Linguagens de Programação

Apresentação

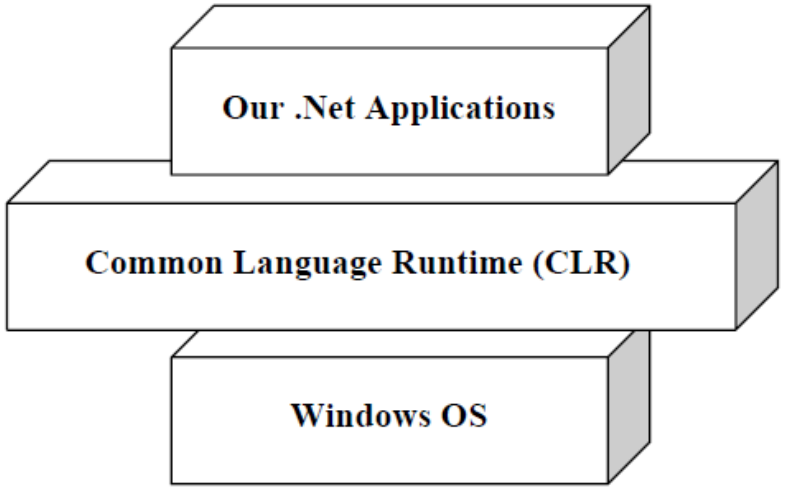
José Martins
Escola Superior de Tecnologia
Instituto Politécnico do Cávado e do Ave
jmartins@ipca.pt

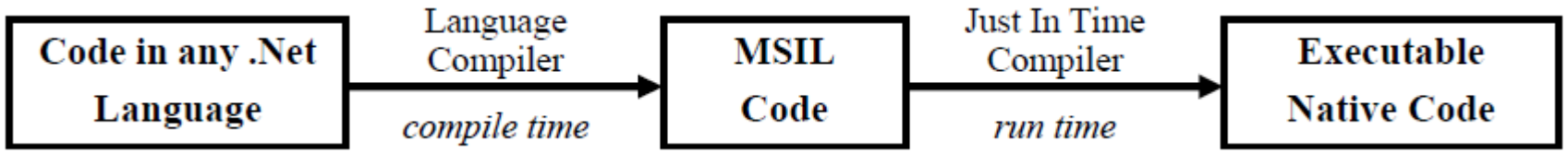
2023/2024

Adaptado de: Prof. Luís Ferreira

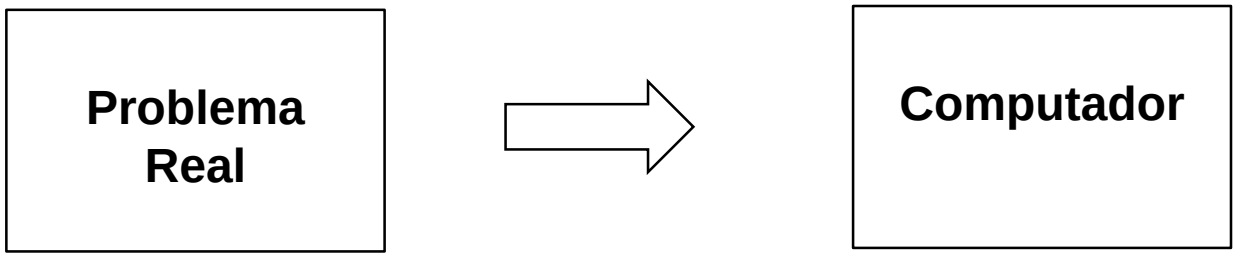
.NET

- *CLR – Common Language Runtime*
- *CLS – Common Language Specification*
- *MSIL – Microsoft Intermediate Language*
- *.NET Framework (FCL, BCL, …)*





Pensar OBJECTOS?

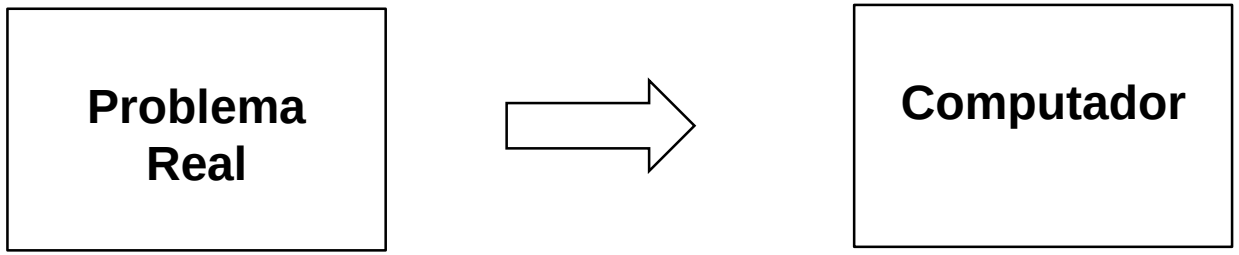


Criar um modelo que representa uma situação real, de forma a que possa ser "interpretado" pelos sistemas computacionais!

Linguagens de Programação

Pensar Objetos

Pensar OBJECTOS?



Criar um modelo que representa uma situação real, de forma a que possa ser "interpretado" pelos sistemas computacionais!

POO: Conceitos básicos

- *Classes/Objetos*
  - *Estado e Comportamento*
  - *Super-Classes e Sub-Classes*
- *Abstração*
- *Encapsulamento*
- *Herança*
- *Polimorfismo*
- *Agregação/Composição*

Pilares da POO

Linguagens de Programação

Pensar Objetos

CLASSES/OBJECTOS

- *Carro*
- *Pessoa*
- *Cão*
- *Gato*
- *Galinha*
- *Cadeira*
- *Empresa*

Linguagens de Programação

Pensar Objetos

Linguagens de Programação

Pensar Objetos

- Coisa material ou abstrata que pode ser percebida pelos sentidos e descrita através das suas características, comportamento e estado.

- Objetivo - Aproximar o mundo digital do mundo real.

Modelo

Classe Caneta

Atributos*:*
Modelo
Cor
Ponta

Métodos:
Escrever
Riscar

Estado:
A escrever

Objeto

Instanciar

Crio um objeto
A partir de uma
classe

cn = nova Caneta

Linguagens de Programação

Pensar Objetos

Instanciar

cn1 = nova Caneta
cn1.cor = “Preto”
cn1.ponta = 0.5
cn1.tampa = Falso
cn1.escrever()

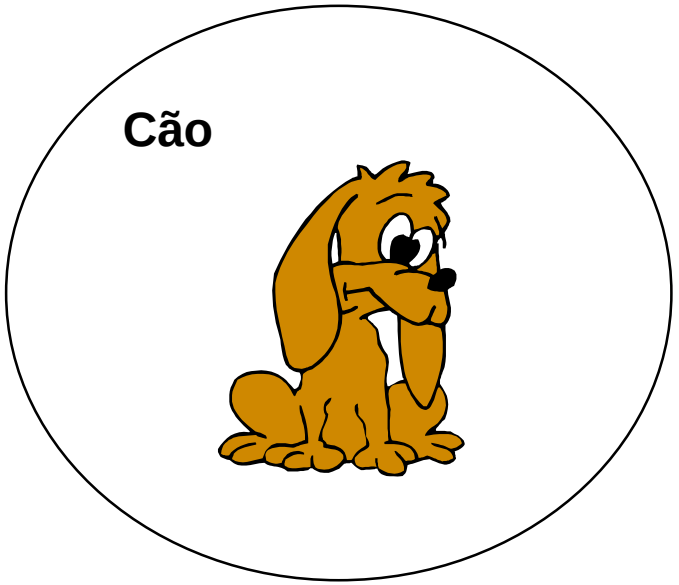
cn2 = nova Caneta
cn2.cor = “Vermelho”
cn2.ponta = 1.0
cn2.tampa =Falso
cn2.riscar()

Objetos diferentes
Pertencem à mesma Classe

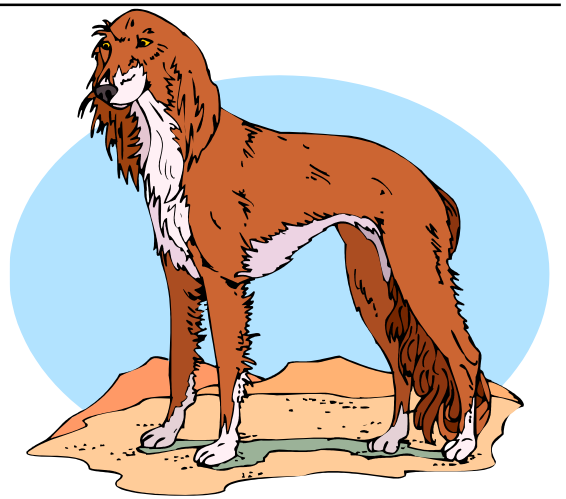
Linguagens de Programação

Pensar Objetos

CLASSES/OBJECTOS



- Nome
- Especie
- Idade
- Tipo de pêlo
- Cor de pêlo
- Cor dos Olhos
- Peso
- ...

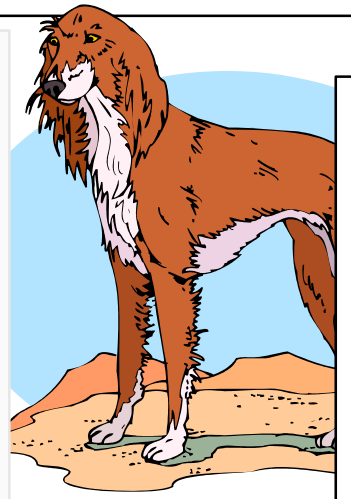


Linguagens de Programação

Pensar Objetos

CLASSES/OBJECTOS

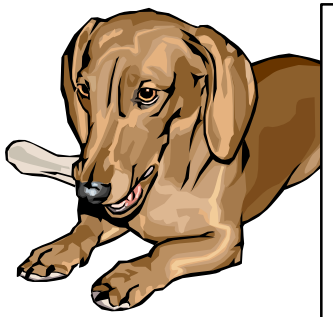
Nome
Especie
Idade
Tipo de pêlo
Cor de pêlo
Cor dos Olhos
Peso
...



Fufu
Também não
10
Longo
Castanho escuro
Azuis
23kg

Cão

Fifi
Sei lá
12
Curto
Castanho
Pretos
13kg

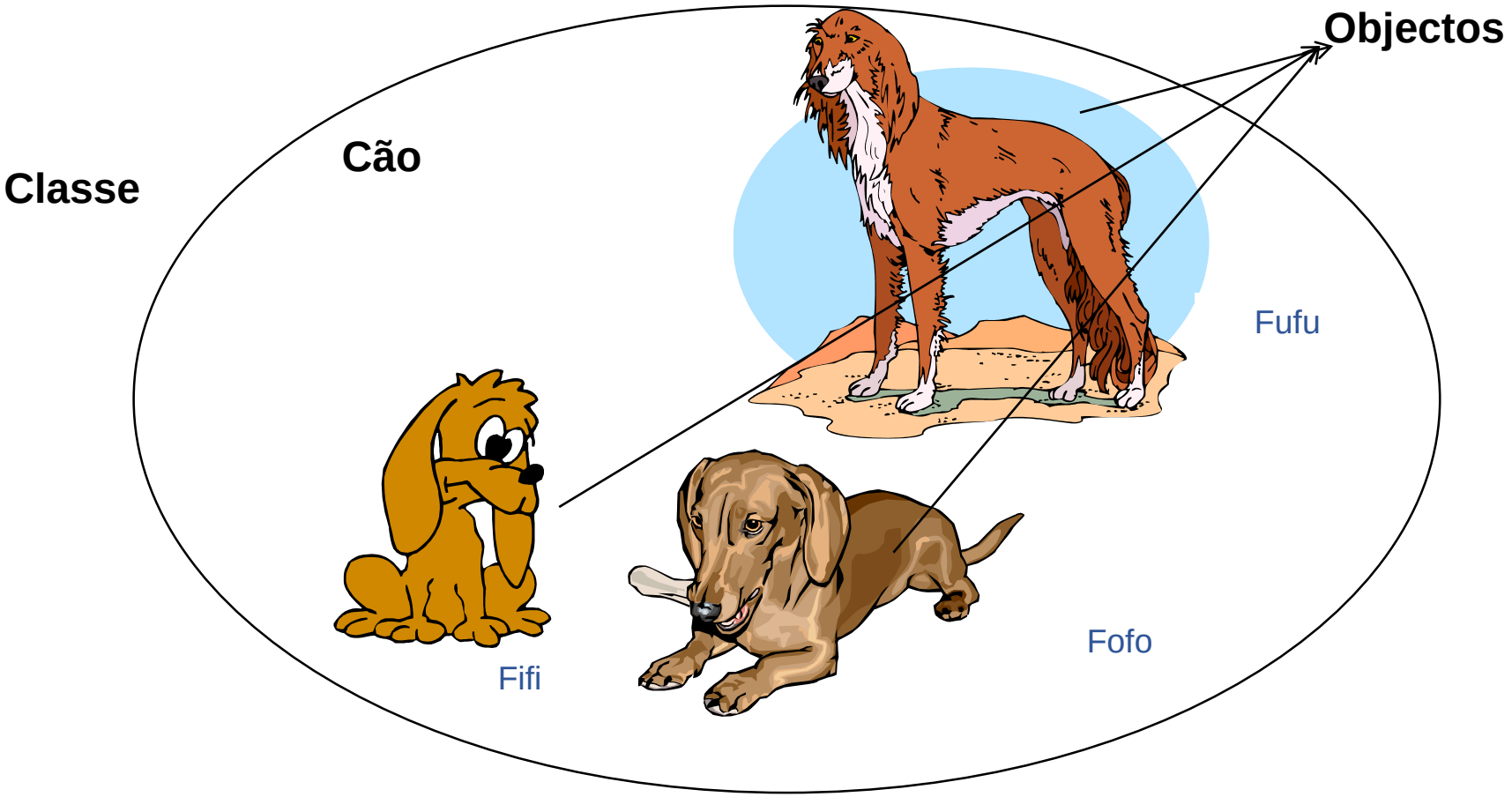


Fofo
Boa
10
Rapado
Cinzento
Castanhos
16kg

Linguagens de Programação

Pensar Objetos

CLASSES/OBJECTOS

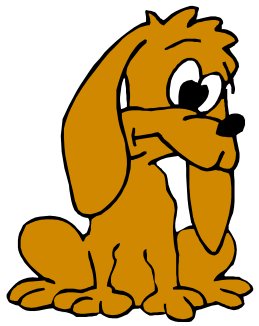


Linguagens de Programação

Pensar Objetos

CLASSE

Cão



Nome

Especie

Idade

Tipo de pêlo

Cor de pêlo

Cor dos Olhos

Peso

...

Ladrar

Comer

Dormir

Brincar

Chatear

Fazer companhia

Atributos ou Propriedades

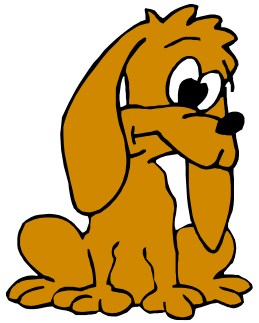
Métodos ou Comportamento

Linguagens de Programação

Pensar Objetos

CLASSES

o cão Ladra



Nome
Idade
Cor de pêlo
Cor dos Olhos
Peso
...
Comer
Dormir
Brincar

Cães e Gatos têm Atributos e Métodos em comum

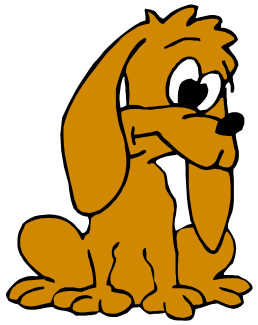
o gato Mia

Linguagens de Programação

Pensar Objetos

CLASSES

SubClasses



Ladrar

Brinca com o osso

Miar

Sobe às árvores

Nome

Idade

Cor de pêlo

Cor dos Olhos

Peso

...

Comer

Dormir

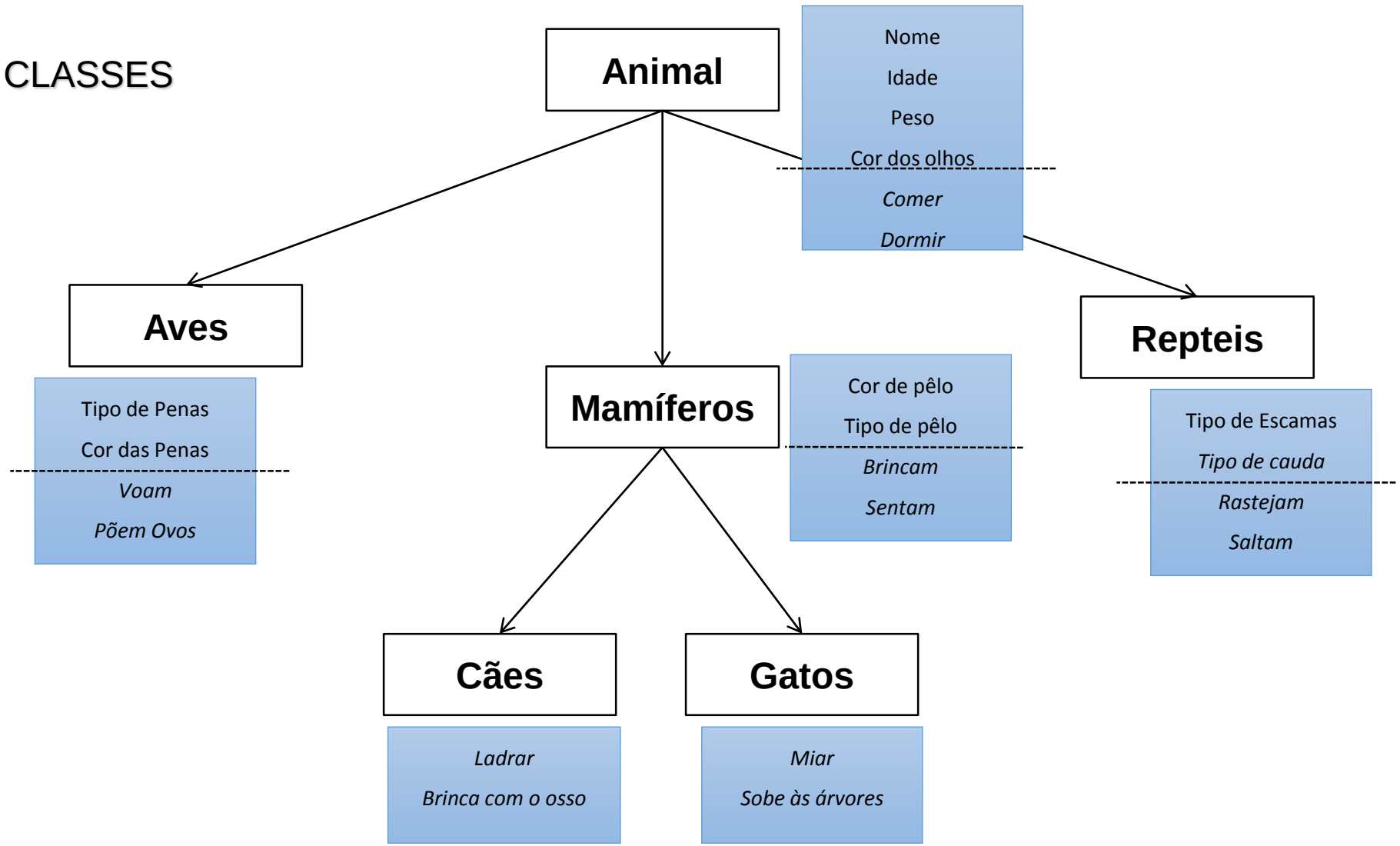
Brincar

ANIMAL

SuperClasse

Linguagens de Programação

Pensar Objetos



Resumo:

- *Classe*
- *Objeto*
- *Atributos (Estado)*
- *Métodos (Comportamento)*
- *Super-Classes*
- *Sub-Classes*

Resumo:

- As características que descrevem um objeto chamam-se atributos.
- As ações que um objeto pode executar são chamadas de métodos ou serviços.
- Ao conjunto de métodos disponíveis de um objeto chama-se interface e define o seu comportamento.

Classes

Em C# uma classe define-se com a palavra reservada **class** seguida do nome que se pretende dar à classe;

Quando definimos as características, funcionalidades e ocorrências que determinado objeto vai ter, estamos na realidade a definir o seu modelo, i.e. a sua classe que contém propriedades, métodos e eventos;

Da definição de uma nova classe resulta um novo tipo de dados como acontece no caso de uma struct;

Uma classe pode ser vista como um modelo ou template a partir do qual podem ser criadas várias instâncias (ou objetos) desse modelo;

Linguagens de Programação

Pensar Objetos

```
/// <summary>
/// This class defines the abstract model of a Student on the context of this program.
/// </summary>
0 references
class Student
{
    // State variables
    // (...)

    // Constructors
    // (...)

    // Destroyer
    // (...)

    // Methods and Properties
    // (...)
}
```

Linguagens de Programação

Pensar Objetos

Chama-se de “estado” ao conjunto atual de dados (valores) internos de uma classe;

As variáveis de estado são também conhecidas por atributos ou campos de uma classe;

```
/// <summary>
/// This class defines the abstract model of a Student on the context of this program.
/// </summary>
6 references
class Student
{
    // State variables
    private int id;
    private string name;
    private float grade;

    // Constructors
    // (...)

    // Destroyer
    // (...)

    // Methods and Properties
    // (...)
}
```

Linguagens de Programação

Pensar Objetos

Em C# uma classe pode ser composta por: variáveis de estado, construtores, destrutor, propriedades, métodos e eventos;

```
/// <summary>
/// This class defines the abstract model of a Student on the context of this program.
/// </summary>
0 references
class Student
{
    // State variables
    // (...)

    // Constructors
    // (...)

    // Destroyer
    // (...)

    // Methods and Properties
    // (...)
}
```

Linguagens de Programação

Pensar Objetos

Construtor:
Tem o nome da Classe

Não devolve nenhum resultado.

Podem existir vários para a mesma Classe.

Contém código para inicialização de objetos. É invocado aquando da criação de um objeto

```
class Student2
{
    // campos
    private static string institution="IPCA";
    private int number;
    private double  mark1;
    private double  mark2;

    // constructor
    public Student2(int n, double m1, double m2)
    {
        number = n;
        mark1 = m1;
        mark2 = m2;
    }

   // métodos
   public void setNumber(int n)
   {
       number = n;
   }
   public void setMark1(double v)...
   public void setMark2(double v)...
   public double CalculateFinalMark()...
}

 Student2 st = new Student2(1234, 12.5, 17.4);
```

Pilares de POO:

- Abstração
- Encapsulamento
- Herança
- Polimorfismo

Linguagens de Programação

Pensar Objetos

ABSTRAÇÃO:

- *Veja-se **abstração** como a tentativa de "esconder" o que "está por trás de"!*
- *A complexidade de um problema é abstraída, tornando só "visível" a parte que interessa tornar pública;*
- *Detalhes que não interessem para determinado contexto não são "divulgados";*
- *O irrelevante num determinado contexto é "ignorado"!*
- *Implementada através de **Public/Private** em atributos e métodos;*

Exemplo(s):

a) *Na folha de vencimento não aparece a cor dos olhos da pessoa!*

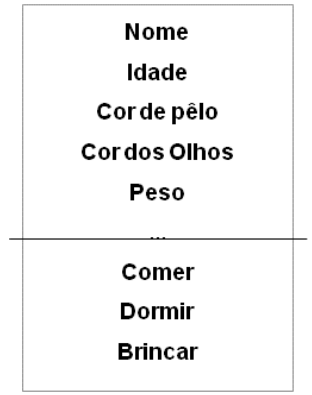
b) *Um carro anda, mesmo sem sabermos que tipo de combustível usa!*

Linguagens de Programação

Pensar Objetos

ENCAPSULAMENTO *(ocultar informações):*

- *O **ENCAPSULAMENTO** protege o estado de acessos indevidos;*
- *O estado de um objeto não pode ser alterado de forma "fácil";*
- *Cada objeto tem o seu próprio estado e o seu comportamento;*
- *Os atributos e métodos estão salvaguardados no interior do objeto;*
- *O estado não deve ser acedido diretamente! É feito através de métodos disponibilizados para esse efeito (**interface pública**);*
- *O estado deve ser **PRIVADO**;*
- ***A forma como um método é implementado não é pública!***



Exemplo:

Qual é o seu vencimento? (não é fácil saber como é calculado)!!!

Linguagens de Programação

Pensar Objetos

ENCAPSULAMENTO:

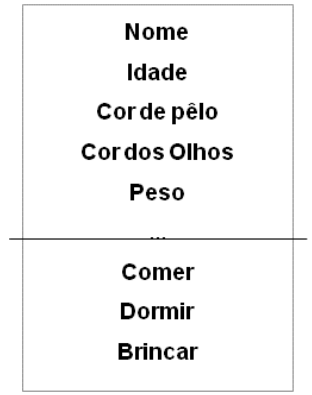
- *O **ESTADO** é o conjunto de valores do objeto num dado instante;*

Exemplo(s):

a) Objecto: *CorDosOlhos* Estado: *preto*

b) Objecto: *NotaExame* Estado: *10*

- *A "utilização" do estado é feita através dos métodos*
  - *Qual a cor dos olhos?* ***CorDosOlhos()***
  - *Que nota teve no exame?* ***NotaExame()***

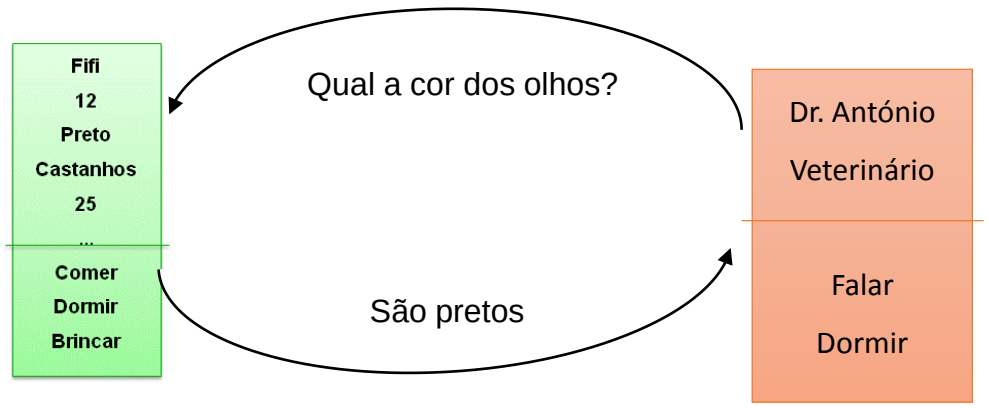


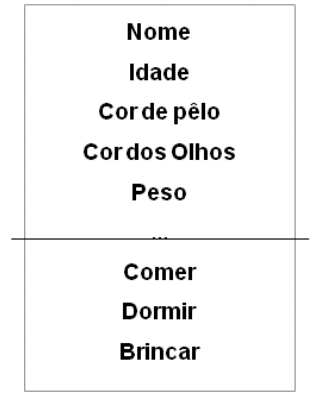
Linguagens de Programação

Pensar Objetos

ENCAPSULAMENTO:

- *Os objetos interagem entre si através de mensagens (**métodos**);*
- *Um objeto necessita dos dados de outro para realizar determinada operação;*
- *Uma mensagem pode ou não alterar o estado de um objeto;*





Linguagens de Programação

Pensar Objetos

ENCAPSULAMENTO (exs):

TELEVISÃO e DVD

- *Cada aparelho tem as suas funções!*
- *A integração entre eles ocorre pela ligação da saída de um objecto à entrada do outro!*
- *Com isto o DVD pode usar a TELEVISÃO para reproduzir imagens.*

TELEFONE:

- *Quando telefono não sei qual o equipamento que o destino possui!*

ENCAPSULAMENTO (exs):

Quando se acelera num carro envia-se uma “mensagem” ao motor do carro usando o acelerador e o carro “sabe” que tem que acelerar.

Não é necessário saber como é feita a aceleração no motor...pé no acelador e ele anda, a implementação de como é feita a aceleração esta encapsulada do utilizador.

Linguagens de Programação

Pensar Objetos

HERANÇA:

- *A classe OBJECTO é a classe base de todas as outras;*
- *Um objecto tem sempre um ASCENDENTE; No mínimo a classe*

Q: Mobiliário → Sala Aulas → Secretária → Aluno

- *A classe Filho herda todas as propriedades da classe Pai e pode estender novas propriedades;*
- *A Herança aumenta a capacidade da **REUTILIZAÇÂO**;*

Exemplo(s):

a) *Animal → Mamífero → Cão*

b) *Mobiliário → Sala Aulas → Secretária → Gaveta*

HERANÇA:

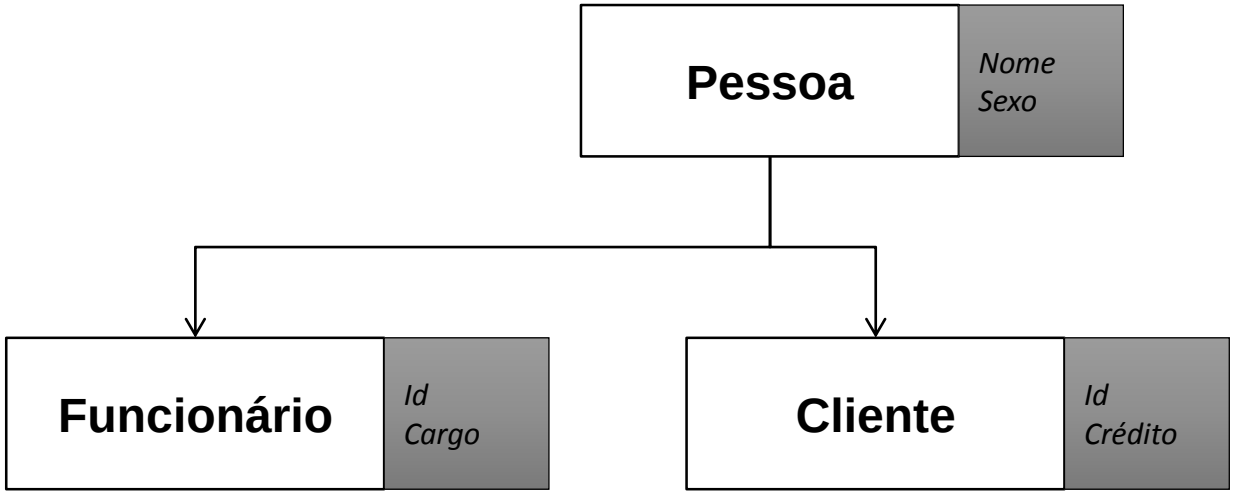
- *Simples*
- *Múltipla*

Linguagens de Programação

Pensar Objetos

HERANÇA:

- *Simples*

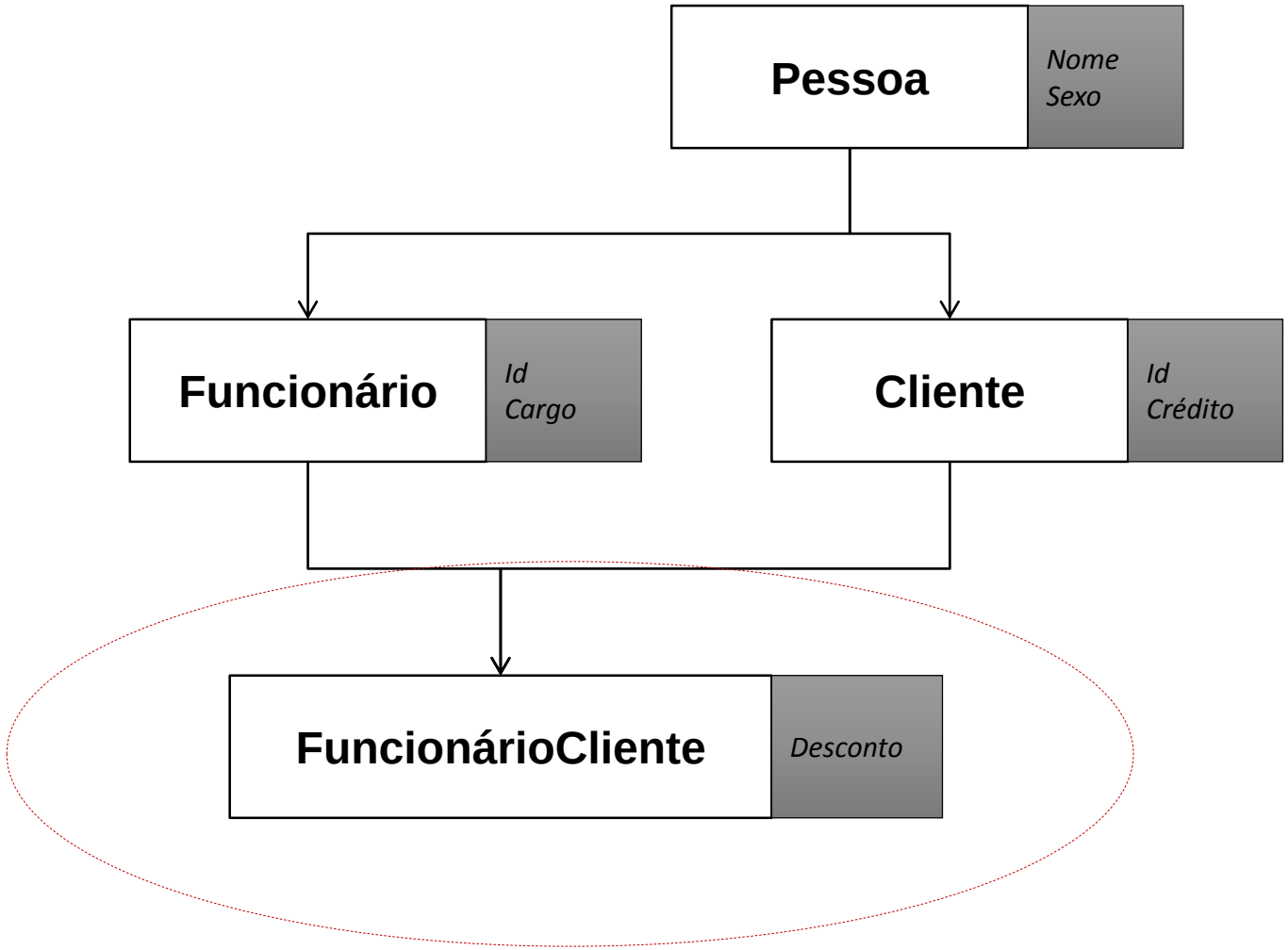


Linguagens de Programação

Pensar Objetos

HERANÇA:

- *Múltipla*



EXERCÍCIO:

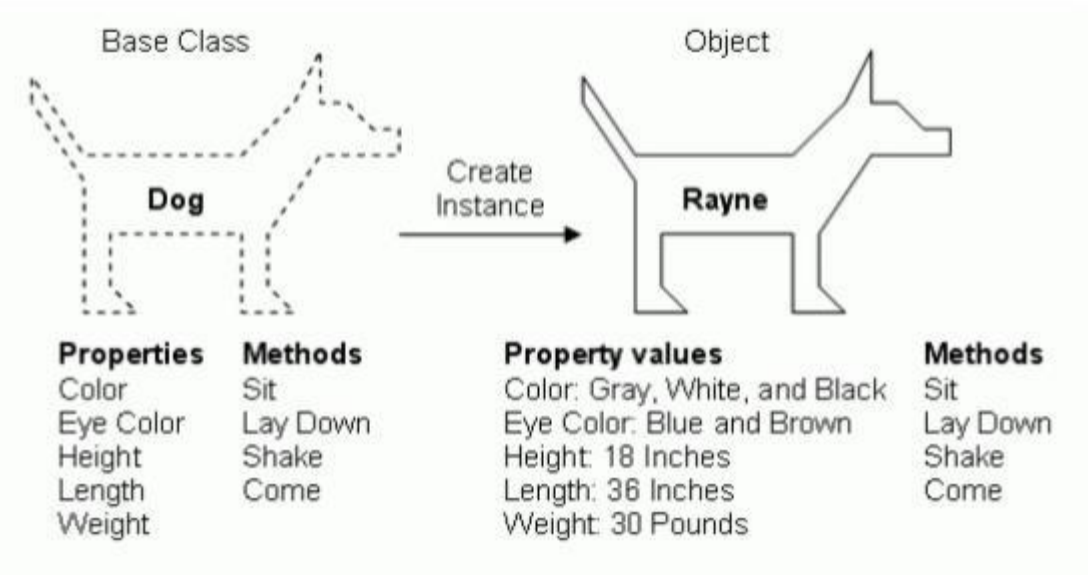
- *Classes e Hierarquia de Classes para:*
  - *Veículos*
  - *Mobiliário*
  - *Pessoas*

- Crie a class Dog e simule a existência de vários objetos desse tipo (especifique a nova classe em papel, e simule a criação de vários objetos indicando o respetivo valor e estado;
- Especifique a class Student e simule a existência de vários objetos desse tipo;

Linguagens de Programação

Exercícios

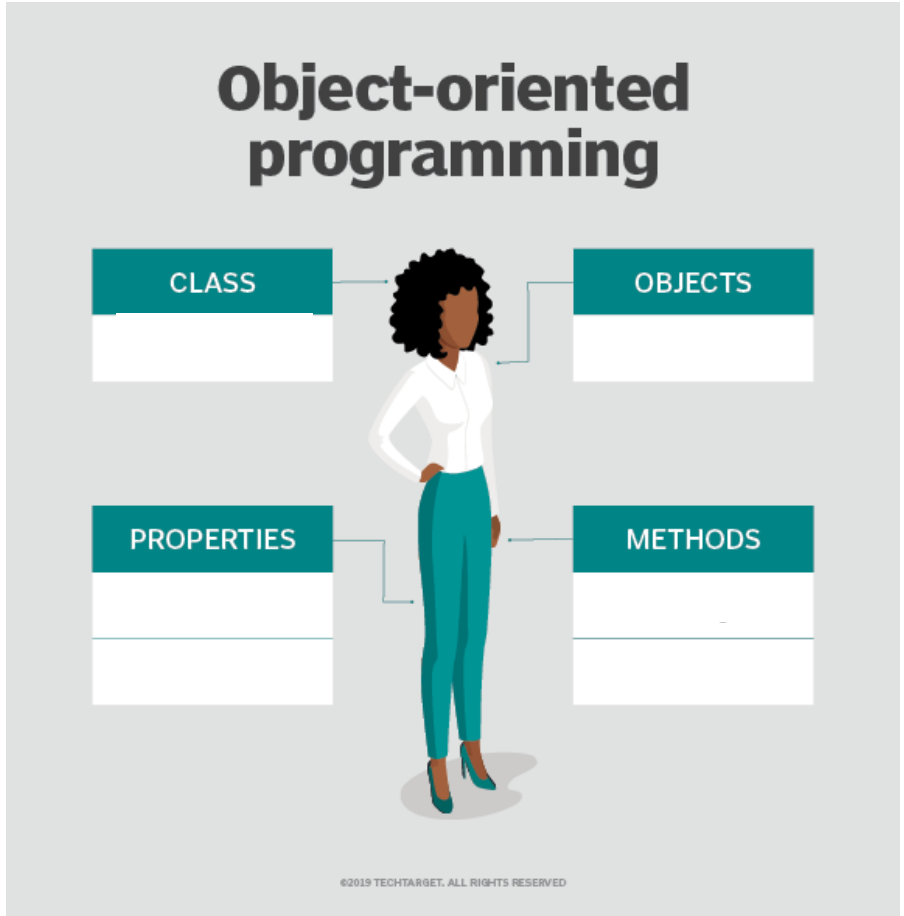
- Crie a class Dog e simule a existência de vários objetos desse tipo (especifique a nova classe em papel, e simule a criação de vários objetos indicando o respetivo valor e estado);



Linguagens de Programação

Exercícios

Alguém que trabalha na receção e/ou RH



Atendendo ao Exemplo do cão
Esquematize o mesmo conceito para:

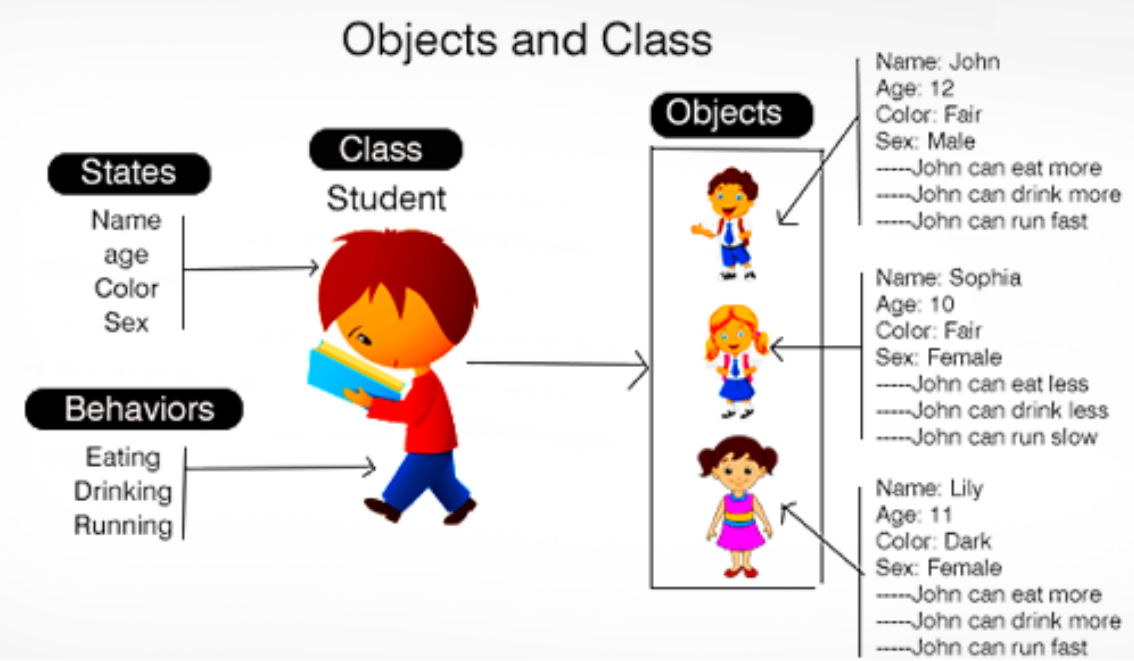
Carro

Livro

Linguagens de Programação

Exercícios

- Especifique a class Student e simule a existência de vários objetos desse tipo;



Polimorfismo:

Classes derivadas de uma mesma superclasse podem invocar métodos que têm a mesma identificação (assinatura) mas comportamentos distintos!

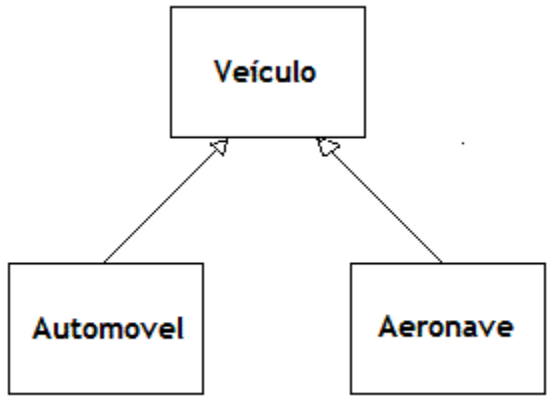
Aplica-se;

- Redefinição de Métodos (*overriding)*
- Métodos abstratos (*abstract*)
- *downcasting/upcasting* - recuperar a referência para o tipo original de um objeto, de modo a obter acesso à sua funcionalidade completa

```
Ref_orig = (Tipo_orig) Ref_upcast;
Automovel a = (Automovel) v
```

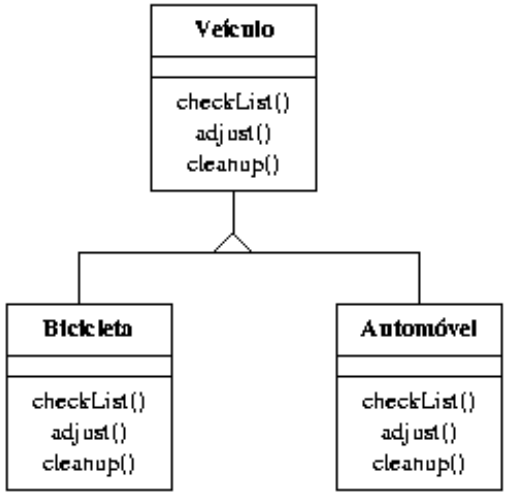
Polimorfismo:

- Polimorfismo em tempo de compilação (Overloading/Sobrecarga);
- Polimorfismo em tempo de execução (Overriding/Sobrescrita);




```
3   public class Veiculo
4   {
5       private string tipo;
6       public string Tipo
7       {
8           get { return tipo; }
9           set { tipo = value; }
10      }
11
12      public Veiculo(string tipoVeiculo)
13      {
14          this.tipo = tipoVeiculo;
15      }
16      public virtual void Mover()
17      { }
18
19      public virtual void Parar()
20      { }
21  }
```

Polimorfismo:



Três métodos na classe *Veículos* são "redefinidos" mais especificamente para Automóveis e Bicicletas:

1. checkList(), verificar o que precisa ser feito (estado) no veículo;
2. adjust(), operações de manutenção;
3. cleanup(), operações de limpeza do veículo.

```
public class Veiculo
   {

        #region Functions

        public virtual void CheckList() { }
        public virtual void Adjust() { }
        public virtual void Cleanup() { }

        #endregion
}
```

Herança: Estruturas | Relações

- Definem a relação entre classes e ajudam a interpretar o papel de cada uma

1. Generalização/Especialização

   Uma classe genérica no topo e sub-classes especializadas

2. Todo-Parte (Agregação)

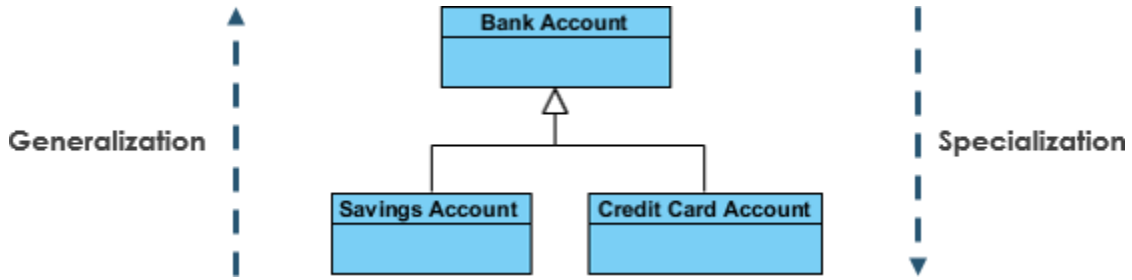
   Uma (sub) classe faz parte de um todo (super-classe)

Estruturas:

1. Generalização – Especialização

- *Generalization relationships are used in class, component, deployment, and use-case diagrams to indicate that the child receives all of the attributes, operations, and relationships that are defined in the parent.*

IBM

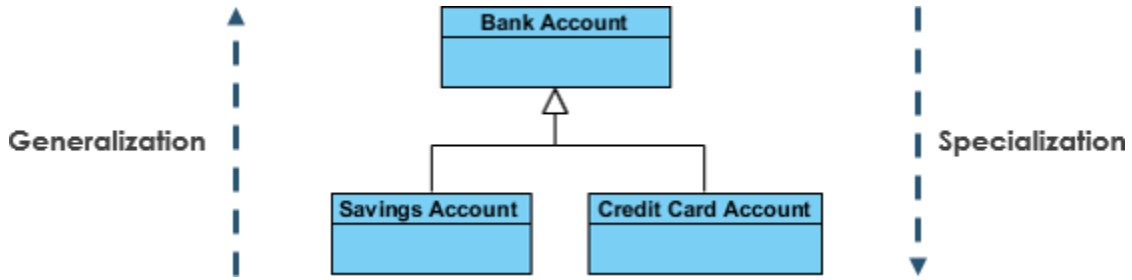


Estruturas:

1. Generalização – Especialização

- *Generalization relationships are used in class, component, deployment, and use-case diagrams to indicate that the child receives all of the attributes, operations, and relationships that are defined in the parent.*

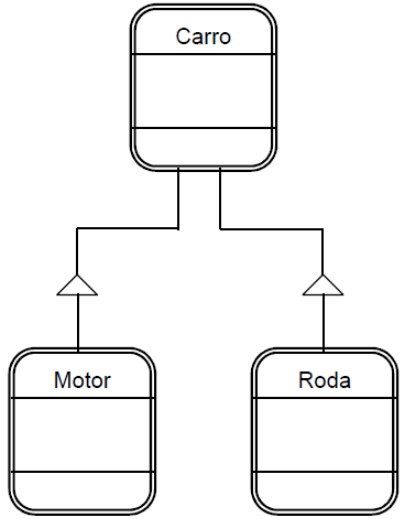
IBM



Estruturas:

1. Todo – Parte

- *Agregação (eventual) e Composição (não dissociados)*
  - *Composição: "está em" "é parte de"*
  - *Agregação: "contém" "consiste em"*
- *Associação: cardinalidade*



Estruturas:

- *Aggregation implies a relationship where the child can exist independently of the parent. Example: Class (parent) and Student (child). Delete the Class and the Students still exist.*

- *Composition implies a relationship where the child cannot exist independent of the parent. Example: House (parent) and Room (child). Rooms don't exist separate to a House*

https://www.visual-paradigm.com

Estruturas:

- *Aggregation implies a relationship where the child can exist independently of the parent. Example: Class (parent) and Student (child). Delete the Class and the Students still exist.*

- *Composition implies a relationship where the child cannot exist independent of the parent. Example: House (parent) and Room (child). Rooms don't exist separate to a House*

https://www.visual-paradigm.com

Exercício:

- Para aceder às necessidades da Biblioteca do IPCA, foi proposto um sistema que deverá atender às seguintes especificações:

  - O registo de utilizadores da biblioteca requer o nome completo. Os utilizadores podem ser alunos, professores ou funcionários;
  - As obras na biblioteca são classificadas em: livros científicos, revistas científicas, revistas informativas, jornais, entretenimento, etc.
  - É importante registar:
    - a língua em que está escrita cada obra
    - o formato em que se encontra a obra
    - os autores da obra
    - as editoras dos vários exemplares, com o ano de edição de cada exemplar.

  Identifique as possíveis classes e objetos e respetivos atributos e métodos

Receita:

1. Identificar as Classes e Objetos (entidades no problema)
2. Identificar as Estruturas e Relacionamentos entre os objetos
3. Identificar os Atributos e os Métodos (verbos) mais importantes

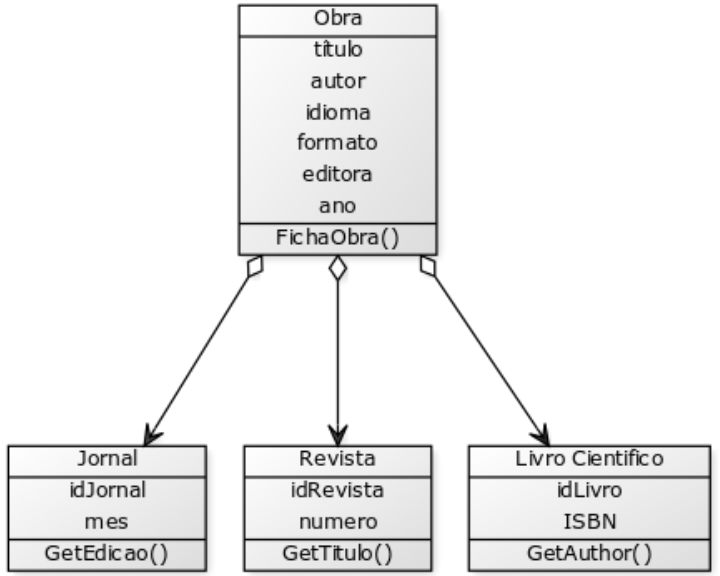
Linguagens de Programação

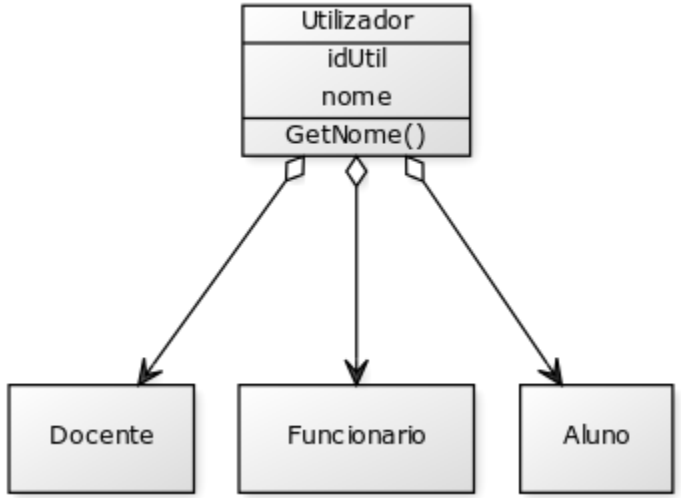
Pensar Objetos

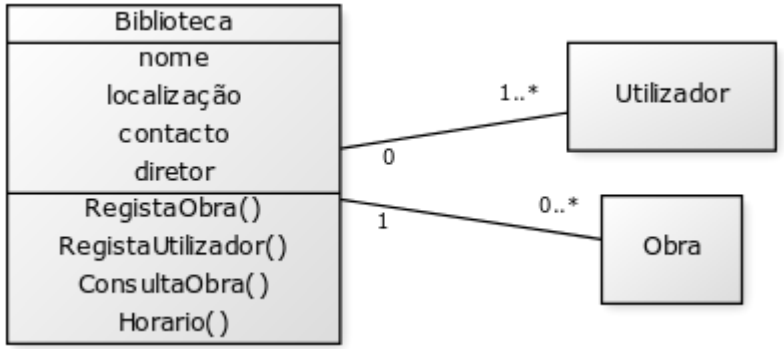
Proposta de Solução:

1. Objetos: não existem

2. Classes: Biblioteca; Utilizador; Obra;





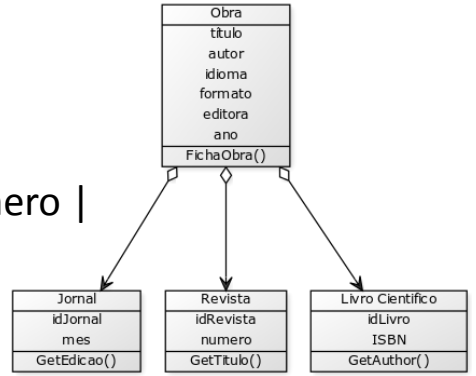


Linguagens de Programação

Pensar Objetos

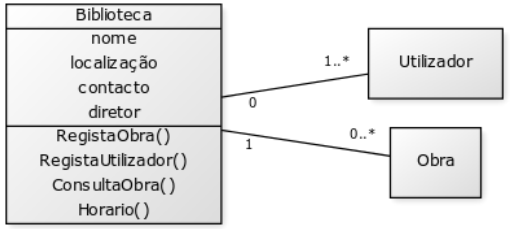
Classe Obra

[Obra |título; autor; idioma; formato; editora; ano |FichaObra()]<>->[Livro Cientifico|idLivro;ISBN | GetAuthor()],[Obra]<>-> [Revista | idRevista; numero | GetTitulo()],[Obra]<>-> [Jornal | idJornal; mes | GetEdicao()]



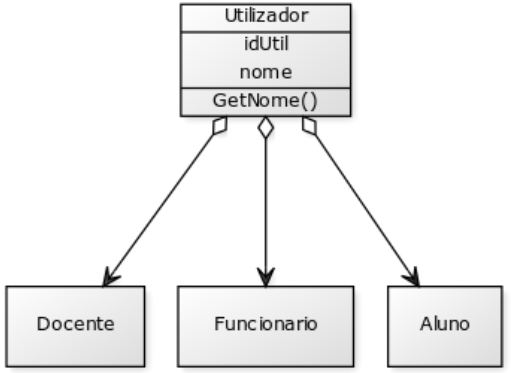
Classe Biblioteca

[Biblioteca | nome; localização;contacto;diretor | RegistaObra();RegistaUtilizador();ConsultaObra();Horario()] 1-0..*[Obra],[Biblioteca]0-1..*[Utilizador]



Classe Utilizador

[Utilizador | idUtil; nome | GetNome()] <>-> [Aluno],[Utilizador]<>-> [Funcionario], [Utilizador]<>->[Docente]



Referências:

- Modelação de Diagramas de Classes

  http://yuml.me

- Generalização, Especialização, Herança

  https://sourcemaking.com/uml/modeling-it-systems/structural-view/generalization-specialization-and-inheritance

- Agregação e Composição

  http://aviadezra.blogspot.pt/2009/05/uml-association-aggregation-composition.html

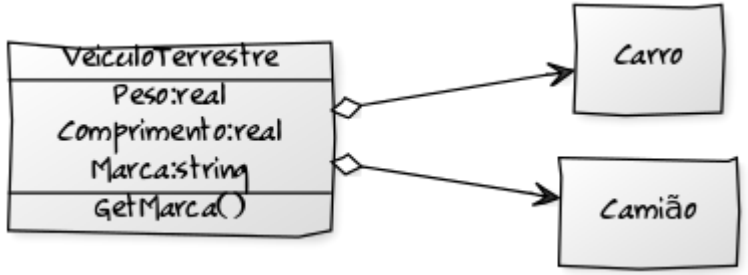
  https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-aggregation-vs-composition/

Linguagens de Programação

Pensar Objetos

Anexo:

- Código YUML

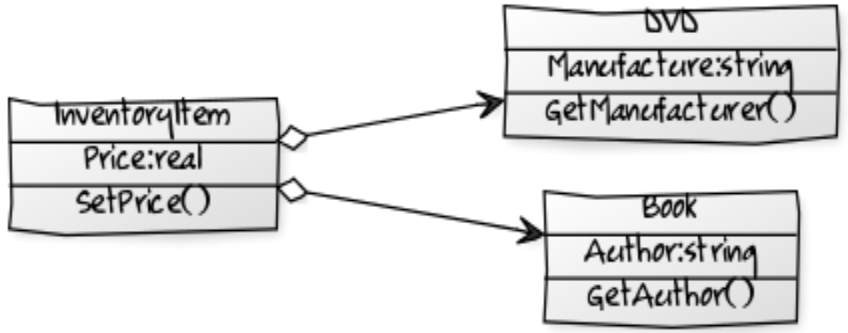


[VeiculoTerrestre|Peso:real;Comprimento:real;Marca:string|GetMarca()]<>->[Camião],[VeiculoTerrestre|Peso:real;Comprimento:real;Marca:string|GetMarca()]<>->[Carro]

https://yuml.me/diagram/scruffy/class/draw

Anexo:

- Código YUML



[InventoryItem |Price:real|SetPrice()]<>->[Book|Author:string | GetAuthor()],[InventoryItem |Price:real|SetPrice()]<>->[DVD | Manufacture:string | GetManufacturer()]

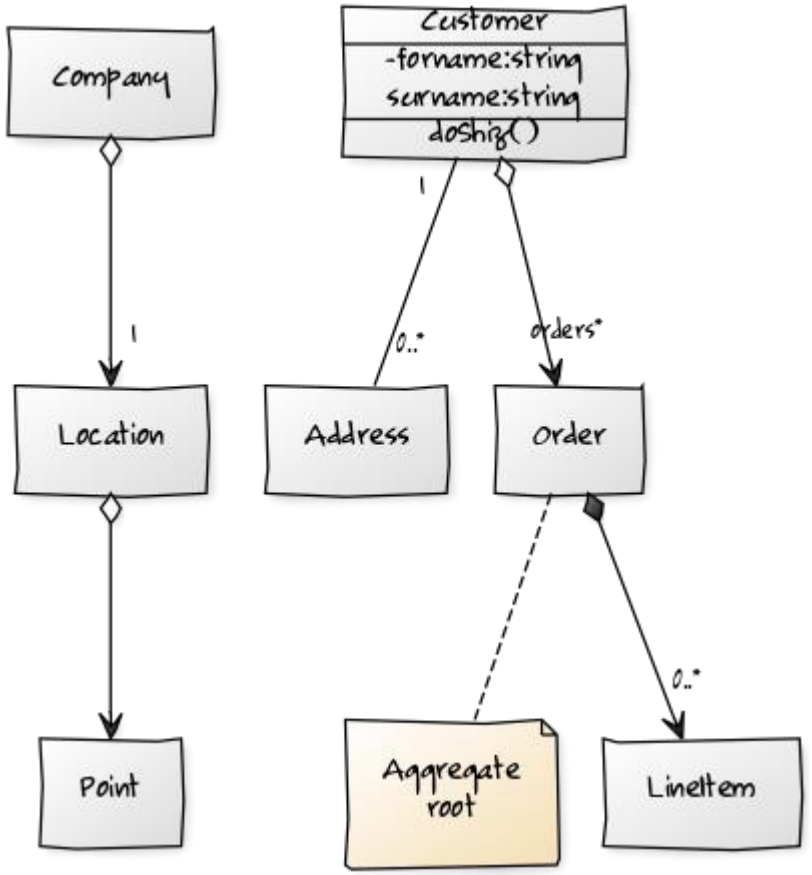
https://yuml.me/diagram/scruffy/class/draw

Linguagens de Programação

Pensar Objetos

Exercício:

- Código YUML
  - Qual?
  - Descrição?

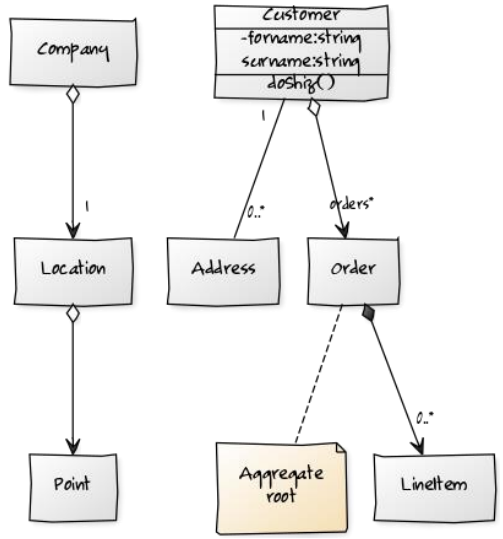


https://yuml.me/diagram/scruffy/class/draw

Linguagens de Programação

Pensar Objetos

Exercício:



- Código YUML
  - [Customer|-forname:string;surname:string|doShiz()]<>-orders*>[Order], [Order]++-0..*>[LineItem], [Order]-[note:Aggregate root{bg:wheat}]
  - [Customer]1-0..*[Address]
  - [Company]<>-1>[Location], [Location]+->[Point]Descrição?

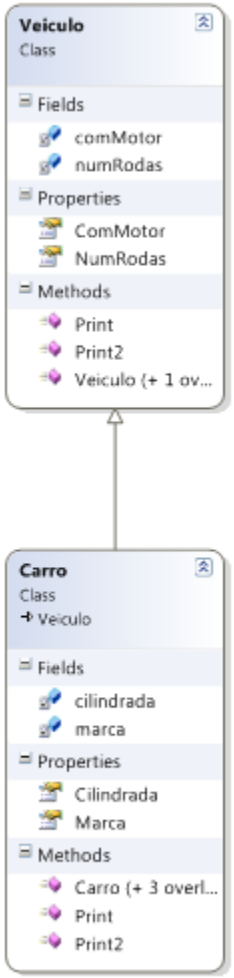
https://yuml.me/diagram/scruffy/class/draw

Herança de Classes

Pilares de OO

- ***Herança***
  - ***Reutilização de Código ➔ Menos erros!***
- ***Encapsulamento***
- ***Abstração***
- ***Polimorfismo***

Herança de Classes



Classe Base (classe Pai)

Classe Derivada (classe Filho)

Herança de Classes

Classe Base (classe Pai)

```
/// <summary>
/// Classe Base
/// Define um Veiculo
/// </summary>
public class Veiculo
{
    int numRodas;
    bool comMotor;

    PROPERTIES

    CONST

    METHODS
}
```

```
/// <summary>
/// Classe Derivada
/// Define uma Carro como sendo Veiculo
/// </summary>
//sealed internal class Carro : Veiculo
public class Carro : Veiculo
{
    #region ESTADO
    double cilindrada;
    string marca;
    #endregion

    PROPERTIES

    CONSTRUTORES

    METODOS
}
```

Classe Derivada (classe Filho)

Herança de Classes

Classe Base (classe Pai)

Classe Derivada (classe Filho)

1. Construtores não são herdados. Cada sub-classe tem de ter o seu construtor!
2. Uso de "*this*" : utilizar método/atributo do objeto corrente
3. Uso de "*base*": utilizar método/atributo do Pai
4. Uso de *new*: alerta compilador do interesse em esconder ("hidding") o método Pai
5. Uso de *virtual*: permite especialização de método (ou *property*) na classe derivada
6. Classe *sealed*: classe que não pode ser derivada

Modificadores de Acesso

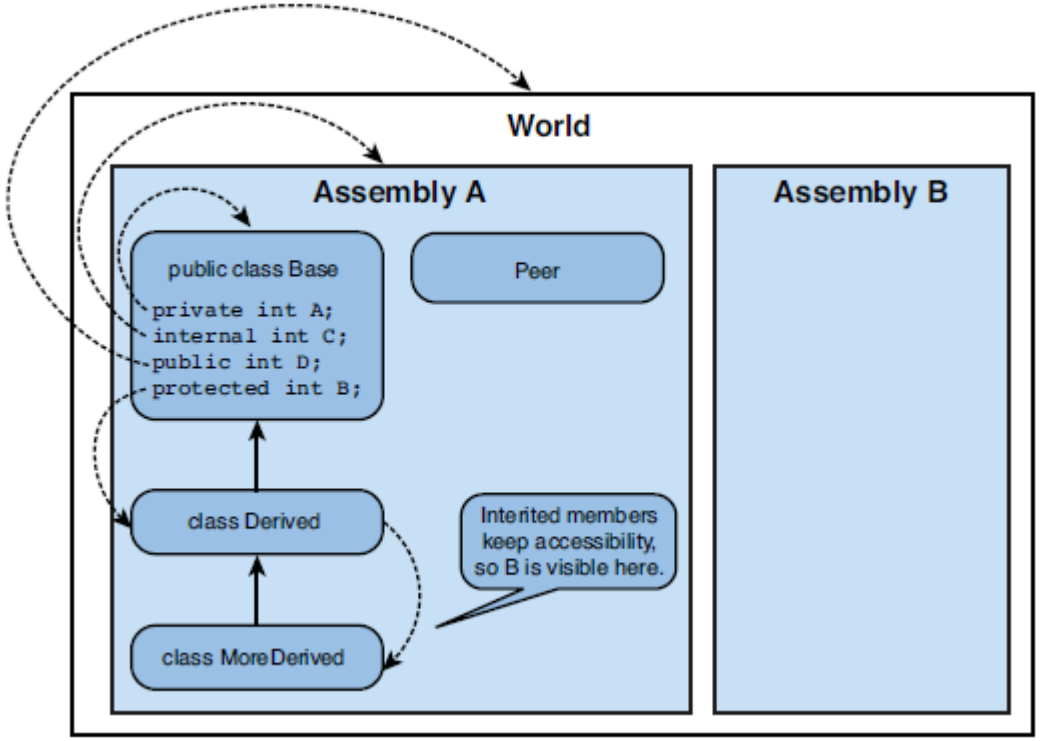
public

protected

internal

private

static



C# Unsleashed – pag 670

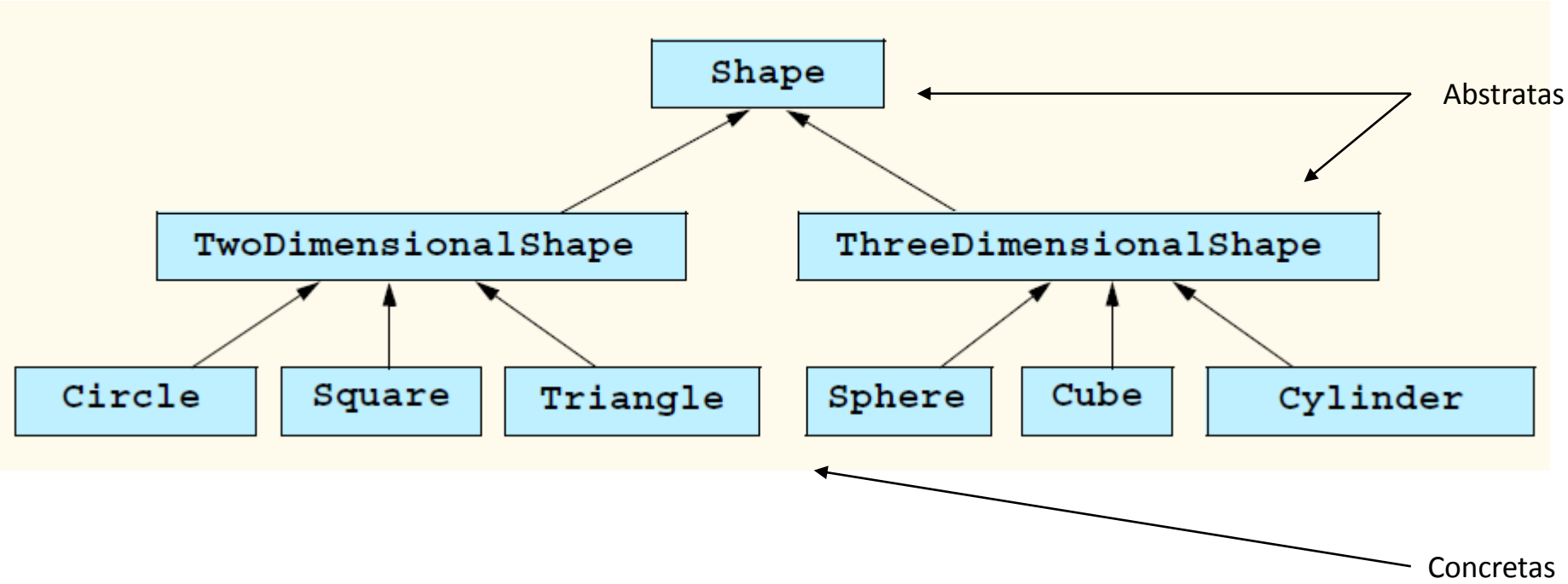
Modificadores de Acesso

- *public* - se vê a classe, vê o método
- *private* - se faz parte da classe, vê o método. Caso contrário não!
- *protected* - mesmo que *private*, alargado aos descendentes (classes derivadas)
- *internal* - *Protected* e restrito ao Assembly corrente
- *static* (class) - Classe que não admite objetos! Tudo é static no seu interior!
- *static* (method) - Pertence à classe. Não pode ser utilizado por nenhum objeto

Classes Abstratas

- Classe base “modelo”
- Não pode ser instanciada
- Classes “incompletas”. Partes por implementar, i.e., abstratas!
- Contém ***abstract Properties*** ou ***abstract Methods***
- Classes que implementam uma classe abstrata diz-se *classe concreta*.
- Permite controlar a implementação de alguns métodos e, controlar o que na herança deve ser implementado!

Classes Abstratas



Outro exemplo: Mamífero

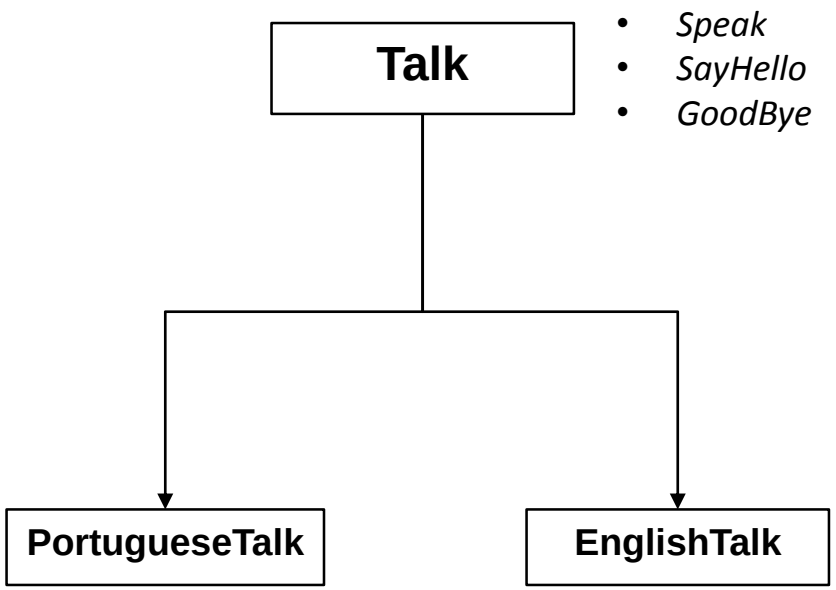
Classes Abstratas

```
/// <summary>
/// Classe abstrata para uma Calculadora
/// </summary>
abstract class Calculadora
{
    public abstract int X
    {
        get;
        set;
    }

    public int Y
    {
        get;
        set;
    }

    public abstract int Soma();

}
```

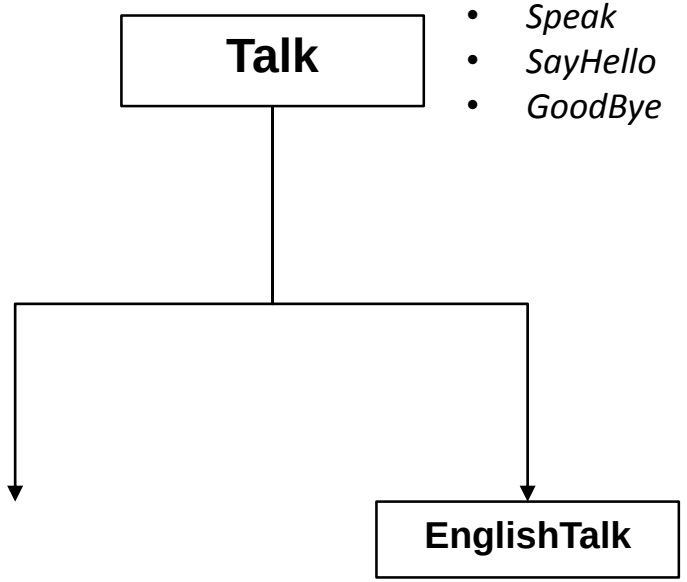
```
/// <summary>
/// Classe concreta: implementa a classe abstracta
/// </summary>
class BoaCalculadora : Calculadora
{
    int x, y;

    public override int X
    {
        get { return x; }
        set { x = value; }
    }

    public override int Soma()
    {
        return x + y;
    }
}
```

Nota: overrride (sobrescrever) é um método em que uma classe derivada
pode declarar outra implementação de um método virtual da classe base

Classes Abstratas



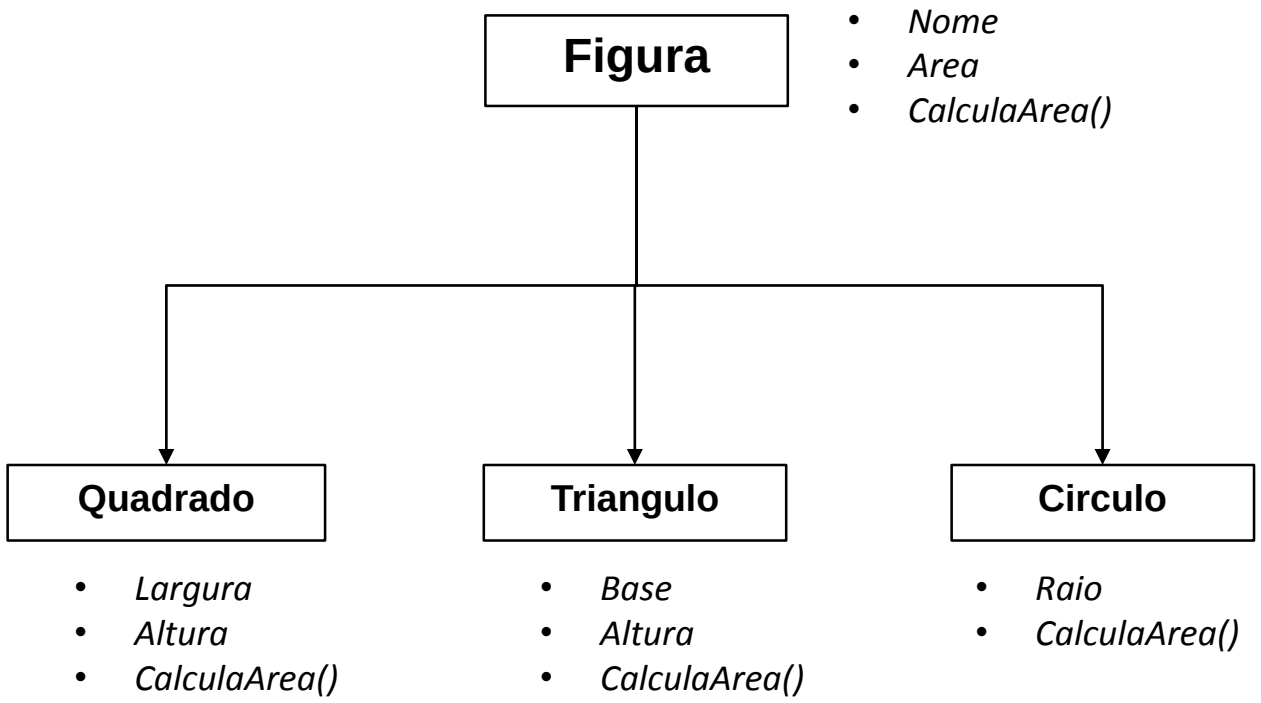
Classes Abstratas



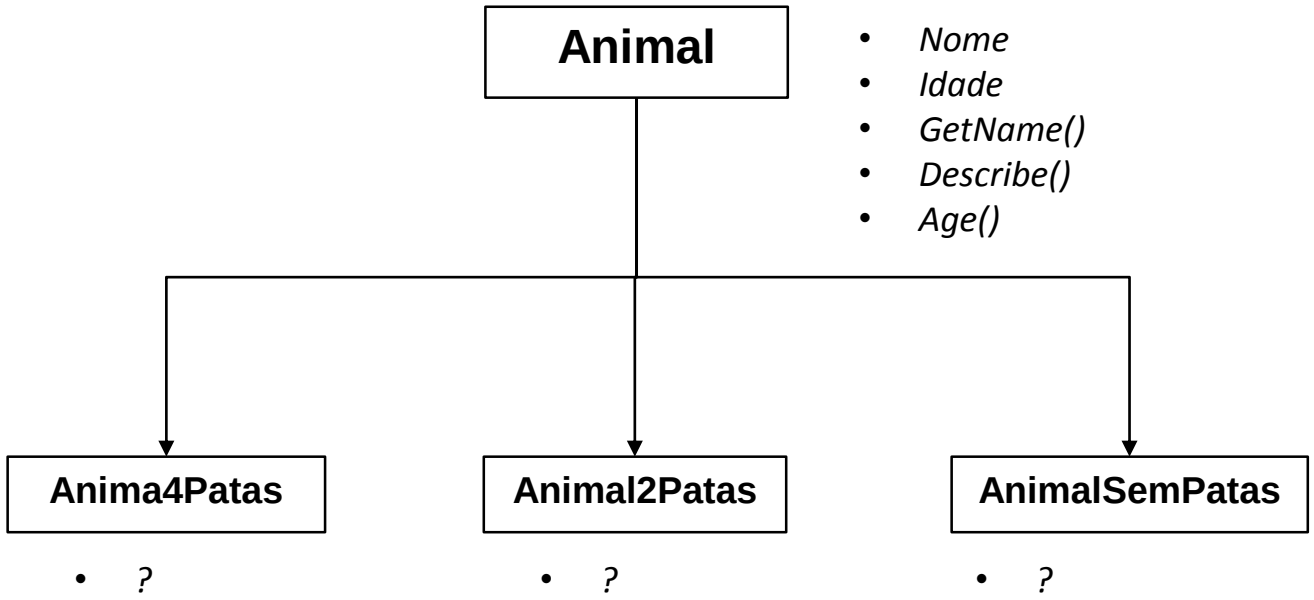
- *Speak*
- *SayHello*
- *GoodBye*

```
public abstract class Talk
   {
        string msg;

        public abstract string Msg
        {
            get;
            set;
        }

        public virtual string SayHello(string msg)
        {
            this.msg = "Hello " + msg;
            return this.msg;
        }

        public abstract string GoodBye();

        public abstract string Speak();
   }
```

Classes Abstratas



Classes Abstratas

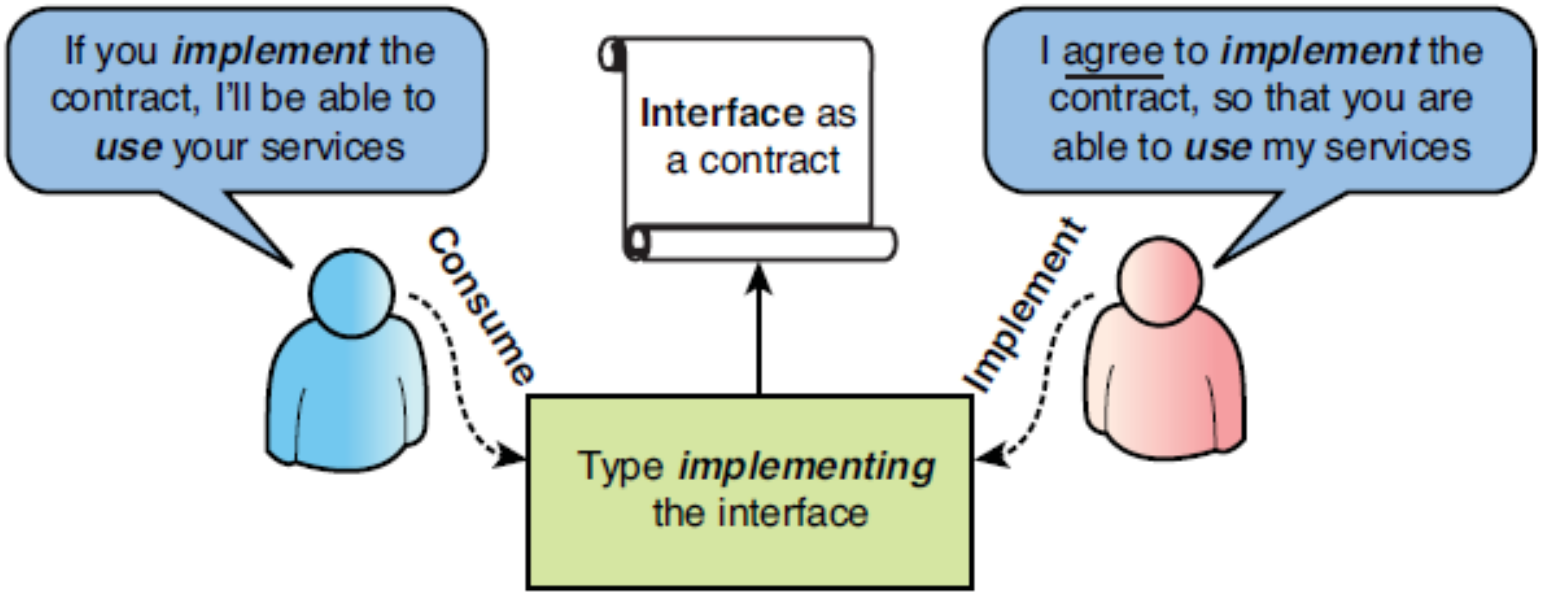


Interfaces

- Espécie de Contracto entre “objetos”
- Garantir que determinado comportamento é suportado
- Forma de suportar uma distribuição de classes e garantir a eficiente integração entre elas.
- Um objeto não precisa de conhecer nada mais sobre outro!

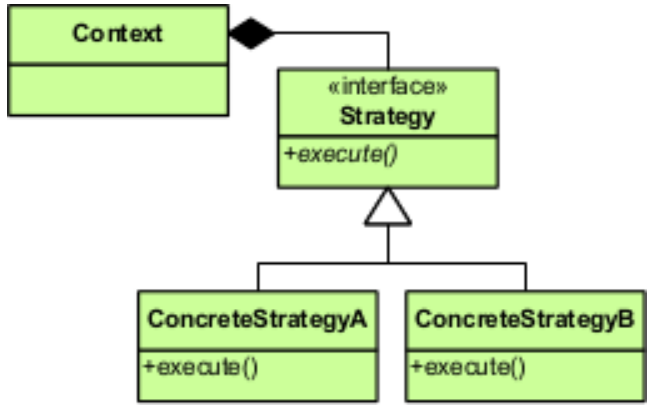
http://programmers.stackexchange.com/questions/108240/why-are-interfaces-useful

Interfaces


If you *implement* the contract, I'll be able to *use* your services
Interface as a contract
I agree to *implement* the contract, so that you are able to *use* my services
Consume
Implement
Type *implementing* the interface

Interfaces

- Definem-se apenas as assinaturas de métodos. Não podem conter atributos nem implementações
- Métodos são *public* por omissão. *Private* não são permitidos!
- Apenas: Métodos, Indexadores, Propriedades, Eventos
- Pode herdar de outros interfaces
- *Strategy Pattern*



http://en.wikipedia.org/wiki/Strategy_pattern

Interfaces

```
/// <summary>
/// Interface para Pessoa
/// </summary>
interface IPessoa
{
    string Nome
    {
        get;
        set;
    }

    string UserID
    {
        get;
        set;
    }

    string GetPass();
    string GetUserId();
}
```

```
class Pessoa :  IPessoa
{
    string nome;
    string userID;
    string passwd;

    public string Nome
    {
        get{return nome;}
        set{nome = value;}
    }

    public string UserID
    {
        get { return userID; }
        set { userID = value; }
    }

    public string GetUserId()
    {
        return "";
    }

    public string GetPass()
    {
        return "";
    }
}
```

Classes Abstratas e Interfaces

- Sempre que uma classe abstrata implementa um interface, tem de declarar os métodos do interface como *abstract*.
- Ver vídeo: http://www.youtube.com/wat

```
/// <summary>
/// Interfrace de Conta Bancária
/// </summary>
interface IConta
{
    string Nome {get; set;}
    double Saldo { get; set; }

    double Levantamento(double x);

}
```

```
/// <summary>
/// Descreve Conta bancária
/// </summary>
abstract class Conta : IConta
{
    string nome;
    double saldo;

    public string Nome
    {
        get{return nome;}
        set{nome=value;}
    }


    public double Saldo
    {
        get{return saldo;}
        set{saldo=value;}

    }
    public abstract double Levantamento(double x);
}
```

Linguagens de Programação

Exercícios

Dada a introdução de dois valores correspondentes ao comprimento e altura de um retângulo, apresente a área respetiva. Faça uso de classes.

```
//Declaração de variáveis
int comprimento, altura, area;

//Obtenção dos dados
Console.Write("Comprimento:");
comprimento = int.Parse(Console.ReadLine());
Console.Write("Altura:");
altura = int.Parse(Console.ReadLine());

//Cálculo
area = comprimento * altura;

//Apresentação do resultado
Console.WriteLine("Área:" + area);
Console.Read();
```

Revisão Polimorfismo

- Muda comportamento
  - Método executado depende da classe do objeto
  - Mesma chamada executa métodos diferentes obj.metodo() vai executar método que foi definido para classe do objeto referenciado por obj
  - Permite executar métodos de subclasses mesmo sem conhecê-las
  - Usado junto com sobrescrita

Revisão Polimorfismo

Polimorfismo

Como definir o comportamento especifico para cada subclasse?

Sobrescrita de método

Cada animal emite sons diferente.
Leão: Rugir
Tigre: Roncar
Cachorro: Latir
Gato: Miar, rosnar
Lobo: Uivar;

Fonte: https://www.ifsp.edu.br/

Revisão Polimorfismo

- Métodos Virtuais são utilizados para implementar o polimorfismo
- A classe que define o método usa a palavra reservada *virtual.* As classes derivadas usam *override.*
  - Empregado
    - public *virtual* double GetSalario() {
    - return salarioFixo;}
  - Vendedor
    - public *override* double GetSalario() {
    - return salarioFixo + 0.05 * totalVendas;}
  - Gerente
    - public *override* double GetSalario() {
    - return salarioFixo + gratificacao; }

Linguagens de Programação

Exercícios

Desenvolva um programa bancário que permita:
1- Levantamento;
2- Deposito;
3- Consulta saldo;
4- Tentativas falhadas;
5- Sair

Linguagens de Programação

Exercícios

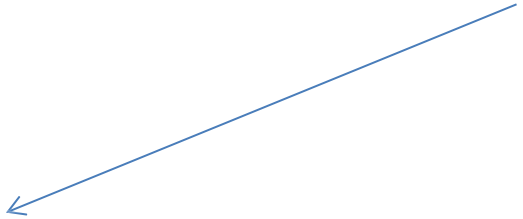
Método Construtor

Crie um método construtor para *carro* com os seguintes parâmetros:
cor, cilindrada, velocidade

Revisão Construtor

```
public class Contato
  {
     private string nome;
     public string Nome       {
        get { return nome; }
        set { nome = value; }
     }
     private string telefone;

     public string Telefone       {
        get { return telefone; }
        set { telefone = value; }
     }
     public Contato(String nome, String telefone)
     {
        this.nome = nome;
        this.telefone = telefone;
     }
  }
```

Construtor da classe, mesmo nome da classe.



Revisão Construtor

- Operador **new** cria uma novo objeto (cria uma instância) a partir de uma classe especificada.
- Ao utilizar o operador new:
  - Novo objeto é alocado dinamicamente na memória, e todas as suas variáveis de instância são inicializadas com valores-padrão predefinidos.
    - null para variáveis objeto
    - 0 para todos os tipos básicos (exceto boolean)
    - false para boolean
- O construtor do novo objeto é ativado;
- Após a execução do construtor, o operador new retorna uma referência (endereço de memória) para o objeto recém criado

Sobrecarga Construtor

Diferentes assinaturas

- Exemplo

```
public Contato(String nome, String telefone)        {
    this.nome = nome;
    this.telefone = telefone;
}
public Contato(String nome)        {
    this.nome = nome;
}
```

Sobrecarga Construtor

Exercício Cálculo da Idade utilizando construtor
Nome, data nascimento,
Calcula a idade

Métodos set e get

- Forma de manter o encapsulamento.
  - Mantem os atributos protegidos (visibilidade private);
  - Cria métodos de acesso public para acessar os atributos.
- Benefícios: fácil manutenção, flexibilidade e extensibilidade.
- Métodos set e get.
  - set :utilizado para modificar uma variável de instância
    - *Sintaxe: public void setNomeVariavelInstancia(Tipo nome) {}*
  - *get:* utilizado para aceder a uma variável de instância
    - *Sintaxe: public Tipo getNomeVariavelInstancia() {}*

Métodos set e get

- Forma clássica através de métodos

```
public class Contato
 {
    private string nome;
    public void getNome(){ return nome;}
    public string setNome(String s){ nome=s;}
 }
```

Métodos set e get

- Outra forma é o uso de propriedades.
  - Cria-se atributos extras chamados de propriedades.

```
public class Contato
  {
     private string nome;
     public string Nome  {
        get { return nome; }
        set { nome = value; }
     }
...
}
```

Métodos sobrecarga

- A sobrecarga permite reutilizar o mesmo nome do método de uma classe, mas com argumentos diferentes. Regras básicas:
  - a sobrecarga de método deve alterar a lista de argumentos;
  - podem alterar o tipo de retorno;
  - podem alterar o modificador de acesso.

Métodos sobrecarga

Assinaturas diferentes

- Exemplo

```
public void inserirContato(Contato c)  {
    listaContato[qtd++] = c;
}

public void inserirContato(String nome, String telefone){
    listaContato[qtd++] = new Contato(nome, telefone);
}
```

This

- Todos os objetos possuem um atributo que é uma referência a ele mesmo

  Usado para acesso a membros do próprio objeto

metros de métodos, por

```
a cor;
or(CorBasica cor) {
or;
```

Exemplo:MétodoConstrutorSobregarga

```
class Pessoa
{
    public string nome;
    public string sobrenome;
    public int anoNascimento;
    public int idade;

    //construtores com assinaturas diferentes
    1 referência
    public Pessoa()...


    // segundo construtor (this referenciar o campo da classe e não do construtor)
    1 referência
    public Pessoa(string nome, string sobrenome, int anoNascimento)
    {
        this.nome = nome;
        this.sobrenome = sobrenome;
        this.anoNascimento = anoNascimento;
        this.idade = Idade();
    }
```