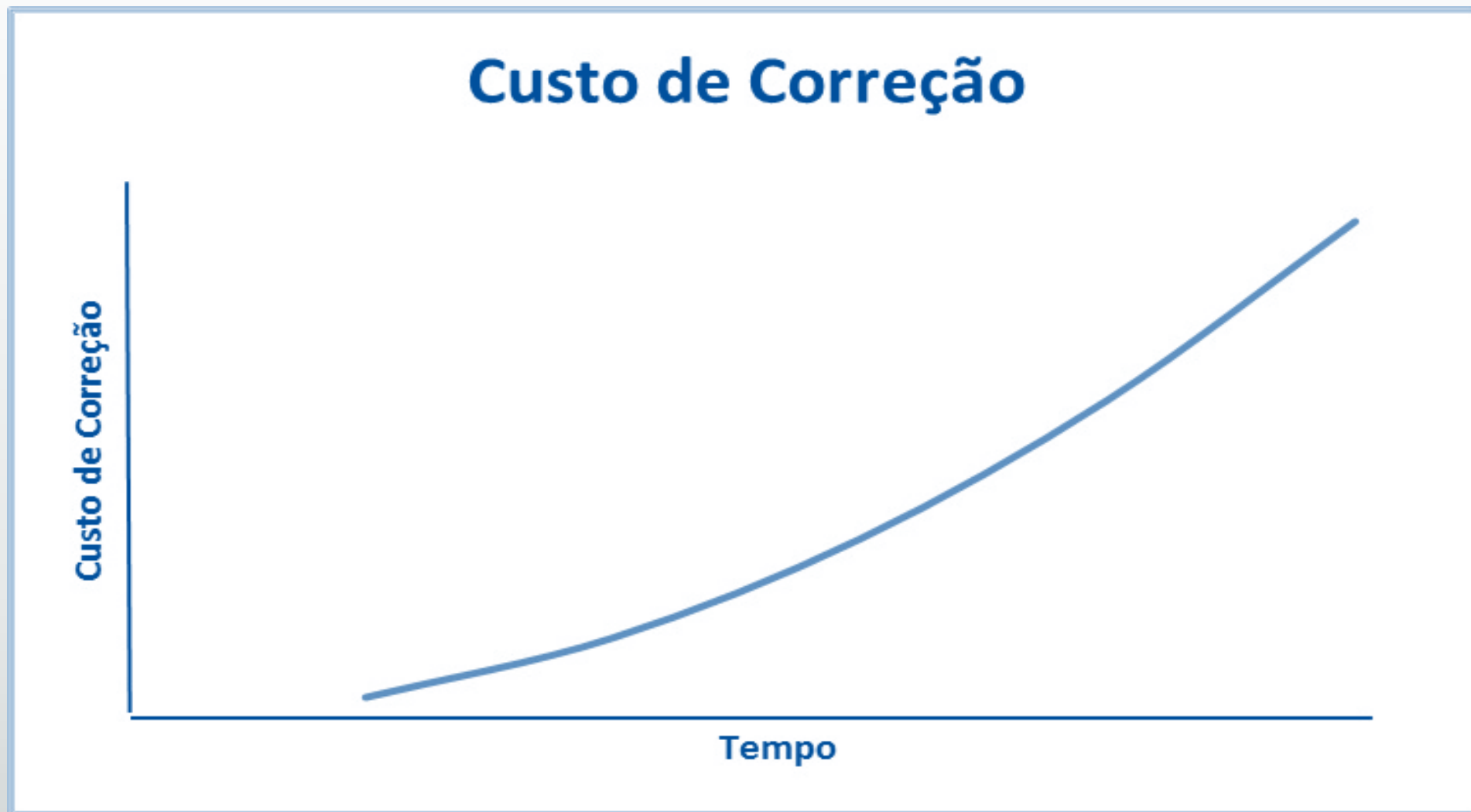


Um dos motivos....

- Programadores argumentam que o **tempo de escrita do código, durante o desenvolvimento**, consumirá apenas de **20 a 40 por cento** do tempo dessa atividade.
- O **restante** será utilizado em atividades como:
 - Leitura e entendimento **do código já escrito**
 - e **manutenção** destes mesmos códigos.

Uma arquitetura bem definida pode reduzir significativamente esse retrabalho

O que esse gráfico nos diz?



Falhas em Projetos de Software

- As falhas em projetos de software são cometidas, em grande parte por :
 - Entendimento inadequado do problema;
 - Baixo grau de padronização durante o desenvolvimento;
 - Documentação pobre;
 - Pobre entendimento sobre a aplicação da orientação a objeto nos cenários de projetos;
 - Entre outros.



Falhas no Design



Falhas no Design



**QUANDO O CÓDIGO FUNCIONA MAS
NÃO DA FORMA QUE ESPERAVA**





Aprimorar o processo de desenvolvimento é um passo importante para mitigar estes problemas.

A programação orientada a objeto é uma importante aliada nesse processo.



**Vamos Relembrar um Pouco de
P.O.O?**

Como você define a Programação
Orientada Objeto?



Pensando em Programação

*"A essência da programação de computadores é uma **constante troca** entre **abordagens orientadas a máquinas** por **conceitos e metáforas** que reflitam melhor o nosso entendimento sobre a forma **como vemos o mundo.**"*

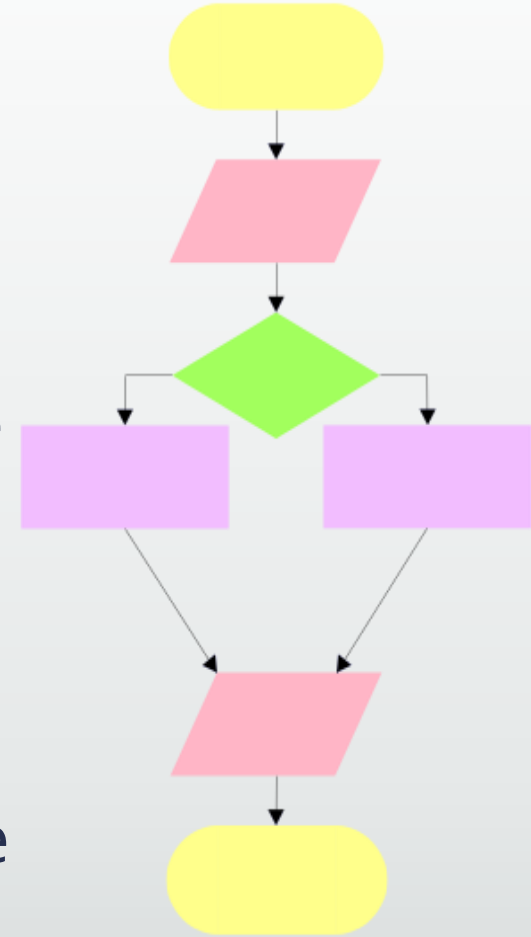


"Objetos representam mais um passo nesta direção"

Existem dois mundos

- Programação Procedural

- As primeiras linguagens de alto nível são tipicamente chamadas de procedurais
- São caracterizadas por um conjunto sequencial de linhas de comando
 - O foco destas linguagens é a sua própria estrutura
 - Exemplos de linguagens procedurais
 - C, COBOL, Fortran, LISP, Perl, HTML, VBScript, entre outras

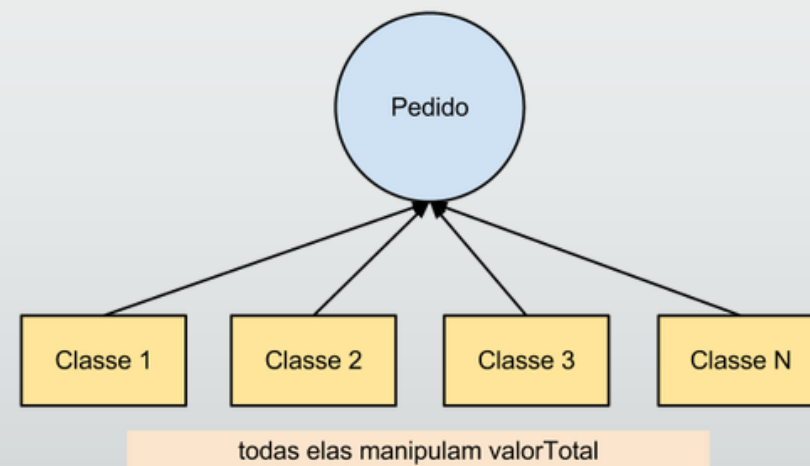


Existem dois mundos

- Programação Orientada a Objeto
 - O foco não é a estrutura, mas sim a forma como os dados são modelados e se inter-relacionam dentro do programa
 - Os códigos dos programadores utilizam “*moldes*” para os dados, estes moldes (*Classes*) ajudam a replicar elementos com as mesmas características (*Objetos*)

Exemplos de linguagens POO incluem:

C++, Visual Basic.NET, Java, Python, PHP, entre outras.



Mas, o que é um Objeto?



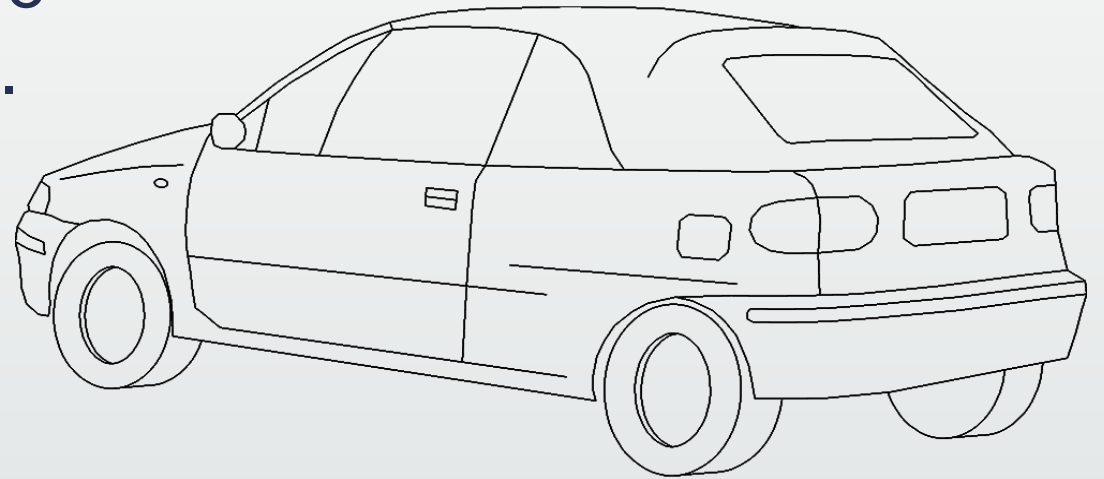
Modelando Objetos

- Tomemos um objeto como exemplo
 1. Quais as características deste objeto?
 2. O que este objeto faz?
 - Quais ações ele realiza?
 - Dê um nome para essas ações
 3. Quais os estados possíveis para este objeto?



Descrevendo nosso objeto

- Se todos os objetos mapeados compartilham das mesmas características
 - Podemos observar o objeto “carro” **a partir de um modelo genérico.**
 - Afinal, ele possui as mesmas características anotadas anteriormente
- Este modelo **classifica** as características que observamos neste objeto



Vamos chamar este
“*molde*” de **Classe**

Descrevendo nosso objeto

- Seguindo as especificações da Classe que detalham como é uma determinada “coisa” chegamos ao objeto, logo:



As especificações da
“coisa” contidas na **Classe**

Objeto: *Pode ser
entendido como um **item**
daquela Classe*

Assim, uma **classe** tem os seguintes elementos

- Quais as características deste objeto?

- 4 portas
- 4 pneus
- Faróis
- Placa



Atributos

- Quais ações ele faz?

- Ligar Faróis
- Ligar/Desligar
- Acelerar
- Frear

Métodos

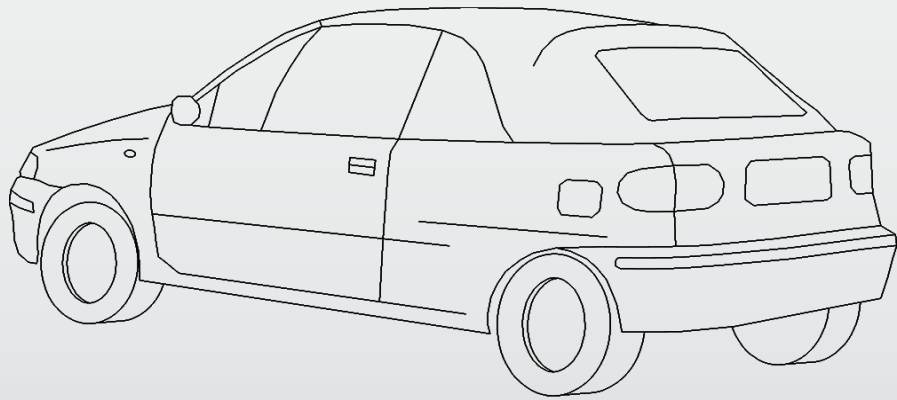
- Quais os estados possíveis?

- Ligado
- Desligado
- Farol acesso, etc.

Estados

Descrivendo nosso objeto

- No momento da criação da classe, ela é apenas uma referência para um futuro objeto
 - O ato de criar um objeto a partir de uma classe chama-se **instanciar**

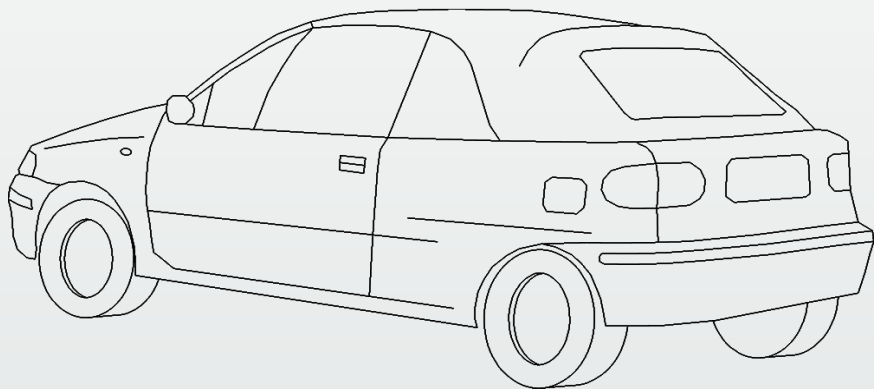


Instanciar

Criar uma instância de um
dado objeto



Novamente, pensando em **coleções** de objetos....



Nossa **classe** pode gerar quantos objetos sejam necessários



Obj1



Obj2



Obj3



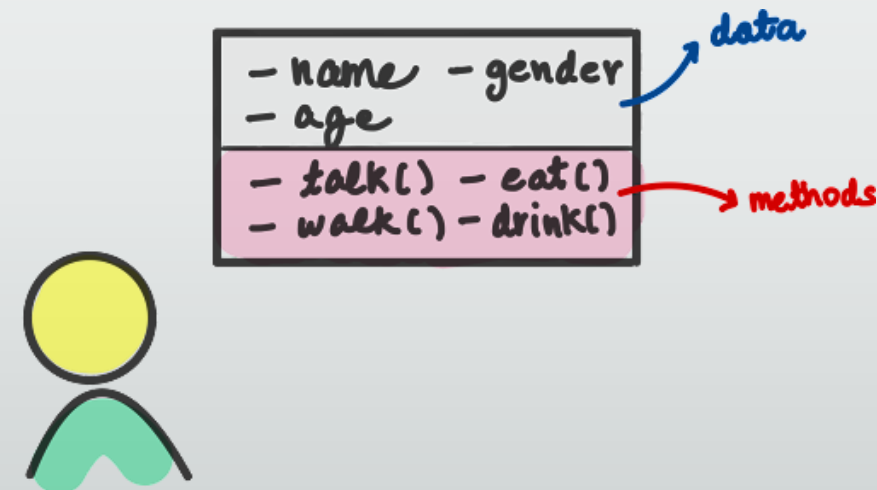
Obj4



“O mundo real pode ser
detalhadamente explicado como
uma coleção de objetos que
interagem entre si”

Programação Orientada a Objeto

- Absolutamente **tudo** em POO é referenciado como “*objetos auto sustentáveis*”.
- Assim, ganhamos em **reutilização** por meio de quatro principais conceitos da programação orientada a objeto
 - Classes
 - Atributos
 - Métodos
 - Objetos



Classes e objetos

- A classe é um modelo (**uma fôrma**) para criar objetos
- Um objeto é uma instância de uma classe
 - Uma “coisa” é criada a partir deste modelo e chamamos de **objeto**
- Em C#, definimos classes que, por sua vez, são utilizadas para criar objetos
- Uma classe tem um **construtor** utilizado para criar novos objetos
- A classe é composta de três coisas: nome, atributos e métodos

As ações da classe

- Em POO, além de definirmos a estrutura dos dados, como eles se comportam e fluem pelo sistema, **nos preocupamos ainda com as operações que podem ser aplicadas sobre estes dados**
- Em POO, a estrutura de dados torna-se um objeto. Este objeto comporta **dados e funções (métodos)**
 - Um objeto pode herdar características de outro
 - Adicionalmente, podemos criar relações entre estes objetos

Por que precisamos de um outro
paradigma **Programação**?



Linguagens orientadas a objeto

- Você está trabalhando para uma empresa de veículos que precisa atualizar seu inventário online. Seu chefe diz que você deve programar **duas propostas** similares, mas separadas de um formulário para o website.
 - Para **Carros**
 - Informações: **Cor**, **potência do motor**, **tipo de transmissão**, numero de portas
 - Para **Caminhões**
 - Informações: **Cor**, **potência do motor**, **tipo de transmissão**, tamanho da cabine, capacidade do reboque

Linguagens orientadas o objeto

▪ Usando Linguagem Procedural

Formulário Carro

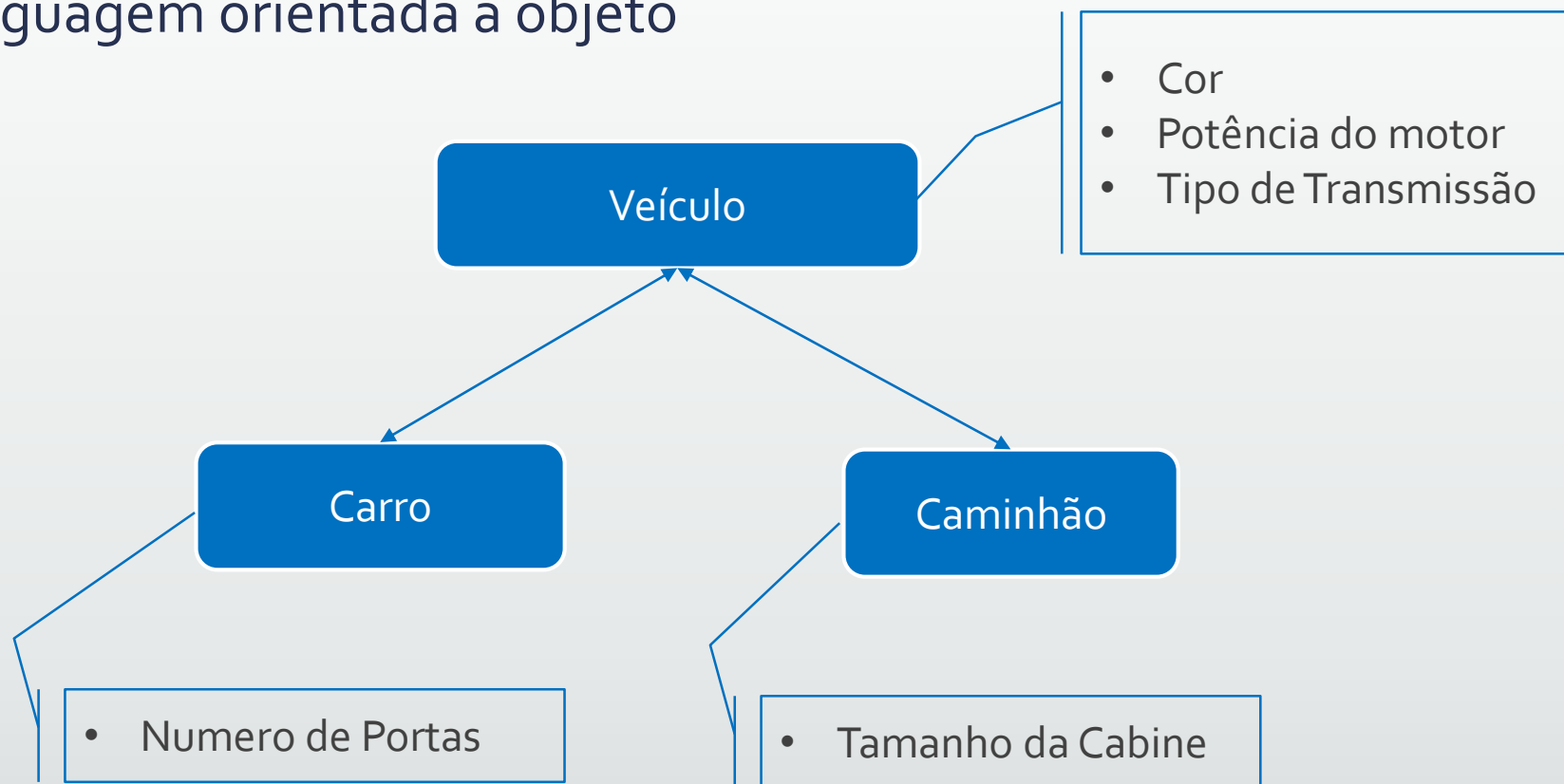
- Cor
- Potência do motor
- Tipo de Transmissão
- Numero de Portas

Formulário Caminhão

- Cor
- Potência do motor
- Tipo de Transmissão
- Tamanho da Cabine

Linguagens orientadas o objeto

- Usando Linguagem orientada a objeto



Cenário 1

- Supondo que em breve você precisará adicionar um formulário para ônibus, o que você faria?
 - Informações : **Cor**, **potência do motor**, **tipo de transmissão**, **número de passageiros**
 - **Solução Procedural**
 - Será necessário recriar todo o formulário repetindo o código da **cor**, **potência do motor**, **tipo de transmissão**
 - **Solução orientada a objeto**
 - Nós simplesmente **estendemos a classe veículo** a partir de uma classe **ônibus** e adicionamos a propriedade **número de passageiros**

Linguagens orientadas a objeto

▪ Usando Linguagem Procedural

Formulário Carro

- Cor
- Potência do motor
- Tipo de Transmissão
- Numero de Portas

Formulário Caminhão

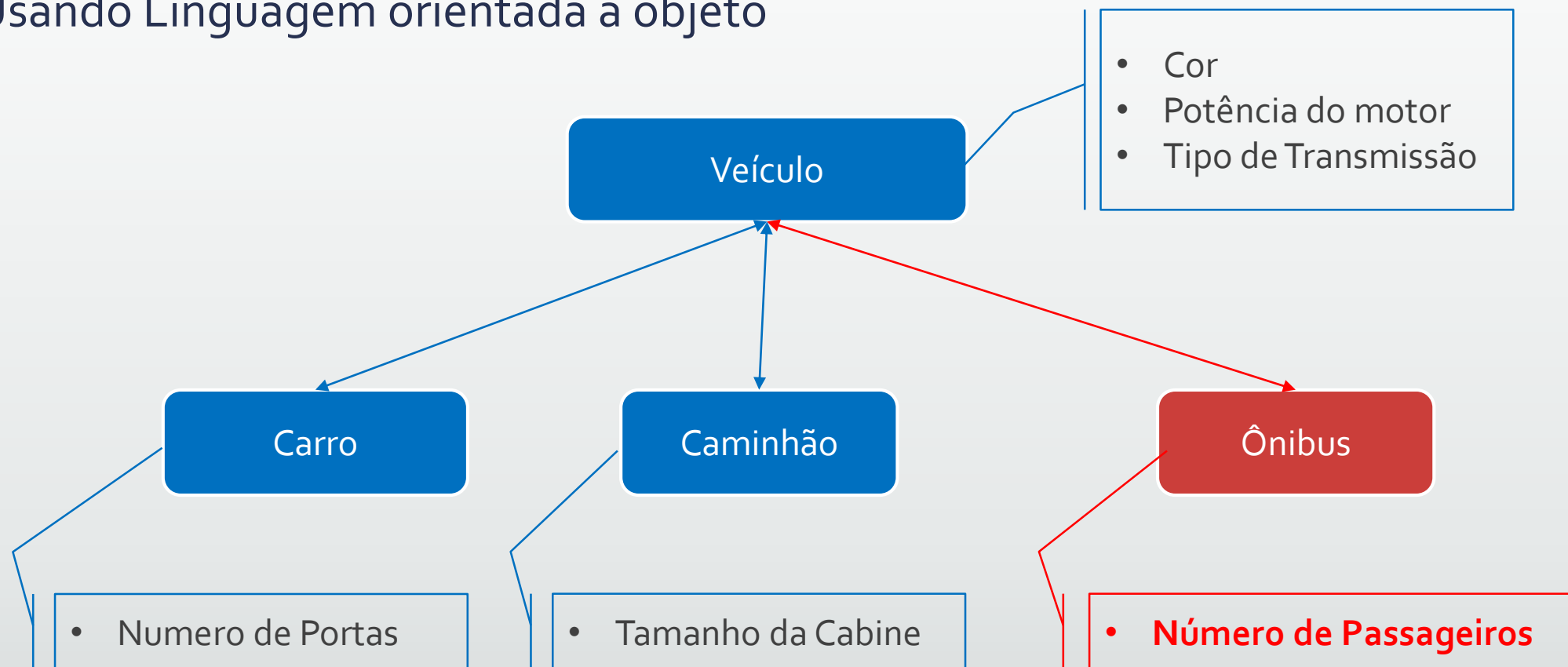
- Cor
- Potência do motor
- Tipo de Transmissão
- Tamanho da Cabine

Formulário Ônibus

- Cor
- Potência do motor
- Tipo de Transmissão
- **Número de Passageiros**

Linguagens orientadas a objeto

- Usando Linguagem orientada a objeto



Cenário 2

- O cliente quer ter a opção de enviar novas cores para o banco de dados utilizando apenas o seu e-mail. As cores são recebidas, cadastradas no banco de dados e disponibilizadas nos relatórios
 - **Solução Procedural**
 - Será necessário **modificar todos os formulários** para receber a **cor** que deverá ser incluída no banco de dados
 - **Solução orientada a objeto**
 - Basta modificar o método **cor** na **classe veículo**, porque **carros, caminhões e ônibus herdam** as características desta classe principal.
 - Ao atualizar a classe veículo, todos os formulários estarão atualizados automaticamente

Linguagens orientadas a objeto

▪ Usando Linguagem Procedural

Formulário Carro

- Cor
- Potência do motor
- Tipo de Transmissão
- Numero de Portas

```
ReceberCor() {  
    CorRecebida = NovaCor;  
    BancoDeDados = CorRecebida  
}
```

Formulário Caminhão

- Cor
- Potência do motor
- Tipo de Transmissão
- Tamanho da Cabine

```
ReceberCor() {  
    CorRecebida = NovaCor;  
    BancoDeDados = CorRecebida  
}
```

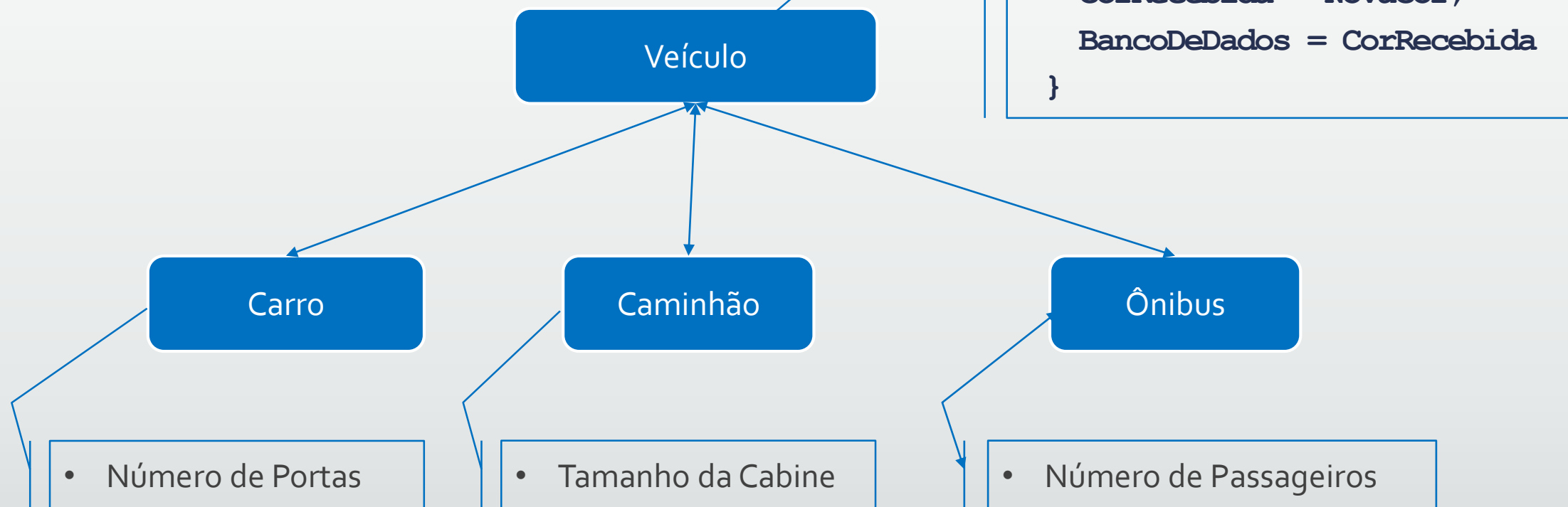
Formulário Ônibus

- Cor
- Potência do motor
- Tipo de Transmissão
- Número de Passageiros

```
ReceberCor() {  
    CorRecebida = NovaCor;  
    BancoDeDados = CorRecebida  
}
```


Linguagens orientadas a objeto

- Usando Linguagem orientada a objeto



Cenário 3

- Encontramos um erro na área de tipo de transmissão
 - **Solução Procedural**
 - Todos os formulários precisam ser atualizados.
 - O erro precisa ser corrigido em cada um deles
 - **Solução orientada a objeto**
 - Basta corrigir o erro no métodos “**tipo de transmissão**” na **classe veículo** e as mudanças serão **propagadas** para as subclasses **carros, caminhões e ônibus**.

Linguagens orientadas a objeto

▪ Usando Linguagem Procedural

Formulário Carro

- Cor
- Potência do motor
- Tipo de Transmissão
- Número de Portas

```
TipoTransmissao() {  
    correção_de_bug  
}
```

Formulário Caminhão

- Cor
- Potência do motor
- Tipo de Transmissão
- Tamanho da Cabine

```
TipoTransmissao() {  
    correção_de_bug  
}
```

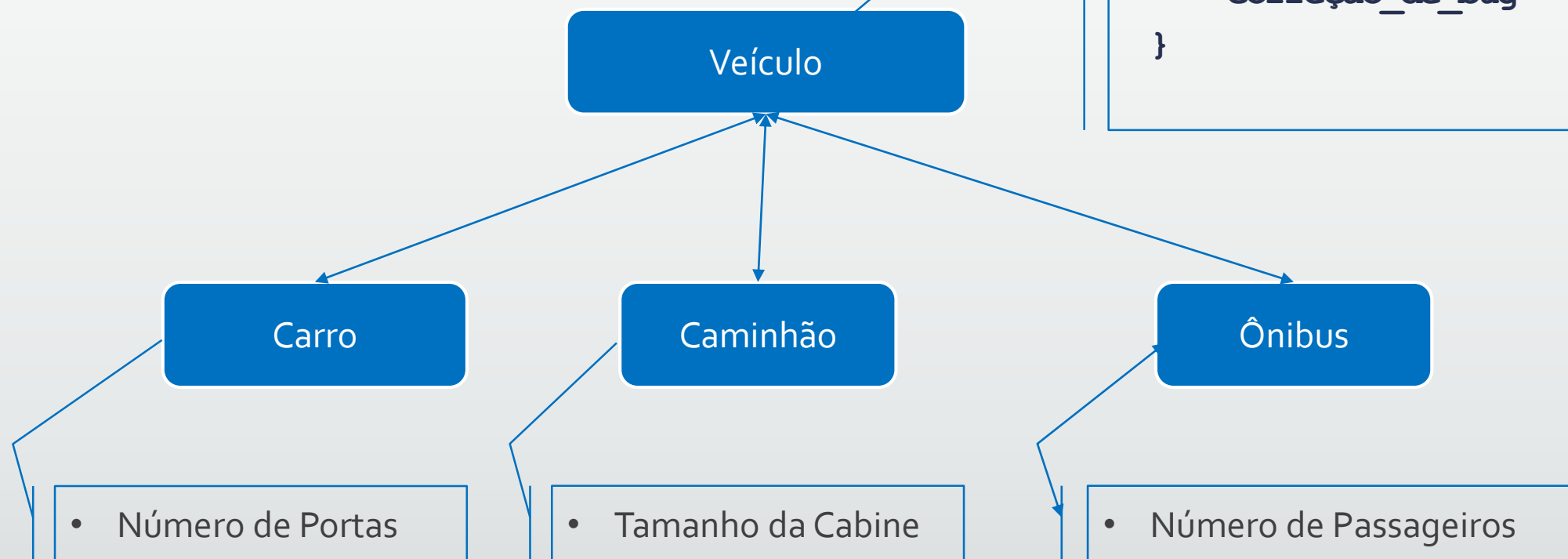
Formulário Ônibus

- Cor
- Potência do motor
- Tipo de Transmissão
- Número de Passageiros

```
TipoTransmissao() {  
    correção_de_bug  
}
```

Linguagens orientadas a objeto

- Usando Linguagem orientada a objeto




O que é um Método?



Métodos

- Consistem em um grupo de **declarações** que são incluídas em uma parte do código para **realizar** uma **determinada tarefa**.
- Eles permitem:
 - Modularizar o código
 - Dividir tarefas complexas
 - Facilitam a manutenção

```
void display() {  
    // code  
}  
....  
display();  
....
```



Method call

Anatomia dos Métodos

```
Modificador TipoDeRetorno NomeDoMetodo (Lista_de_Parâmetros)
{
    //Código do método
}
```

- **Modificadores Gerais:** Definem o tipo de acesso
 - *Public* : Acesso irrestrito a partir de todas as classes da aplicação
 - *Private*: Não podem ser utilizados em outra classe
 - *Protect*: Fornece acesso apenas a classes de um mesmo pacote

Anatomia dos Métodos

```
Modificador TipoDeRetorno NomeDoMetodo (Lista_de_Parâmetros)
{
    //Código do método
}
```

- **TipoDeRetorno**: Define o tipo do dado que será retornado pelo método.
 - Duas possibilidades
 - Se existe retorno, informa-se o tipo (**int, string, boolean, etc**)
 - Se não existe retorno: **void**

Anatomia dos Métodos

```
Modificador TipoDeRetorno NomeDoMetodo (Lista_de_Parâmetros)  
{  
    //Código do método  
}
```

- **NomeDoMetodo**: Nome que será dado ao método
 - Segue convenções similares a criação de variáveis, mas deve representar uma ação
 - ListarValores(), Calcular(), AbrirPorta(), etc

Anatomia dos Métodos

```
Modificador TipoDeRetorno NomeDoMetodo (Lista_de_Parâmetros)
{
    //Código do método
}
```

- **Lista_de_parâmetros:** Valores que o método precisa receber para realizar uma ação.
 - Opcional, se o método não precisa de nenhuma informação, não é necessário informar os parâmetros.
 - **CalcularDoisValores** (int valorA, int ValorB) { }

Anatomia dos Métodos

```
Modificador TipoDeRetorno NomeDoMetodo (Lista_de_Parâmetros)
{
    //Código do método
}
```

- **Código do Método (com retorno):** Descreve a ação que será realizada
 - Comandos sequencialmente incluídos a fim de resolver um determinado problema

```
Public int CalcularDoisValores (int valorA, int ValorB) {
    return valorA + valorB;
}
```

Anatomia dos Métodos

```
Modificador TipoDeRetorno NomeDoMetodo (Lista_de_Parâmetros)
{
    //Código do método
}
```

- **Código do Método (sem retorno):** Descreve a ação que será realizada
 - Comandos sequencialmente incluídos a fim de resolver um determinado problema.

```
Public void CalcularDoisValores (int valorA, int ValorB) {
    System.out.println(ValorA + ValorB) ;
}
```

Métodos estáticos (static)

- Em OO, para que um método possa ser executado, você precisa antes de tudo instanciar o objeto dessa classe.

```
//Instanciando o objeto  
Porta P3 = new Porta();
```

```
//Atribuindo os valores  
P3.numero=10  
P3.cor = amarela;  
P3.AbrirPorta(true);
```

Métodos estáticos não cumprem essa regra

```
public static void AbrirPorta(boolean valor) {  
  
}
```

// Para utilizar métodos estáticos...
Porta.AbrirPorta(true); // sem criar o objeto

O que é um Método Construtor?



Construtores

- São um tipo especial de método, utilizado para inicializar objetos
 - Os construtores são invocados no momento da criação do objeto

```
ClsCliente ObjCli = new ClsCliente (CPF) ;
```

- Eles ajudam a *construir* o objeto repassando informações necessárias à sua inicialização.

Construtores – Existem regras!

- Para implementar um construtor existem basicamente duas regras:
 1. O construtor precisa ter o mesmo nome da sua classe
 2. O construtor não deve ter um tipo de retorno explícito

Construtores – “dois tipos básicos”

Tipos de Construtores

```
graph TD; A[Tipos de Construtores] -.-> B[Default]; A -.-> C[Parametrizado];
```

Default

Existem, mas como não foram implementados pelo programador, realizam apenas atividades padrão

Parametrizado

São implementados pelo programador, quando uma inicialização da classe é requerida

Construtores – Na Prática!

```
class TesteC {  
    int valor = 20;  
    TesteC() {  
        numero=valor;  
    }  
}
```

Construtor sem
Parâmetro



```
public class AulaPOO{  
    public static void main(String []args) {  
        TesteC T1 = new TesteC();  
        System.out.println(T1.numero);  
    }  
}
```

Construtores – Na Prática!

```
class TesteC {  
    int numero = 0;  
    TesteC(int valor) {  
        numero=valor;  
    }  
}
```

Construtor
Parametrizado



```
public class AulaPOO{  
    public static void main(String []args) {  
        TesteC T1 = new TesteC(10);  
        System.out.println(T1.numero);  
  
        TesteC T2 = new TesteC(20);  
        System.out.println(T2.numero);    }  
}
```



Quando um método está
Sobrecarregado?



Sobrecarga de Métodos em C# (Overloading)

- Supondo que você precisa criar um programa genérico para calcular a média de um conjunto de valores (n1,n2,n3).
- No entanto, algumas vezes você terá apenas os valores(n1 e n2)
 - **Como podemos resolver isso?**

Sobrecarga de Métodos em C# (Overloading)

▪ Solução Bizarra!

```
public float MediaA(float n1, float n2) {  
    return (n1 + n2) / 2  
}
```

```
public float MediaB(float n1, float n2, float n2) {  
    return (n1 + n2 + n3) / 3  
}
```



Sobrecarga de Métodos em C# (Overloading)

- Solução Elegante!

```
public float Media(float n1, float n2) {  
    return (n1 + n2) / 2  
}
```

```
public float Media(float n1, float n2, float n2) {  
    return (n1 + n2 + n3) / 3  
}
```



O que acontece quando o método **sobrecarregado** é acessado?

```
var Calc= new Calculadora();  
Calc.Media()
```

Pode-se ainda incluir informações sobre cada sobrecarga!

▲ 1 de 2 ▼ float Calculadora.Media(int n1, int n2)

- byte
- Byte
- ByteArrayContent
- Calc
- Calculadora
- CancellationToken
- CancellationTokenRegistration
- CancellationTokenSource

(variável local) C

Sobrecarga de Métodos em C# (Overloading)

- A sobrecarga vai além da quantidade de parâmetros!
 - A alteração da ordem de tipos na lista de parâmetros já pode constituir uma sobrecarga

```
public void ListaCliente(String nome, int Idade) {  
    System.out.println(nome + " - " + Idade);  
}
```



```
public void ListaCliente(String nome, float Altura) {  
    System.out.println(nome + " - " + Altura);  
}
```

Sobrecarga de Métodos em C# (Overloading)

- Por que utilizar sobrecarrega?
 - Separar funcionalidades similares em diferentes métodos apenas com base na quantidade de parâmetros pode criar um código extremamente confuso e de difícil manutenção.
 - Quando utilizamos a sobrecarga, os métodos têm o mesmo nome e isso facilita a manutenção e entendimento acerca do que cada método faz e o motivo dos diferentes parâmetros.

Existem Construtores Sobrecarregados?



Construtores podem ser sobrecarregados

- Como os métodos, os construtores também podem utilizar a sobrecarga;
- Tipo e número de parâmetros são utilizados para diferenciar os construtores sobrecarregados;
- A regra se mantém, construtores não possuem tipo de retorno.

Construtores podem ser sobrecarregados – Na Prática!

- Tomemos o nosso exemplo TesteC

Neste exemplo, a classe TesteC possui um construtor que não recebe qualquer parâmetro

```
class TesteC {  
    int numero = 20;  
    TesteC() {  
        valor=numero;  
    }  
}
```



Criar sobrecarga

```
class TesteC {  
    int numero = 20;  
    TesteC() {  
        valor=numero;  
    }  
    TesteC(int num) {  
        valor=num;  
    }  
}
```

E agora? O que acontece quando sobrecarregamos?

Construtores podem ser sobrecarregados – Na Prática!



- Analise o código e descreva o que ocorre

```
class TesteC {  
    int numero = 35;  
    TesteC() {  
    }  
  
    TesteC(int num) {  
        numero=num;  
    }  
}
```

```
public class AulaPOO{  
    public static void main(String []args) {  
        TesteC T1 = new TesteC();  
        System.out.println(T1.numero);  
  
        TesteC T2 = new TesteC(2);  
        System.out.println(T2.numero);    }}
```

O que é **Herança** e como ela pode nos ajudar na reutilização de **código**?



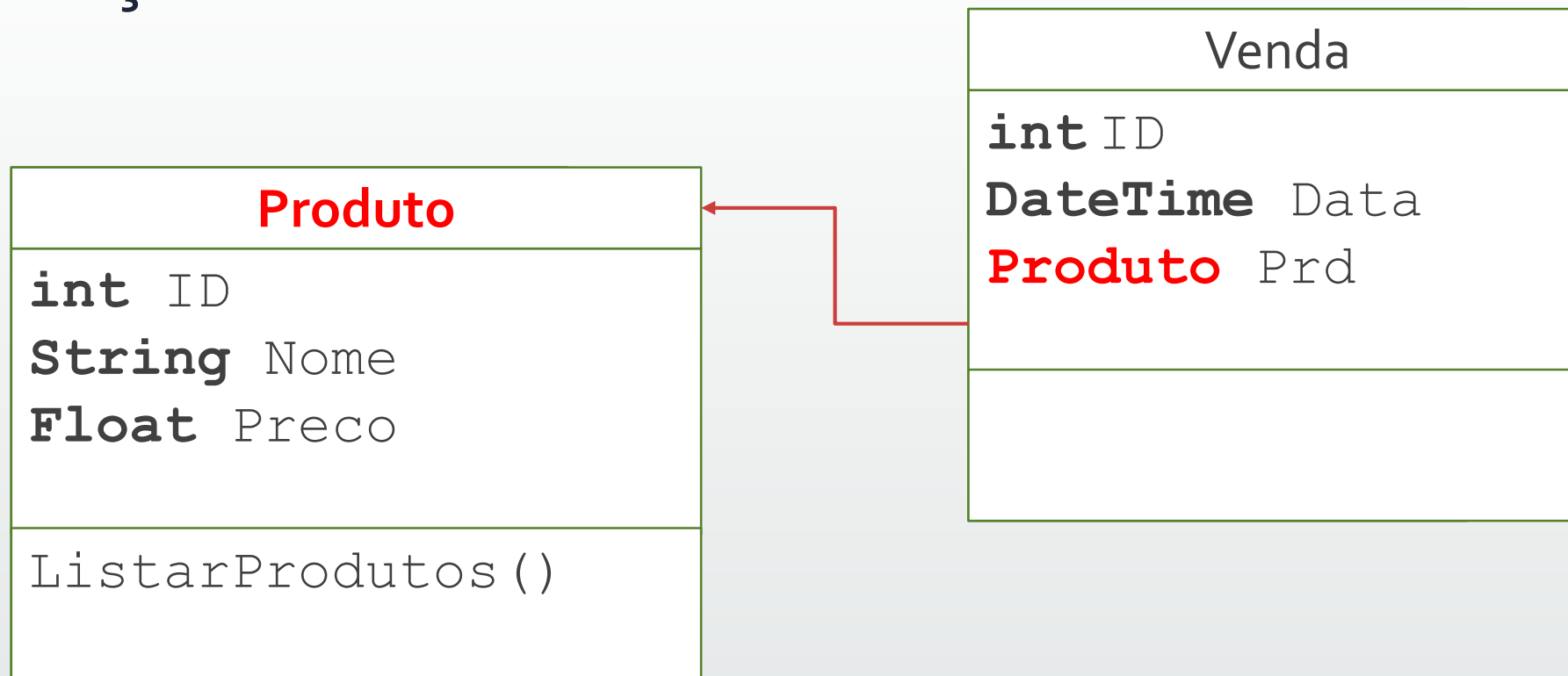
Reutilização

- Uma das mais importantes vantagens da orientação a objeto é exatamente sua capacidade de facilitar a reutilização de código.
- O aproveitamento de classes, atributos, métodos **que já estão testados e funcionando** é o que torna este paradigma tão relevante do desenvolvimento de software.

Os Mecanismos da Reutilização

- Existem dois mecanismos de básicos de relacionamento entre entidades / objetos
 - **Composição** (ou Delegação)
 - Consiste em utilizar um objeto como um “tipo” para o atributo de uma outra classe.
 - **Herança**
 - Quando criamos uma classe que mantem um ligação direta (*extend*) com outra classe.

Composição



- Neste exemplo, o objeto **produto** é utilizado como um atributo na classe **venda**

Herança

- Um recursos muito importante da orientação a objeto;
- Consiste em um mecanismo pelo qual um objeto **adquire todas as propriedades e comportamentos** de um outro objeto (pai);
- Objetos são formados por classes, assim, estas classes precisam saber se **[herdaram algo?]** e **[de quem?]**

Herança

- A herança deve ser utilizada quando se deseja especializar uma classe e quando existe a relação de “é um” entre a subclasse e a superclasse.
- Exemplos :
 - Aluno é uma pessoa
 - Professor é uma pessoa

Herança em C#

- A sintaxe para criação de herança em C# é simples.
 - Basicamente, dizemos que uma classe herda ou “estende” outra, através do uso do caractere “:”
 - Assim, definimos a classe Aluno da seguinte forma:

Modificador **class** **subclass** **herda** **superclasse**



public **class** **Aluno** **:** **Pessoa**

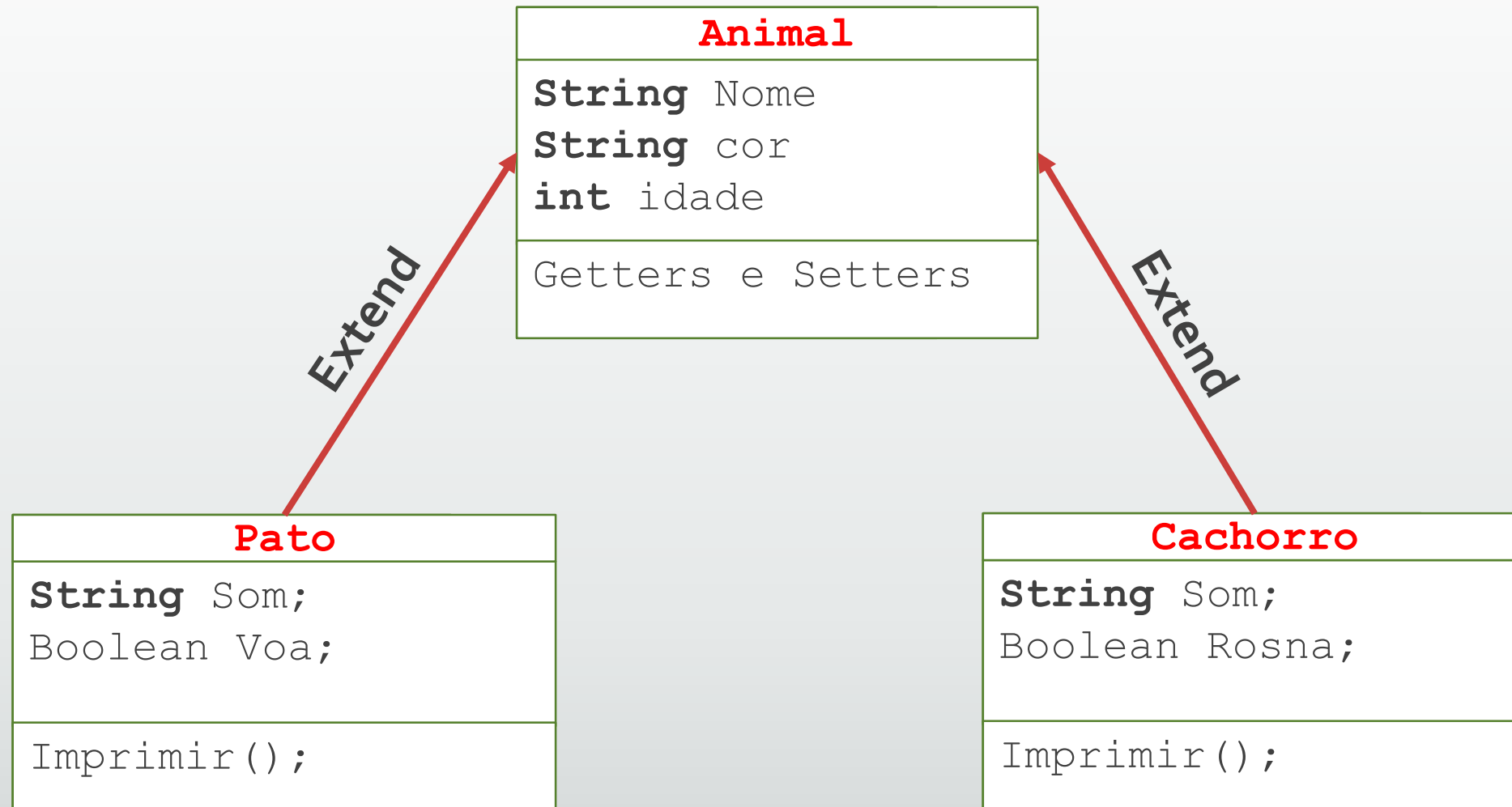
- O Caractere “:” indica que você quer criar uma nova classe que herda a estrutura de uma outra classe já existente.

Herança em C#

```
1. class Super {  
2.     Atributos  
3.     Getters e setters  
4.     Métodos  
5.     .....  
6.     .....
```

```
1. class Sub : Super {  
2.     Atributos  
3.     Getters e setters  
4.     Métodos  
5.     .....  
6.     .....
```

Como podemos codificar este exemplo utilizando herança?



Vamos ver na prática?

- Criando a Super classe Animal

```
1. public class Animal {  
2.     private String? nome;  
3.     private String? cor;  
4.     private int idade;
```

```
5.     Métodos getters e setters aqui!
```


Vamos ver na prática?

- Criando a Sub classe Pato

```
1. public class Pato : Animal {  
2.     bool voa;  
3.     private String som = "QUEN-QUEN!";  
4.     public void EsteAnimalFaz () {  
5.         Console.WriteLine("Este animal faz: " + som);  
6.     }  
7. }
```

Vamos ver na prática?

- Criando a Sub classe Cachorro

```
1. public class Cachorro : Animal{  
2.     private String som="AU-AU!";  
3.     bool rosna  
4.     public void EsteAnimalFaz () {  
5.         Console.WriteLine ("Este animal faz: " + som) ;  
6.     }  
7. }
```

Vamos ver na prática?

- Agora, vamos utilizar esta estrutura na classe principal (Main())

```
1. public class AulaPOO {  
2.     public static void main(String[] args) {  
3.         Pato p = new Pato ();  
4.         p.Nme = "Duck";  
5.         p.Cor= "Branco";  
6.         p.Idade = 1;  
7.         // Atributos específicos do Pato  
8.         p.Voa= true;  
9.         //resultado  
10.        Console.WriteLine ("Nome: " + p.nome);  
11.        Console.WriteLine ("Cor: " + p.cor);  
12.        p.EsteAnimalFaz ();  
13.    }}
```

Vamos ver na prática?

- Agora, vamos utilizar esta estrutura na classe principal (Main())

```
1. public class AulaPOO {  
2.     public static void main(String[] args) {  
3.         Cachorro c = new Cachorro();  
4.         c.nome = "Dog";  
5.         c.cor = "Preto";  
6.         c.idade = 7;  
7.         // Atributos específicos do cachorro  
8.         c.rosna = true;  
9.         //resultado  
10.        Console.WriteLine("Nome: " + c.nome);  
11.        Console.WriteLine("Cor: " + c.cor);  
12.        c.EsteAnimalFaz();  
13.    } }
```

Bons estudos!