Programação de Soluções Computacionais

Classes, objetos, princípios da orientação a objetos

Otaviano Silvério

Princípios Básicos - Programação Orientada a Objetos

Programação Estruturada

Composição dos Programas:

Um programa é composto por um conjunto de rotinas
A funcionalidade do programa é separada em rotinas
Os dados do programa são variáveis locais ou globais

Princípios Básicos - Programação Orientada a Objetos

Programação Estruturada

Fluxo de Execução:

O programa tem início em uma rotina principal
A rotina principal chama outras rotinas
Estas rotinas podem chamar outras rotinas, sucessivamente
Ao fim de uma rotina, o programa retorna para a chamadora

Princípios Básicos - Programação Orientada a Objetos

Programação Orientada a Objetos

Composição do programa:

A funcionalidade do programa é agrupada em objetos
Os dados do programa são agrupados em objetos
Os objetos agrupam dados e funções correlacionados

Princípios Básicos - Programação Orientada a Objetos

Programação Orientada a Objetos

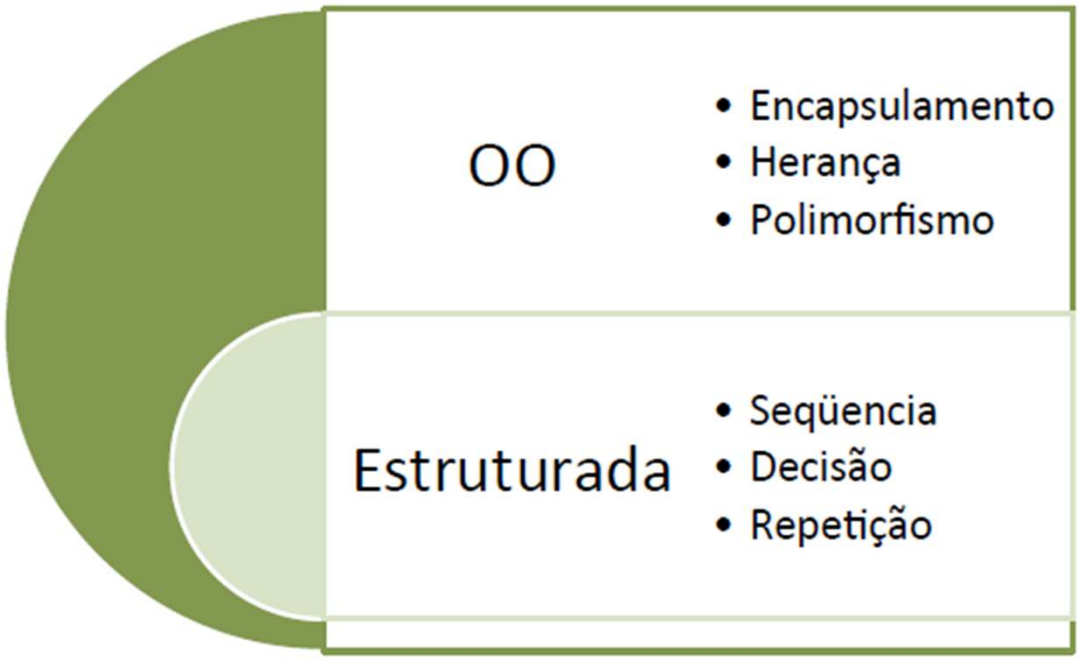
Fluxo de Execução:

Similar ao anterior
Os objetos colaboram entre si para a solução dos objetivos
A colaboração se realiza através de chamadas de rotinas

Princípios Básicos - Programação Orientada a Objetos

“Orientação a Objetos consiste em considerar os sistemas computacionais como uma coleção de objetos que interagem de maneira organizada.”

Princípios Básicos - Programação Orientada a Objetos



Princípios Básicos - Programação Orientada a Objetos

O ser humano se relaciona com o mundo através do conceito de objetos.

Estamos sempre identificando objetos ao nosso redor.

Para isso:

atribuímos nomes
classificamos em grupos – **classes**.

Princípios Básicos - Programação Orientada a Objetos

Definição

Um objeto é a representação computacional de um elemento ou processo do mundo real

Cada objeto possui um conjunto de características e comportamentos

Princípios Básicos - Programação Orientada a Objetos

Definição

Uma característica descreve uma propriedade de um objeto,
ou seja, algum elemento que descreva o objeto

Exemplo de características de um objeto identificado como **carro:**

Cor
Marca
Número de portas
Ano de fabricação e tipo de combustível

Princípios Básicos - Programação Orientada a Objetos

Definição:

Um comportamento representa uma ação ou resposta de um objeto a uma ação do mundo real.

Exemplos de comportamento para o objeto **carro**

Acelerar
Parar
Andar
Estacionar

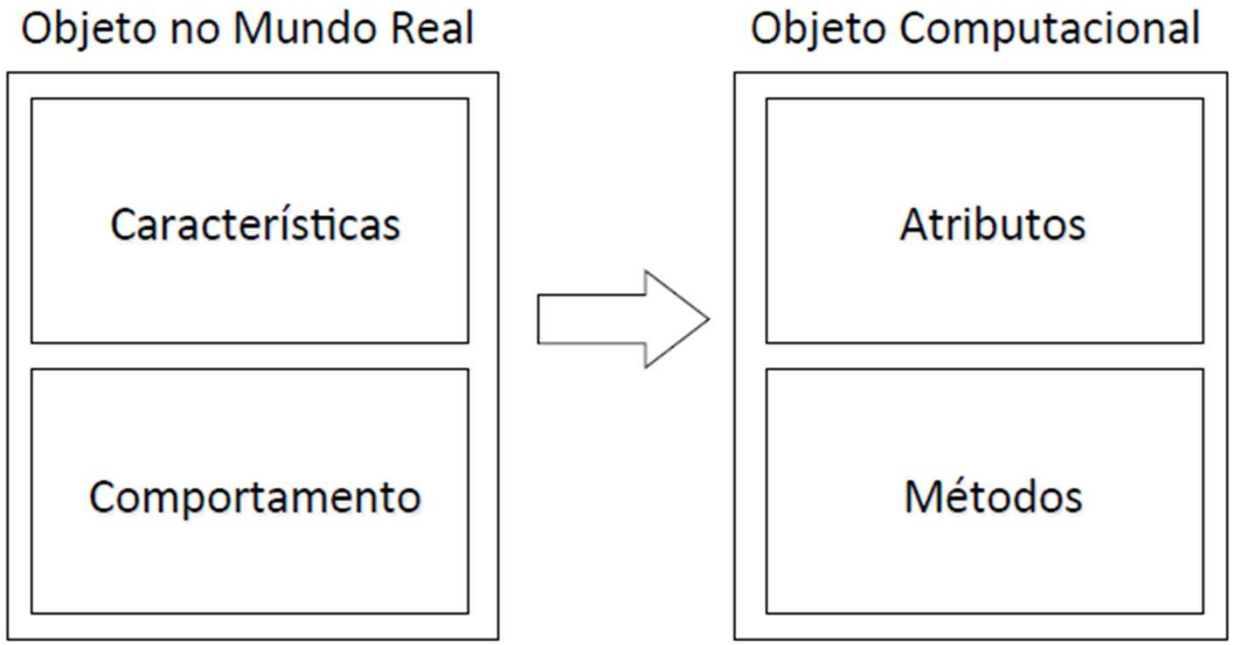
Princípios Básicos - Programação Orientada a Objetos

Exemplos:

- ✓ Cachorros
  características: nome, cor, raça
  comportamentos: latir, correr

- ✓ Bicicletas
  características: marcha atual, velocidade atual
  comportamentos: trocar marcha, aplicar freios

Princípios Básicos - Programação Orientada a Objetos



Princípios Básicos - Programação Orientada a Objetos

Identificar as características e o comportamento de objetos do mundo real é o primeiro passo da programação OO.

Observe um objeto e pergunte:

1. Quais os possíveis características deste objeto e quais estados elas assumem?
2. Quais comportamentos (ações) que ele pode executar?

Princípios Básicos - Programação Orientada a Objetos

“As características que descrevem um objeto são chamadas de atributos”.

Princípios Básicos - Programação Orientada a Objetos

Objetos não são considerados isoladamente

Um processo natural é identificar características e comportamentos semelhantes entre objetos

Objetos com características e comportamentos semelhantes são agrupados em classes

Princípios Básicos - Programação Orientada a Objetos

A unidade fundamental em programação em orientação a objetos (POO) é a **classe**.

Classes contém:

- ✓ Atributos: determinam o estado do objeto;
- ✓ Métodos: semelhantes a procedimentos em linguagens convencionais, são utilizados para manipular os atributos.

Princípios Básicos - Programação Orientada a Objetos

Como representar isso?

Notação gráfica das classes e dos objetos:

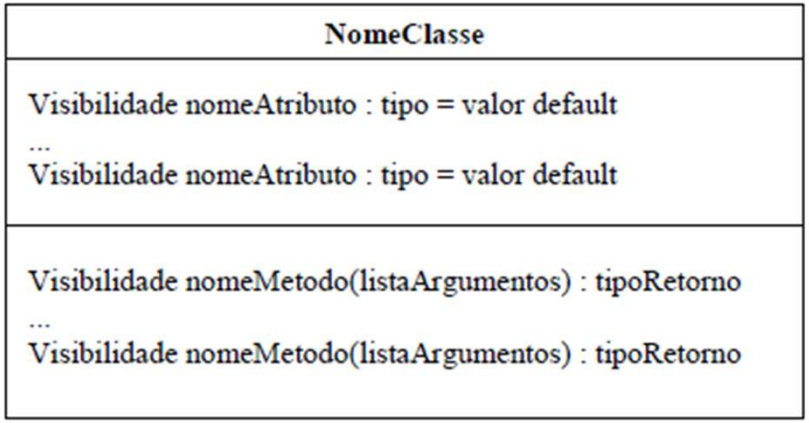
A UML (Unified Modeling Language) é o sucessor de um conjunto de métodos de análise e projeto orientados a objeto (OOA&D). A UML está, atualmente, em processo de padronização pela OMG (Object Management Group).

A UML é um modelo de linguagem, não um método. Um método pressupõe um modelo de linguagem e um processo. O modelo de linguagem é a notação que o método usa para descrever o projeto. Os processos são os passos que devem ser seguidos para se construir o projeto.

Princípios Básicos - Programação Orientada a Objetos

Como representar isso?

Notação gráfica das classes e dos objetos:

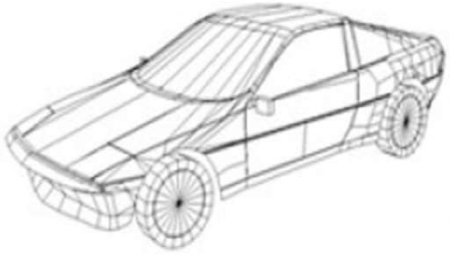


Princípios Básicos - Programação Orientada a Objetos

Carro
Número de Rodas Cor Cor Lateral
Anda Para Acelera Estaciona

Princípios Básicos - Programação Orientada a Objetos

CLASSE →



Tipo: ?

Cor: ?

Placa: ?

Número de Portas: ?

Tipo: Porsche

Cor: Cinza

Placa: MHZ-4345

Número de Portas: 2

← OBJETOS →

Tipo: Ferrari

Cor: Vermelho

Placa: JKL-0001

Número de Portas: 4

Princípios Básicos - Programação Orientada a Objetos

Drawn io - Representação em UML

```
class Carro {
        private:
                float velocidade;
                int ano;
                int modelo;
                int portas;
                char cor;
        public:
                void acelerar(float vel) {
                        if (vel > 0 && vel <= 50) {
                                velocidade = vel;
                        }
                }
        void trocarMarcha(int m) {
                if (m >= 1 && m <= 21) { marcha = m; }
                }
        };
```

Um exemplo simples

- Seja o contexto de automação bancária
- Podemos identificar as seguintes classes
  - cliente
  - agência
  - conta
  - conta corrente
  - conta poupança
  - dentre outras

Um exemplo simples – conta corrente

- Atributos
  - número
  - agência
  - saldo
- Métodos
  - depositar
  - sacar
  - consulta saldo

Um exemplo simples – conta corrente

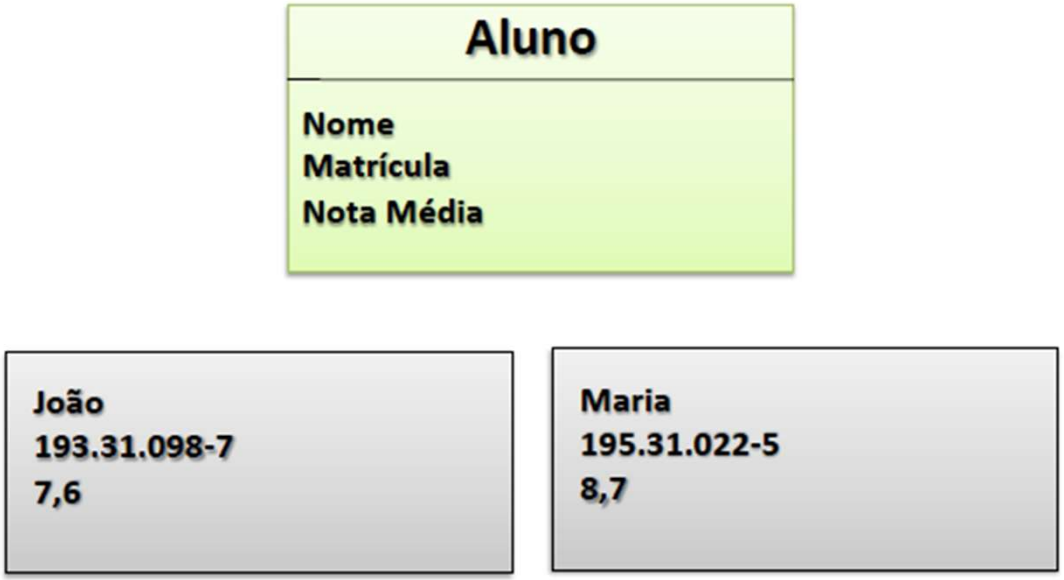
- O código a seguir mostra uma possível implementação para esta classe em Java

```
public class ContaCorrente {
    private int numero, agencia;
    private double saldo;

   public void inicializarContaCorrente(int n, int ag) {
    numero = n;
    agencia = ag;
    saldo = 0;
   }
```

Um exemplo simples – conta corrente

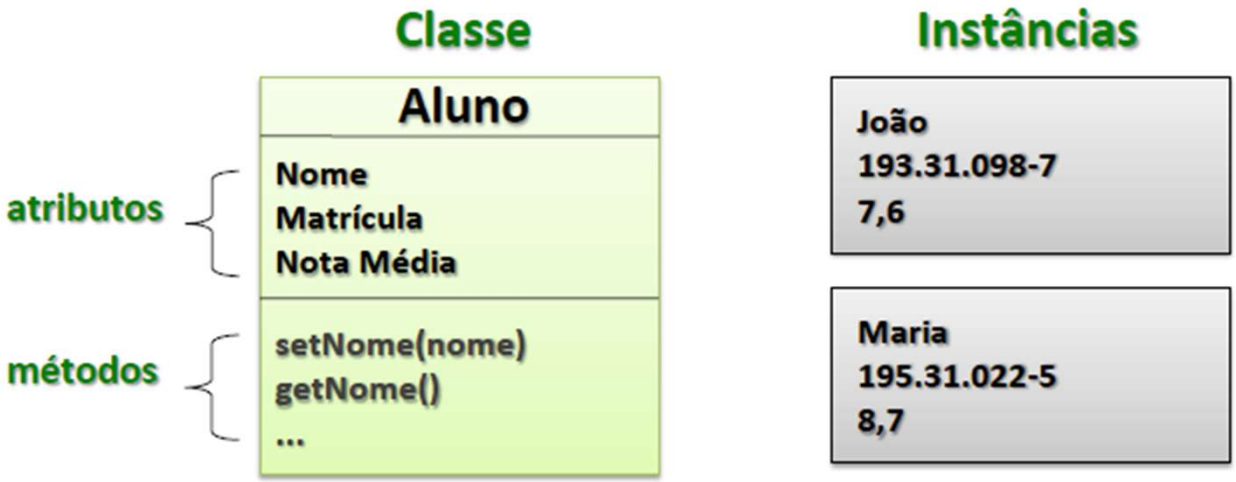
```
public void sacar(double valor){
     saldo = saldo - valor;
   }

   public void depositar(double valor){
    saldo = saldo + valor;
   }

   public double consultarSaldo(){
    return (saldo);
   }
```

Princípios Básicos - Programação Orientada a Objetos



Princípios Básicos - Programação Orientada a Objetos

As classes proveem a estrutura para a construção de objetos - estes são ditos **instâncias** das classes



Princípios Básicos - Programação Orientada a Objetos

Princípios Básicos - Programação Orientada a Objetos

Um objeto é uma instância de uma única classe.

Uma instância de um objeto é uma unidade de programação que é armazenada em uma variável.

Um programa orientado a objetos é composto por um conjunto de objetos que interagem entre si.

Princípios Básicos - Programação Orientada a Objetos

BORA PRATICAR?

Extraia as possíveis classes, cada qual com suas características e métodos:

Em um projeto de software, algumas informações devem ser armazenadas sobre alguns elementos de deveras importância. Dentro do contexto do cliente, existem pessoas, cada qual com seus nomes, celulares, endereços e cargos. Também se tem equipamentos, com descrição, ano, modelo, peso, valor. Para cada pessoa, de acordo com os dias trabalhados, deve-se: calcular comissão, calcular atrasos e descontos. Já as máquinas, precisam de calcular o número de manutenções de acordo com a data de fabricação.

Princípios Básicos - Programação Orientada a Objetos

BORA PRATICAR?

Extraia as possíveis classes, cada qual com suas características e métodos:

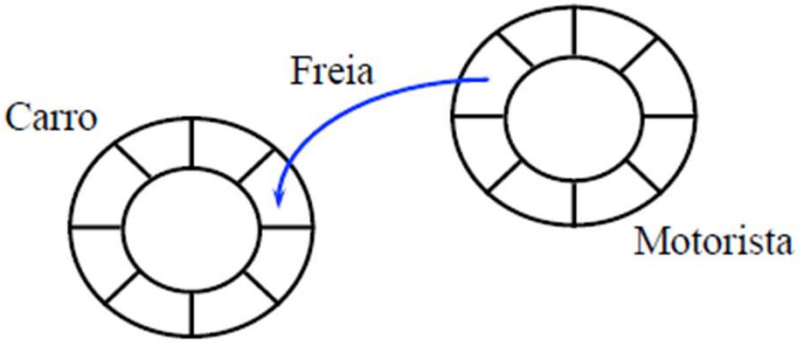
Em um projeto de software, algumas informações devem ser armazenadas sobre alguns elementos de deveras importância. Em uma clínica, um processo de cadastro de dados de pacientes, com nome, endereço, cpf, peso, identidade. Também á o processo de cadastro de médicos, com seus nomes, CRMs, telefones para contato. Há também o registro de consultas, onde se armazena a data de marcação, data de confirmação, plano de saúde utilizado, valor da consulta. Além disso, pode ser necessário um registro de um procedimento médico de intervenção, com sua data, horário, descrição, valor, e itens utilizados. Nos procedimentos, pode ser necessário que haja um cálculo do valor a ser pago pela sua execução.

Princípios Básicos - Programação Orientada a Objetos

Um programa OO é um conjunto de objetos que colaboram entre si para a solução de um problema

Objetos colaboram através de trocas de mensagens

A troca de mensagem representa a chamada de um método



Princípios Básicos - Programação Orientada a Objetos

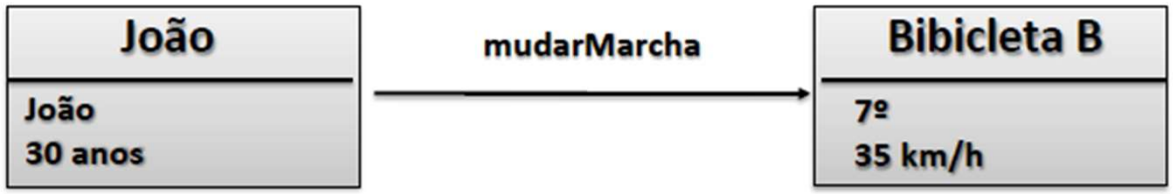
Um envio de mensagem sempre possui:

- Um emissor
- Um receptor
- Um seletor de mensagens (nome do método chamado)
- Parâmetros (opcionais)

Uma mensagem pode retornar um valor

Princípios Básicos - Programação Orientada a Objetos

Métodos operam no estado interno de um objeto e servem como mecanismo de comunicação entre objetos.



Princípios Básicos - Programação Orientada a Objetos

```
Qualificador_de_acesso class Nome_Da_Classe
{
   // atributos da classe
   // métodos da classe
}
```

```
// Class Lampada
public class Lampada
{
     // Atributos
     boolean acesa;

     // Métodos
     public void ligar()
     {   acesa = true;   }
     public void desligar()
     {   acesa = false;  }
}
```

```
// Class Bicicleta
class Bicicleta
{
     // Atributos
     int velocidade = 0;
     int marcha = 1;

     // Métodos
     void mudarMarcha(int novoValor)
     {   marcha= novoValor;   }
     Void aumentaVelocidade(int incremento)
     {   velocidade+= incremento;  }
}
```

Princípios Básicos - Programação Orientada a Objetos

Para instanciarmos um novo objeto devemos utilizar o operador new, como nos exemplos abaixo:

```
NomeDaClasse  nomeDoObjeto = new NomeDaClasse();
```

- ✓ Criando dois objetos bicicleta:

```
Bicicleta bicicleta1 = new Bicicleta();
Bicicleta bicicleta2 = new Bicicleta();
```

- ✓ Invocando seus métodos:

```
bicicleta1.mudarMarcha(2);
bicicleta2.aumentaVelocidade(5);
```

Princípios Básicos - Programação Orientada a Objetos

A classe provê a estrutura para a construção de objetos.

Um objeto é uma instância de uma classe.

Contém um estado (valores de seus atributos).

Expõe o seu comportamento através de métodos (funções).

Princípios Básicos - Programação Orientada a Objetos

É um princípio fundamental da OO:

Esconder o estado interno (valores dos atributos).

Obrigar que interações com os atributos sejam executadas através de métodos.

Com o encapsulamento um objeto determina a permissão que outros objetos terão para acessar seus atributos (estado).

Princípios Básicos - Programação Orientada a Objetos

Definição

É a utilização de técnicas de programação e mecanismos de linguagem de programação para agrupar e restringir acesso à atributos métodos e classes

Objetivo:

Reduzir a complexidade externa (interface) das classes
Preservar a integridade dos dados internos dos objetos

Princípios Básicos - Programação Orientada a Objetos

Encapsulamento de atributos: métodos de acesso

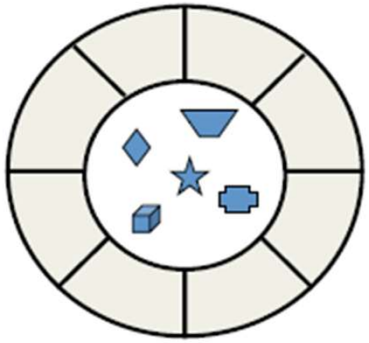
Encapsulamento de métodos: classes

Encapsulamento de classes: pacotes

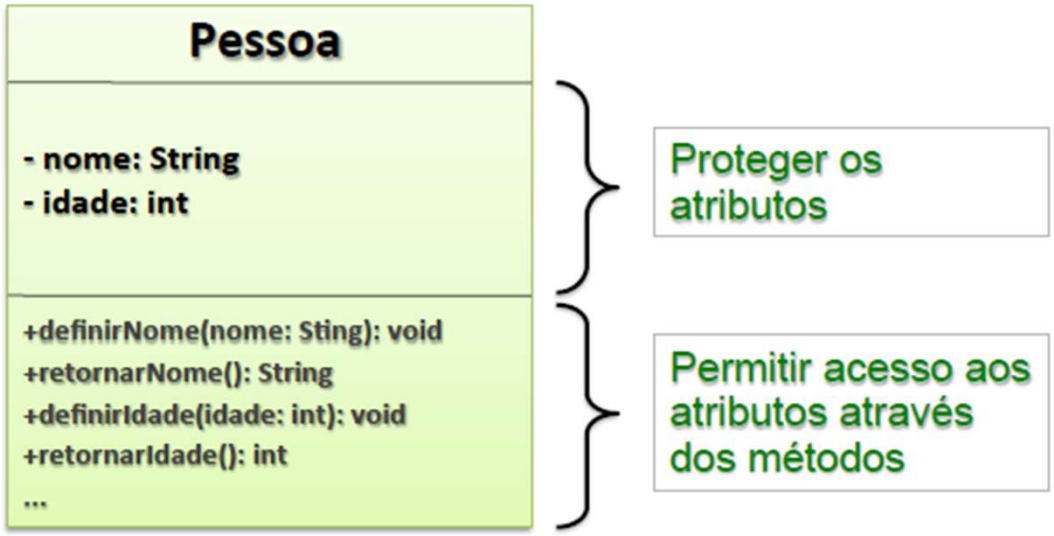
Princípios Básicos - Programação Orientada a Objetos

Atributos e Métodos

- Os métodos formam uma "cerca" em torno dos atributos
- Os atributos não podem ser manipulados diretamente
- Os atributos somente podem ser alterados ou consultados através dos métodos do objeto



Princípios Básicos - Programação Orientada a Objetos



Princípios Básicos - Programação Orientada a Objetos

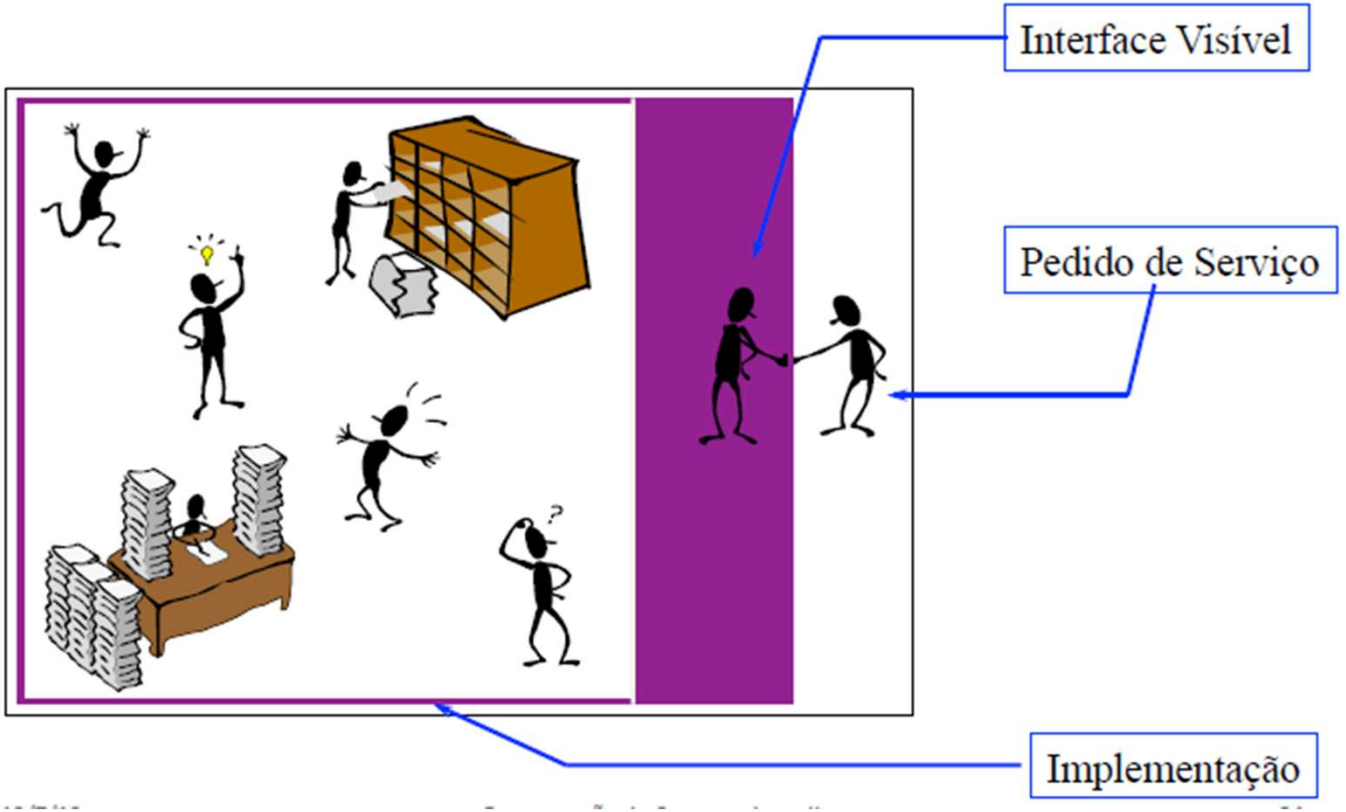
Um objeto X é denominado cliente de um objeto Y se utiliza métodos de Y

Pelo encapsulamento:

Clientes de um objeto podem utilizar seus métodos sem conhecer os detalhes de sua implementação

A implementação de um objeto pode ser alterada sem o conhecimento de seus clientes, desde que a interface visível seja mantida

Princípios Básicos - Programação Orientada a Objetos



Próxima Aula

Programação Orientada a Objetos: Classe

- Classe é um agrupamento de objetos
- A classe consiste nos métodos e nos dados que um determinado objeto irá possuir.
- Objetos são criados quando uma mensagem solicitando a criação é recebida pela sua classe.
- A programação orientada a objetos consiste em implementar as classes e na utilização das mesmas, através da sua intercomunicação.
- Um objeto é uma instância da classe

Programação Orientada a Objetos: Classe

- Níveis de Acesso:

Os níveis de acesso a atributos, métodos e classes são definidos da seguinte forma:

- **public:** Método ou atributo visível a todas as classes (público)
- **protected:** Método ou atributo visível nas subclasses (protegido)
- **private:** Método ou atributo visível somente na classe onde é utilizado (privado)

Programação Orientada a Objetos: Classe

SOBRECARREGAMENTO (Overloading)

- Consiste em possuir mais de um método com o mesmo nome, porém com diferentes parâmetros (número, ordem e tipos)
- Este recurso é utilizado para métodos que realizam tarefas semelhantes porém sobre tipos de dados diferentes
- Normalmente este recurso também é utilizado no construtor da classe
- O sobrecarregamento facilita a reutilização de código:

  Exemplo: Circle c = new Circle();

  c.move(x,y);//Recebe a coordenada x e y c.move(p); //Recebe o objeto ponto

Programação Orientada a Objetos: Classe

Override (Sobrescrita)

Redefinição de métodos, sobrescrita ou overriding é um mecanismo da programação orientada a objetos.

Ele permite que uma subclasse forneça um método que já é fornecido por uma de suas superclasses.

A redefinição ocorre quando um método cuja assinatura já tenha sido especificada recebe uma nova definição (ou seja, um novo corpo) em uma classe derivada.

Programação Orientada a Objetos: Classe

Override (Sobrescrita)

Como indicar que o método poderá ser reescrito? Utilizando o atributo virtual

na hora de construir o método da classe pai, exemplo:

```
public virtual void comer(string alimento)
        {
            Console.WriteLine("A Pessoa comendo {0}",alimento);
        }
```

Programação Orientada a Objetos: Classe

Override (Sobrescrita) : Para fazer a classe ser sobrescrita, é necessário utilizar a herança. Veja o exemplo 03override:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

public class Teste
{
    public class Pessoa
    {
        public string nome { set; get; }
        public int idade { set; get; }

        public virtual void comer(string alimento)
        {
            Console.WriteLine("A Pessoa comendo {0}",alimento);
        }
    }

    public class Sergio : Pessoa
    {
        public override void comer(string alimento)
        {
            if (alimento == "peixe")
                Console.WriteLine("Sérgio não gosta de peixe. Não vai comer.");
            else
                base.comer(alimento); //base chama o método pai
        }
    }

    public static void Main()
    {
        string comida;
        Pessoa teste = new Pessoa();
        Console.Write("Qual comida você dará a Pessoa e Sergio? ");
        comida = Console.ReadLine();
        teste.comer(comida);
        Sergio teste3 = new Sergio();
        teste3.comer(comida);
        Console.ReadLine();
    }
}
```

Programação Orientada a Objetos: Classe

MÉTODOS:

- CONSTRUTORES
- DESTRUTORES
- MODIFICADORES
- ACESSORES
- OUTROS MÉTODOS

Programação Orientada a Objetos: Classe

- **MÉTODOS CONSTRUTORES**
- Método especial. Deve possuir o mesmo nome da classe
- Inicializa os dados (variáveis membro) do objeto
- Garante que objetos iniciem em um estado consistente
- Construtores normalmente são sobrecarregados
- Construtores não podem retornar um valor
- O construtor que não recebe nenhum parâmetro é conhecido como “construtor default”

Programação Orientada a Objetos: Classe

- **MÉTODOS DESTRUTORES**
- Devolve os recursos para o sistema
- Não tem parâmetros, não retorna valor. Sua assinatura é: protected void finalize()
- O método finalize somente é chamado imediatamente antes da coleta de lixo (garbage colletion). Ele não é chamado quando um objeto sai de escopo, por exemplo.
- Normalmente não é necessário a criação deste método pois o mesmo é herdado da classe **Object**

Programação Orientada a Objetos: Classe

- **MÉTODOS MODIFICADORES (setXXX)**
- Permitem a modificação dos dados (variáveis membro) do objeto
- Devem ser do tipo **public**, pois são acessados de forma externa ao objeto
- Normalmente começam com o “set” a fim de facilitar o entendimento de seu objetivo.

Programação Orientada a Objetos: Classe

- **MÉTODOS ACESSORES (getXXX)**
- Permitem a recuperação dos dados (variáveis membro) do objeto
- Devem ser do tipo public, pois são acessados de forma externa ao objeto
- Normalmente começam com o “get” a fim de facilitar o entendimento de seu objetivo.

Programação Orientada a Objetos: Classe

- **OUTROS MÉTODOS**
- Executam tarefas que são de responsabilidade do objeto e que irão representar o comportamento do mesmo.
- Estes métodos podem ser públicos **(public)**, protegidos **(protected)** ou privados **(private)**, conforme sua utilização ;
- Para que o método seja acessado externamente o mesmo de ser do tipo público **(public)**
- Caso o método seja apenas auxiliar à classe sem uma ligação direta com o comportamento do objeto o mesmo deve ser do tipo privado **(private)**. Normalmente este métodos são utilizados por um outro método público.

Programação Orientada a Objetos: Classe

GENERALIZAÇÃO X ESPECIALIZAÇÃO

GENERALIZAÇÃO

- A generalização consiste em obter similaridades entre as várias classes e partir destas similaridades, novas classes são definidas.
- Estas classes são chamadas superclasses

ESPECIALIZAÇÃO

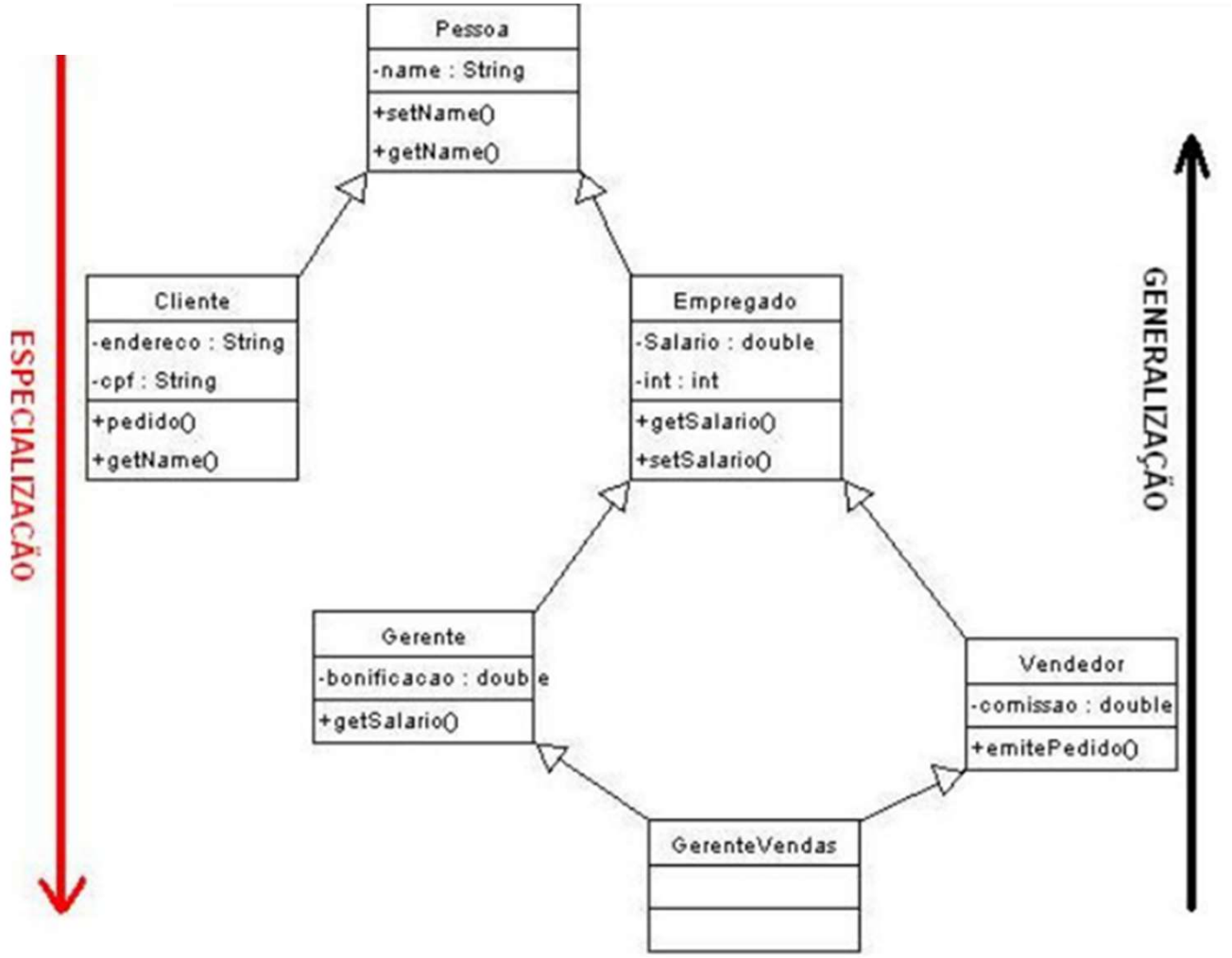
- A especialização por sua vez consiste em observar diferenças entre os objetos de uma mesma classe e dessa forma novas classes são criadas.
- Estas classes são chamadas subclasses.

Programação Orientada a Objetos: Classe

GENERALIZAÇÃO

X

ESPECIALIZAÇÃO



Programação Orientada a Objetos: Classe Abstrata

- Ao subir na hierarquia de heranças, as classes se tornam mais genéricas e, provavelmente mais abstratas;
- Em algum ponto, a classe ancestral se torna tão geral que acaba sendo vista mais como um modelo para outras classes do que uma classe com instâncias específicas que são usadas;
- Uma classe abstrata não pode ser instanciada, ou seja, não há objetos que possam ser construídos diretamente de sua definição.
- Classes abstratas correspondem a especificações genéricas, que deverão ser concretizadas em classes derivadas (subclasses);
- Uma classe abstrata serve apenas para definir um comportamento comum que todas as classes derivadas devem seguir.

Programação Orientada a Objetos: Classe Abstrata

- Classes abstratas são classes que não podem ser instanciadas, mas é possível declarar uma variável (referência) deste tipo.
- São utilizadas apenas para permitir a derivação de novas classes.
- Sintaxe:

```
abstract class NomeDaSuperclasse
{ // corpo da classe abstrata... }
```

- Portanto:

  NomeDaSuperclasse f = new NomeDaSuperclasse( ); –> Erro

Programação Orientada a Objetos: Classe Abstrata

- Uma **classe abstrata** é uma classe que não tem instâncias diretas.
- Uma **classe concreta** é uma classe que pode ser instanciada.
- As **classes abstratas** podem possuir métodos abstratos.

Programação Orientada a Objetos: Classe Abstrata

Classes e Métodos Abstratos

- Um método abstrato promete que todos os descendentes não abstratos dessa classe abstrata irão implementar esse método abstrato;

- Os métodos abstratos funcionam como uma espécie de guardador de lugar para métodos que serão posteriormente implementados nas subclasses;

- Uma classe pode ser declarada como abstrata mesmo sem ter métodos abstratos;

Programação Orientada a Objetos: Classe Abstrata

Regras sobre Classes Abstratas

- Toda classe derivada de uma classe abstrata deve obrigatoriamente implementar os métodos abstratos da superclasse, caso contrário um erro de compilação é gerado.

  • Uma classe que tenha um ou mais métodos abstratos deve ser obrigatoriamente definida como abstrata, caso contrário um erro de compilação é gerado.

  • Uma classe abstrata pode conter métodos não abstratos, isto é, com implementação.

  • Se esses métodos não abstratos não forem definidos (sobrepostos) nas subclasses, então, quando um objeto da subclasse realizar a chamada a um desses métodos, o código contido na classe abstrata será executado (devido à herança).

Programação Orientada a Objetos: Classe Abstrata

```
public abstract class Figura {
    public abstract double calcularArea( );

    public void imprimeArea( ){
        System.out.println(calcularArea( ));
    }
}

public class Main {
     public static void main(String[] args) {
          Quadrado q = new Quadrado(4);
          Circulo c = new Circulo(2);
          System.out.print("Área da Figura 1 é: ");
          q.imprimeArea( );
          System.out.print("Área da Figura 2 é: ");
          c.imprimeArea( );
     }
}
```

Programação Orientada a Objetos: Classe Abstrata

```
public class Circulo extends Figura {

        double raio;

        public Circulo (double raio) {
                this.raio = raio;
        }

        public double calcularArea( ) {
                double area = 0;
                area = 3.14 * raio * raio;
                return area;
        }
}
```

Programação Orientada a Objetos: Classe Abstrata (em Python)

```
class Foo:
    def __getitem__(self, index):
        ...
    def __len__(self):
        ...
    def get_iterator(self):
        return iter(self)

class MyIterable(ABC):

    @abstractmethod
    def __iter__(self):
        while False:
            yield None

    def get_iterator(self):
        return self.__iter__()

    @classmethod
    def __subclasshook__(cls, C):
        if cls is MyIterable:
            if any("__iter__" in B.__dict__ for B in C.__mro__):
                return True
        return NotImplemented

MyIterable.register(Foo)
```

Programação Orientada a Objetos: Interface

```
public class Circulo extends Figura {

	double raio;

	public Circulo (double raio) {
		this.raio = raio;
	}

	public double calcularArea( ) {
		double area = 0;
		area = 3.14 * raio * raio;
		return area;
	}
}
```

Programação Orientada a Objetos: Interface

Interface de um Objeto/Classe

● Interface **em Java** é uma palavra-chave usada para definir uma coleção de definições de métodos e de valores de constantes.

- São semelhantes as classes abstratas, mas todos os métodos comportam-se como abstratos.
- Os métodos são qualificados como public por default.
- Não definem atributos comuns
- Só definem constantes, (“atributos” qualificados como public, static e final).
- Não definem construtores
- Não podem ser instanciadas.

Programação Orientada a Objetos: Interface

- Uma interface não pode ser instanciada (Não se pode criar objetos)
- Definem tipo de forma abstrata, apenas indicando a assinatura dos métodos
- Os métodos são implementados por classes e, para isso, é utilizada a palavra-chave implements.
- Mecanismo de projeto
    - podemos projetar sistemas utilizando interfaces
    - projetar serviços sem se preocupar com a sua implementação (abstração)

Programação Orientada a Objetos: Interface

implements

- Classe que implementa uma interface deve definir os métodos da interface:

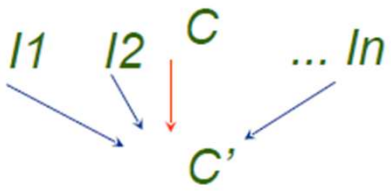
- **classes concretas** têm que implementar os métodos

- **classes abstratas** podem simplesmente conter métodos abstratos correspondentes aos métodos da interface

Programação Orientada a Objetos: Interface

- **Definição de Classes: Forma Geral**

```
class C'
   extends C
   implements I1, I2, ..., In {
     /* ... */
   }
```



Programação Orientada a Objetos: Interface

- **Exemplo de uso de Interface**

```
public interface Figura {
    public double calcularArea( );
}

public class Quadrado implements Figura {
     double lado;
     public Quadrado(double lado) {
       this.lado = lado;
     }

     public double calcularArea( ) {
        double area = 0;
        area = lado * lado;
        return area;
     }
}
```

Como a classe Quadrado implementa a interface Figura, então, o método calcularArea( ) deve obrigatoriamente ser implementado.

Programação Orientada a Objetos: Interface

- **Exemplo de Uso de Interface**

```
public class Circulo implements Figura {

    double raio;

    public Circulo (double raio) {
        this.raio = raio;
    }

    public double calcularArea( ) {
        double area = 0;
        area = 3.14 * raio * raio;
        return area;
    }
}
```

Programação Orientada a Objetos: Interface

- **Exemplo de Uso de Interface**

```
public class Main {

    public static void main(String[] args) {

        Figura f1 = new Quadrado(4);

        Figura f2 = new Circulo(2);

       System.out.println("Área da Figura 1 é: "
                    + f1.calcularArea( ) + "\n"
                   + "Área da Figura 2 é: "
                    + f2.calcularArea( ));
       }
}
```

Observe que uma interface não pode ser instanciada mas é possível um objeto, declarado como sendo do tipo definido por uma interface, receber objetos de classes que implementam tal interface.

Programação Orientada a Objetos: Interface

- **Classes (abstratas)**
- Agrupa objetos com implementações compartilhadas
- Define novas classes através de herança simples (herda de uma única classe abstrata ou não)
- Só uma pode ser supertipo de outra classe
- Podem conter métodos não-abstratos (com implementação)

- **Interfaces**
- Agrupa objetos com implementações diferentes;
- Define novas interfaces através de herança múltipla (implementa várias interfaces)
- Várias podem ser supertipo do mesmo tipo.

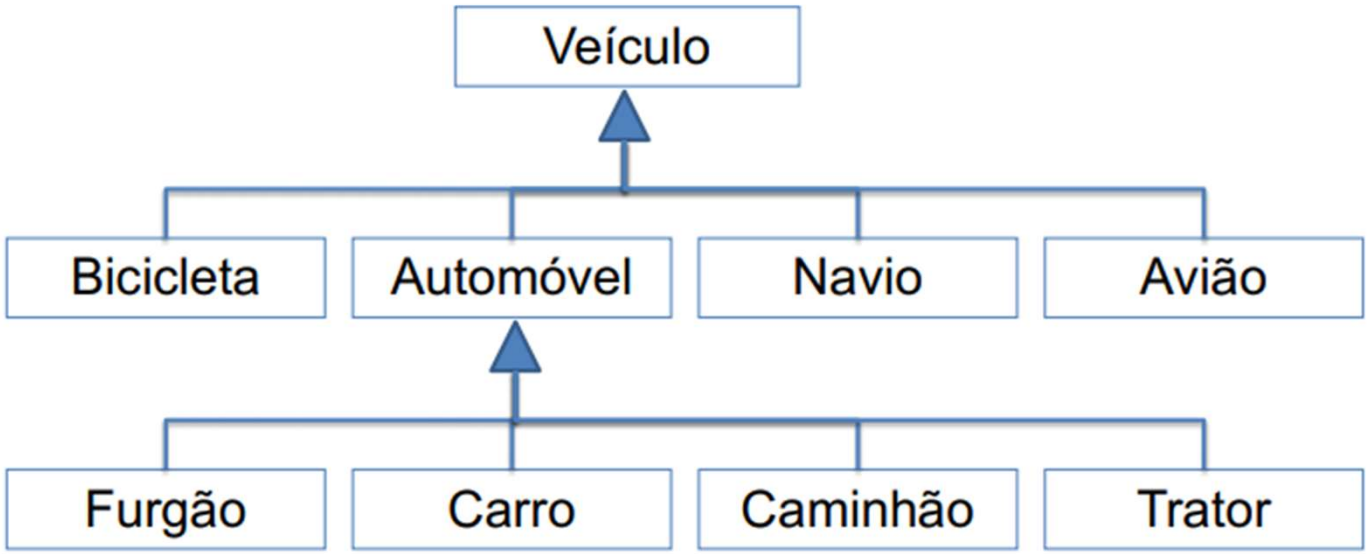
Programação Orientada a Objetos: Interface (em Python)

```python
import abc

class FormalParserInterface(metaclass=abc.ABCMeta):
    @classmethod
    def __subclasshook__(cls, subclass):
        return (hasattr(subclass, 'load_data_source') and
                callable(subclass.load_data_source) and
                hasattr(subclass, 'extract_text') and
                callable(subclass.extract_text))

class PdfParserNew:
    """Extract text from a PDF."""
    def load_data_source(self, path: str, file_name: str) -> str:
        """Overrides FormalParserInterface.load_data_source()"""
        pass

    def extract_text(self, full_file_path: str) -> dict:
        """Overrides FormalParserInterface.extract_text()"""
        pass

class EmlParserNew:
    """Extract text from an email."""
    def load_data_source(self, path: str, file_name: str) -> str:
        """Overrides FormalParserInterface.load_data_source()"""
        pass

    def extract_text_from_email(self, full_file_path: str) -> dict:
        """A method defined only in EmlParser.
        Does not override FormalParserInterface.extract_text()
        """
        pass
```

Programação Orientada a Objetos: Herança

- **Herança** é a capacidade de uma subclasse de ter acesso as propriedades da superclasse a ela relacionada.
- Dessa forma as propriedades de uma classe são propagadas de cima para baixo em um diagrama de classes.
- Neste caso dizemos que a subclasse herda as propriedades e métodos da superclasse
- A relação de herança entre duas classes é uma relação da seguinte forma: A “e um tipo de” B, onde A e B são classes. Caso esta relação entre as classes não puder ser construída, em geral, também não se tem uma relação de herança entre a classe A a partir da classe B.

Programação Orientada a Objetos: Herança



Programação Orientada a Objetos: Herança

- **Herança X Uso**

Além da relação de herança entre as classes existe a relação de uso

HERANÇA

classe A “é um tipo de” B

USO / AGREGAÇÃO (Relação de Conteúdo)

- classe D “contém” classe C”
- classe D “usa” classe C”
- classe C “é parte da” classe D

Exemplo: Uma equipe contém um gerente e um grupo de vendedores

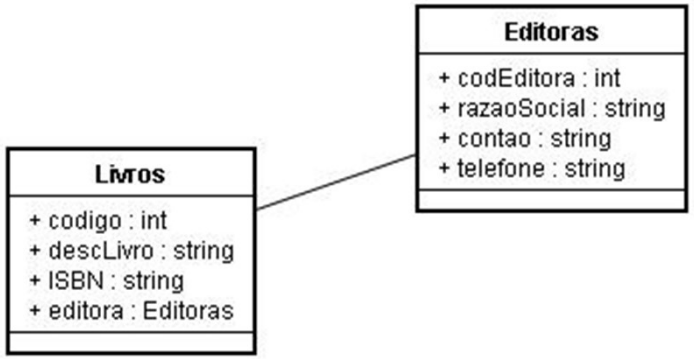
Programação Orientada a Objetos: Herança

Associação

Descreve um vínculo que ocorre entre classes - associação binária -, mas é possível até mesmo que uma classe esteja vinculada a si própria, - associação unária-, ou que uma associação seja compartilhada por mais de uma classe, o que conhecemos por associação ternária ou N-ária, tipo de associação mais rara e também mais complexa.

Representamos as associações por meio de retas que ligam as classes envolvidas, essas setas podem ou não possuir setas nas extremidades indicando a navegabilidade da associação, ou seja, o sentido em que as informações são passadas entre as classes - não obrigatório-. Ou seja, se não há setas, significa que essas informações podem ser transmitidas entre todas as classes de uma associação.

Exemplo: A forma mais comum de implementar associação é ter um objeto como atributo de outro, neste exemplo, abaixo temos uma associação entre a Classe Livros e a classe Editoras. No código cria-se um objeto do tipo Livro e outro do tipo Editora. Um dos atributos do Livro é a Editora.



Programação Orientada a Objetos: Herança

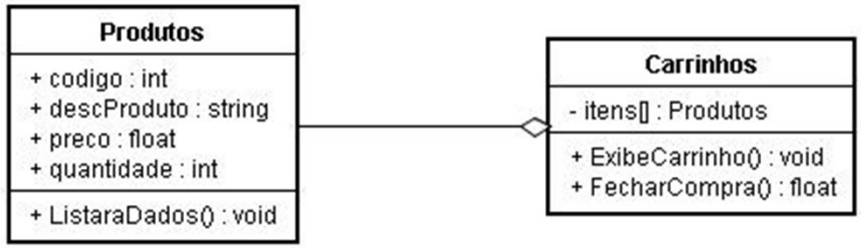
Agregação

É um tipo especial de associação onde tenta-se demonstrar que as informações de um objeto (chamado objeto-todo) precisam ser complementados pelas informações contidas em um ou mais objetos de outra classe (chamados objetos-parte); conhecemos como todo/parte.

O objeto-pai poderá usar as informações do objeto agregado.

Um exemplo esta relação pensando em um ambiente Web, onde teríamos o carrinho de compras (classe Carrinhos) com vários itens do tipo produtos (classe Produtos).

Para agregar os produtos ao carrinho, usa-se o método IncluirItem( ) na classe Carrinhos, que contém outro método chama ExibeCarrinho() responsável por listar todos os itens pedidos, por meio da listagem dos dados do produto -método ListarDados() da classe Produtos-, e um método FechaCompra( ) reponsável por efetuar a soma dos itens adicionados no carrinho apresentando ao final o preço a ser pago pelo cliente. Na próxima figura vemos o exemplo desta agregação.

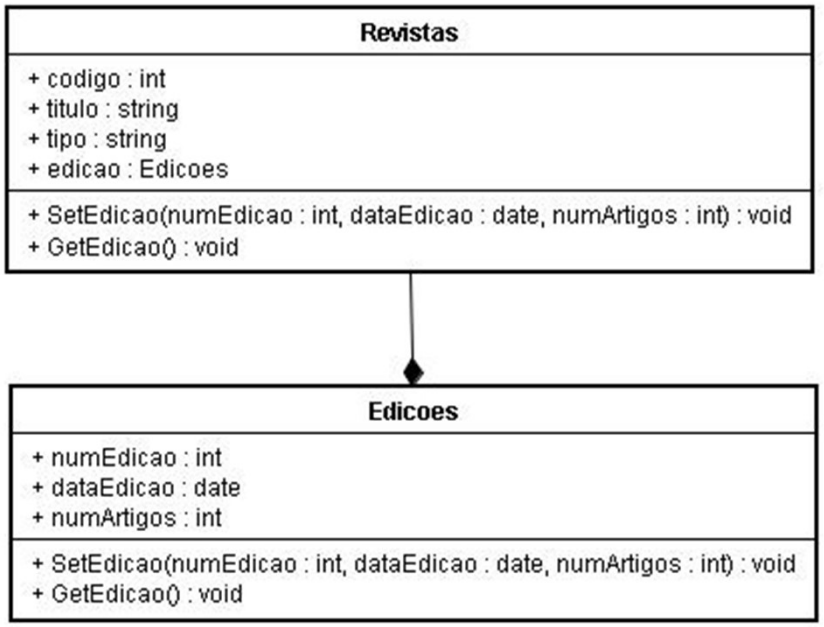


Programação Orientada a Objetos: Herança

Composição

Pode-se dizer que composição é uma variação da agregação. Uma composição tenta representar também uma relação todo - parte. No entanto, na composição o objeto-pai (todo) é responsável por criar e destruir suas partes. Em uma composição um mesmo objeto-parte não pode se associar a mais de um objeto-pai.

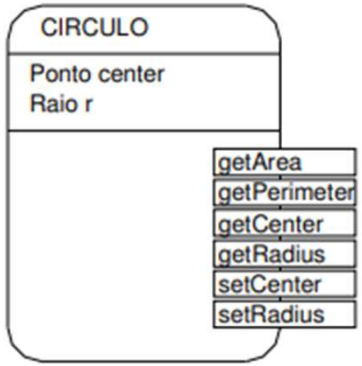
Veja o Diagrama de Classes que representa uma associação do tipo composição. Nela temos o objeto-todo Revistas e objeto-parte Edicoes.



Programação Orientada a Objetos: Encapsulamento

- **Encapsulamento** é um termo que indica que os dados contidos em um objeto somente poderão ser acessados através de seus métodos.

- Dessa forma não é possível alterar os dados diretamente, somente através de métodos. Ex: O raio somente pode ser alterado/recuperado pelos métodos setCenter/getCenter.



Programação Orientada a Objetos: Polimorfismo

Os objetos respondem às mensagens que eles recebem através dos métodos. A mesma mensagem pode resultar em diferentes resultados.

Outras Definições:

- É a capacidade de objetos instanciados de diferentes classes (com uma superclasse comum) responderem à chamada de métodos, de forma diferente (particular)
- Capacidade de um objeto decidir que método aplicar a si mesmo.
- Capacidade de assumir formas diferentes. Permite programar de forma genérica para manipular de uma grande variedade de classes.

Programação Orientada a Objetos: Polimorfismo

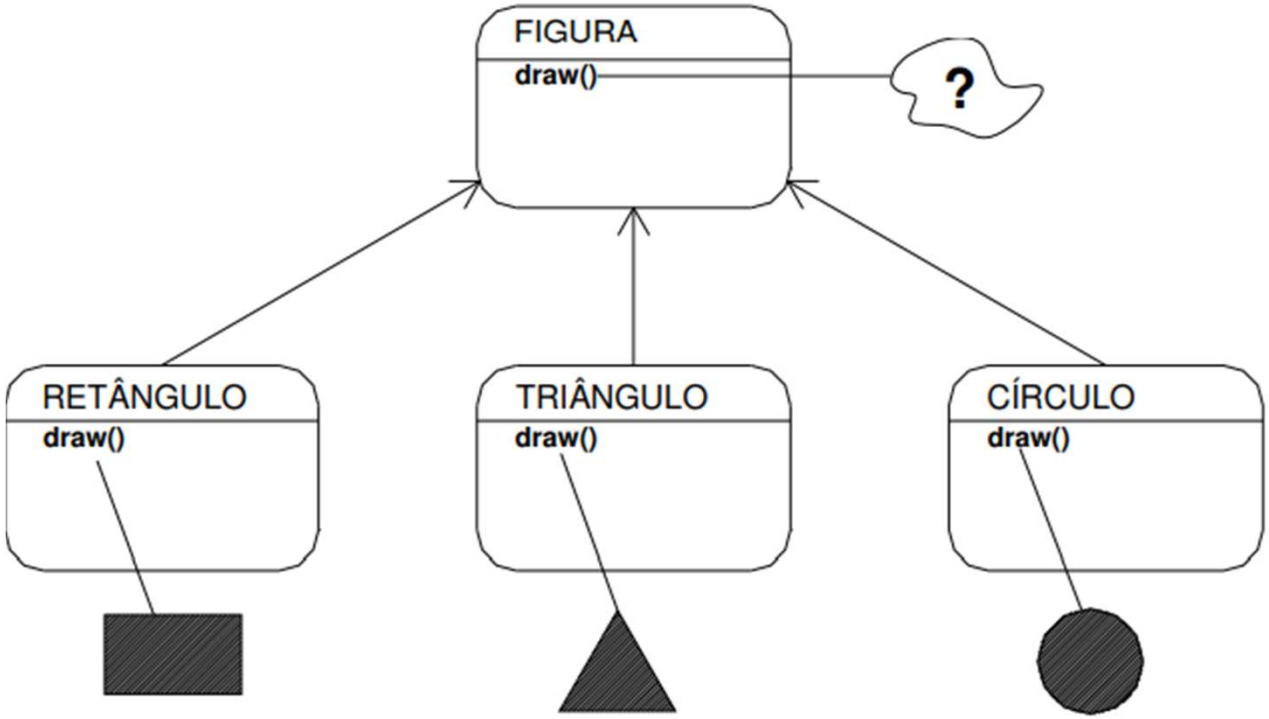
Esta propriedade é chamada de **polimorfismo**

- Exemplo: Método getSalario()
- Para um empregado qualquer: getsalario() = Salario;
- Para o gerente: getsalario() = salario + bonificacao;

Exemplo: Método **draw()**

- Para uma figura qualquer desenha uma forma não definida
- Para o retângulo, triângulo e círculo o mesmo método responde de uma forma diferente

Programação Orientada a Objetos: Polimorfismo



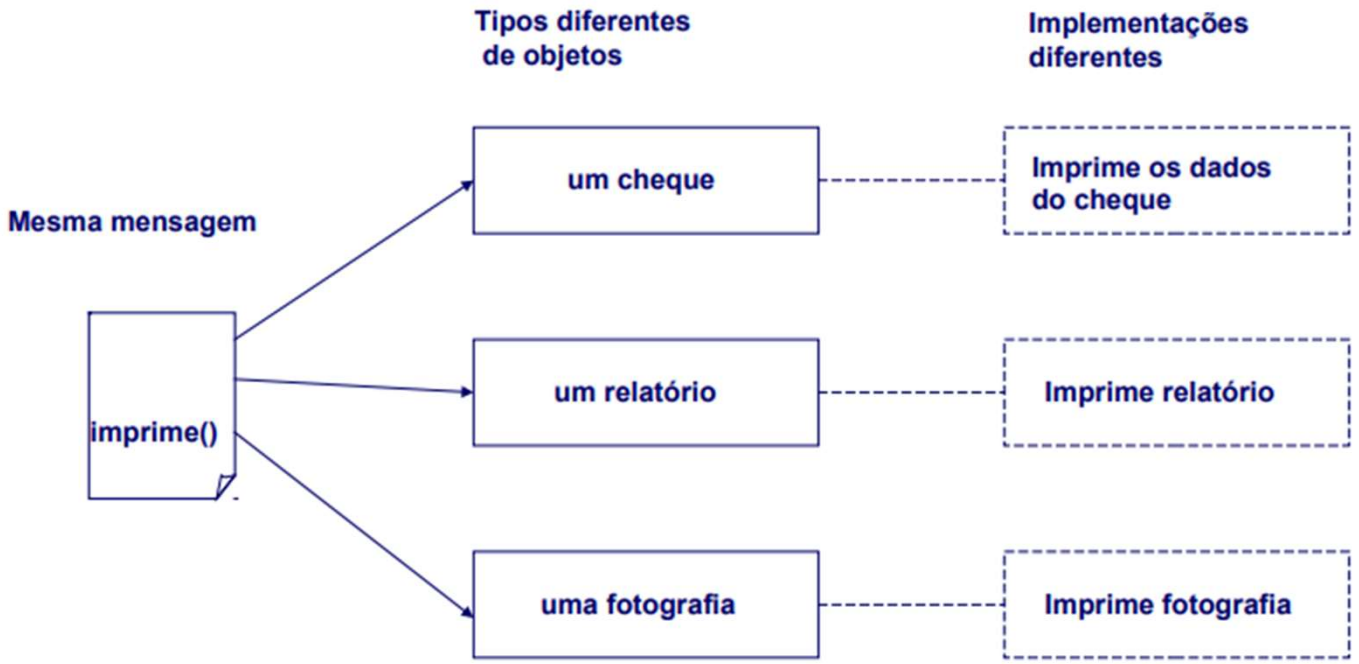
Programação Orientada a Objetos: Polimorfismo

```
class Super:
 def hello(self):
  print("Olá, sou a superclasse!")

class Sub (Super):
 def hello(self):
  print("Olá, sou a subclasse!")

teste = Sub()
teste.hello()
```

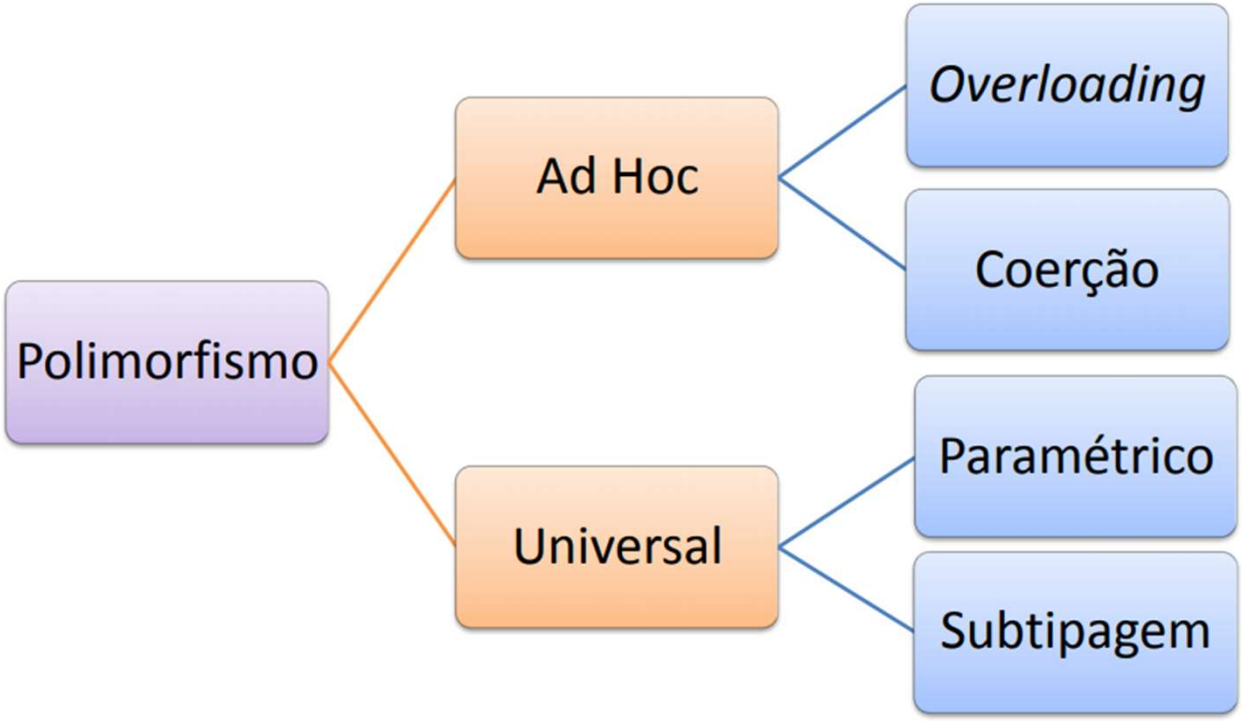
O resultado vai ser:

- Olá, sou a subclasse!

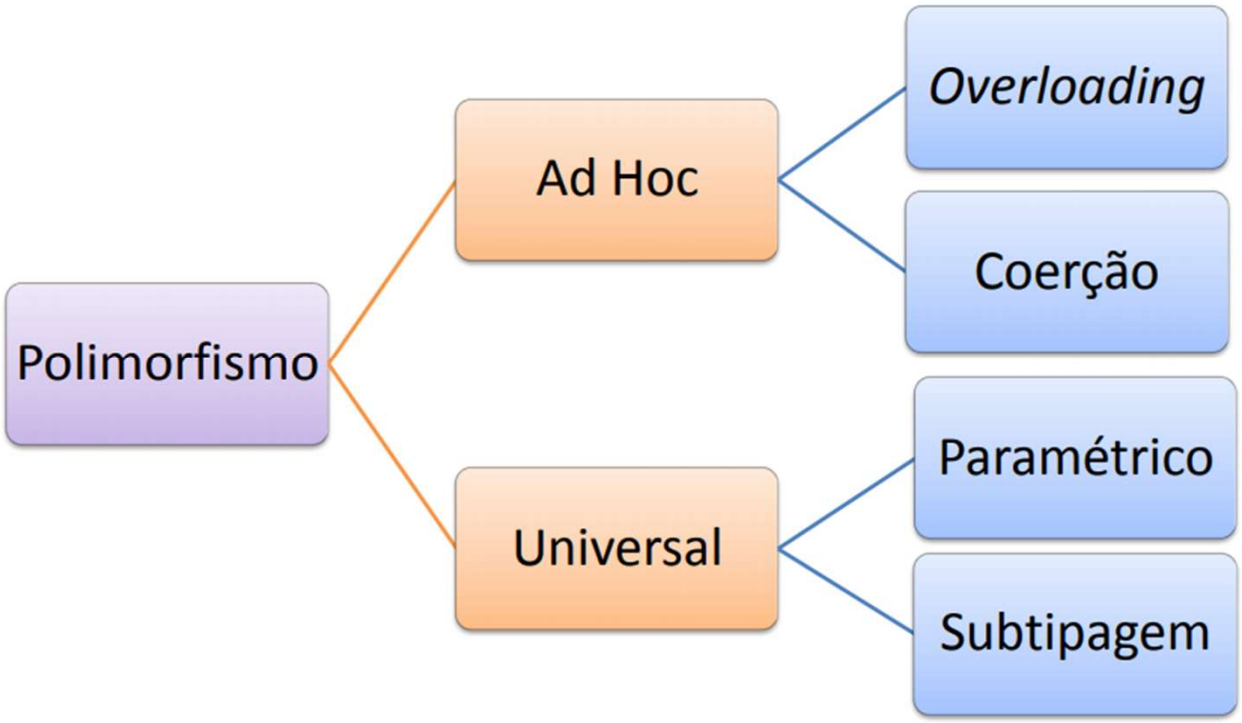
Programação Orientada a Objetos: Polimorfismo



Programação Orientada a Objetos: Polimorfismo



Programação Orientada a Objetos: Polimorfismo



Sobrecarga (overloading) – Utiliza os tipos para escolher a definição

Coerção – Utiliza a definição para escolher o tipo de conversão