**Tópico 1 – Conceitos importantes**

**Classe**

**O que uma classe faz?**

#### Ela descreve que propriedades (ou **atributos**) um objeto terá e qual o comportamento dos Objetos da classe (são os chamados **métodos**).

#### **A especificação de uma classe** é composta por:

#### 

#### Imagem relacionada**Classe:** descreve que propriedades (ou **atributos**) um objeto terá e qual o comportamento dos Objetos da classe (são os chamados **métodos**). **Especificação de uma classe:** é composta de nome da classe, conjunto de atributos e conjunto de métodos. **Nome da classe:** é um identificador para a classe, que possibilita referenciá-la posteriormente. **EX:** pode referenciá-la no momento da criação de um objeto

**Nome da classe**

**Conjunto de métodos**

**Conjunto de atributos**

**Classe**

**Nome da classe:** é um identificador para a classe, que possibilita referenciá-la posteriormente.

EXEMPLO Pode referencia-la no momento da criação de um objeto

**Conjunto de atributos:** Descreve as propriedade da classe. Cada atributo possui um nome e um tipo associado.

Pode ter também um valor\_default opcional.

**Tipo:** Pode ser o nome de uma classe (nas linguagens de programação orientadas a objetos pura). Ou do grupo de tipos primitivos das linguagens de POO como inteiro, real, caracter.

**Valor\_default opcional:** especifica um valor inicial para o atributo.

#### 



**Classe**

**. . . . . . . . . . . . . . . . . . . . . . . . . . .**

**Conjunto de métodos**

Definem as funcionalidades da classe, isto é, define o que será possível fazer com os objetos dessa classe. É um equivalente a um procedimento ou uma função, porém manipula apenas suas variáveis locais e atributos definidos para a classe. Cada método é especificado por uma assinatura, composta por um identificador para o método (nome do método), o tipo para o valor de retorno e sua lista de argumentos, sendo cada argumento identificado por seu tipo e nome.

#### 

**Classe**

**Mecanismo de sobrecarga (overloading)**

Através desse mecanismo dois métodos de uma mesma classe podem ter o mesmo nome, contanto que tenham assinaturas diferentes, ou seja, com argumentos diferentes. Nesse caso não há conflito, pois o compilador é capaz de detectar qual método deve ser escolhido com base nos tipos dos argumentos de cada método. A escolha do método correto é chamada de ligação prematura (early binding).

#### 

**Classe**

**Visibilidade:** Pode ser para métodos e atributos.   
  
Três categorias de visibilidade podem ser definidas:

**PROTEGIDO**

**O atributo ou método de um objeto pertencente a essa classe pode ser acessado apenas por objetos de classes que tenham herança dessa classe, ou seja, pode ser acessado apenas por objetos de classes derivadas dessa classe. Simbolizado em UML por #.**

**PÚBLICO**

**O atributo ou método de um objeto pertencente a essa classe pode ser acessado por qualquer outro objeto (visibilidade externa total). Simbolizado em UML por +.**

**PRIVADO**

**O atributo ou método de um objeto pertencente a essa classe não pode ser acessado por nenhum outro objeto (nenhuma visibilidade externa). Simbolizado em UML por -.**

****

**Objetos**

**Visibilidade:** Pode ser para métodos e atributos.   
  
Três categorias de visibilidade podem ser definidas:

**Objetos**

#### **Objetos:** São instâncias de classes. Em um sistema implementado em linguagens orientadas a objetos, basicamente todo o processamento ocorre através de objetos.

#### F,{d01544f5-c31a-4184-932b-12cad39134f4}{74},10.41667,3.541667

#### Em linguagens de programação orientadas a objetos, tudo pode ser potencialmente representado como um objeto. Sob o ponto de vista desse paradigma, um objeto é similar a uma variável normal. EX: Uma variável do tipo int definida em uma linguagem de programação como C ou Java possui: espaço em memória para armazenar seu estado (valor) e um conjunto de operações que podem ser aplicadas por meio dos operadores para valores inteiros definidos pela linguagem. Da mesma maneira: um objeto criado possui um espaço em memória para armazenar seu estado (valores dos seus atributos) e um conjunto de operações aplicáveis a esse objeto (métodos definidos pela classe).

Um programa orientado a objetos possui um conjunto de objetos que interagem por meio de “trocas de mensagens”, ou seja, através da aplicação de métodos a objetos.  
  
Recomendação: Estrutura de um objeto e implementação dos seus métodos privados.  
  
Encapsulação: é o princípio de projeto pelo qual cada componente de um programa deve agregar toda a informação relevante para sua manipulação como uma unidade (uma cápsula). É um mecanismo poderoso quando aliado ao conceito de ocultamento de informação.   
  
Ocultamento de informação: princípio onde cada componente deve manter oculta sob sua guarda uma decisão de projeto única. Para a utilização desse componente, apenas o mínimo necessário para sua operação deve ser revelado (tornado público).   
  
Recomendação: o uso da encapsulação e ocultamento de informação, que representa o estado de um objeto, deve ser mantida oculta.

**Interface operacional:** conjunto de assinaturas dos métodos públicos.   
Apenas a assinatura dos métodos é revelada, desssa maneira, detalhes internos sobre a operação do objeto não são conhecidos. Assim, o usuário do objeto pode trabalhar em níveis mais altos de abstração, sem se preocupar com detalhes internos da classe. Tal fator possibilita a simplificação de programas com funcionalidades complexas.

**RESUMINDO**

Desejamos criar um tipo: Aluno

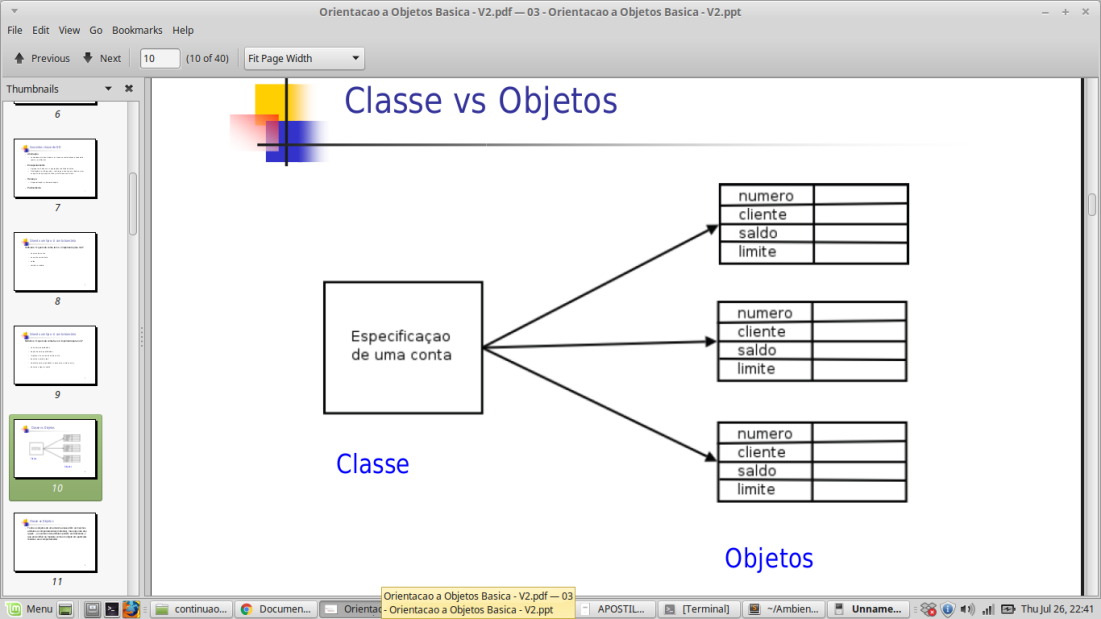
Atributos: O que o Aluno precisa ter?

* Nome
* CPF
* RG
* Endereço
* Telefone

Métodos: O que um Aluno pode fazer?

* Estudar
* Pegar livro na biblioteca
* Fazer prova
* Tirar dúvida da matéria

Classe Vs Objeto



**Tópico 2 – Conceitos chave de OO**

Para que uma linguagem seja considerada Orientada a Objetos, ela deve atender a 4 tópicos:

**Abstração:** Existem 3 pontos que devem ser considerados:

* Identidade – que deve ser única dentro do sistema, para evitar conflito.
* Propriedades – elementos que definem os objetos, ou seja, são as características que um objeto possui.
* Métodos - são as ações que um objeto pode executar.

**Encapsulamento:** um dos principais conceitos, pois ele possibilita esconder determinadas propriedades. Para melhor entender esse conceito, vamos a uma analogia:

Ao clicar no botão de ligar uma TV, não sabemos o que acontece internamente a TV. Portanto, podemos dizer que os métodos que ligam a TV estão encapsulados.

**Herança:** Mecanismo que possibilita que as características em comum a várias classes possam ser unidas e separadas em uma classe base, a superclasse. Assim, a partir da classe base, define-se as classes mais específicas. As subclasses (classes mais específicas) possuem características (atributos e métodos) da classe base acrescidos as suas características particulares.

A herança permite a otimização da produção de um programa, pois ela possibilita a diminuição das linhas de código. Podemos fazer outra analogia a vida real para melhor compreensão desse conceito:

Um filho herda características do pai, que por sua vez, herda características do avô. Em orientação a objetos, ocorre o mesmo; um “objeto” herda as características dos ancestrais. Podendo ser uma herança direta, que é o caso de um filho que herda as características do pai, ou indireta, que são todos os demais casos de herança.

EX:

SER VIVO > ANIMAL > MAMÍFERO > GOLFINHO

Golfinho herda as características de Mamífero (herança direta) e as características de Animal (herança indireta).

Formas de relacionamentos em herança:  
Extensão: A superclasse permanece inalterada, enquanto a subclasse estende a superclasse. Também referenciado como herança estrita.  
Especificação: A superclasse não implementa nenhuma funcionalidade, ela apenas especifica o que a subclasse deve oferecer. Apenas a interface da superclasse é herdada pela sbclasse, isto é, apenas o conjunto de especificação dos métodos públicos.  
Combinação de extensão e especificação: A subclasse herda a interface e uma implementação padrão de métodos da superclasse. A subclasse pode redefinir métodos da superclasse para especializá-los ou oferencer a implementação de um método apenas declarado pela superclasse. Esse comportamento pode ser denominado herança polimórfica.

Algumas linguagens possuem também outra forma de relacionamento:  
  
Contração: A subclasse elimina métodos da superclasse com o objetivo de criar uma “classe mais simples”. Para realizar a eliminação, ocorre a redefinição de métodos com o corpo vazio. Porém esse mecanismo viola os princípios da substituição, pois uma subclasse não poderá ser utilizada em todos os pontos onde a superclasse puder ser utilizada. Esse mecanismo deve ser evitado, se for detectada sua necessidade, o melhor a fazer é reanalizar a hierarquia de classes e detectar as inconsisências.

**Polimorfismo:** Podemos resumir o polimorfismo como a alteração do funcionamento de um método que foi herdado do ancestral. Para melhor compreensão do conceito, segue o exemplo abaixo:

EX: Supondo uma classe Peixe e outra Cachorro, ambos são herdeiros da classe “Animal”, com um método “Movimentar”, porém esse método se comporta de maneiras diferentes para as chamadas ao Peixe e para as chamadas ao Cachorro, uma vez que um Peixe e um Cachorro se movimentam de maneira diferente. Podemos dizer que o método “Movimentar” é um método polimórfico.

Gif DE PEIXE, CACHORRO, MOVIMENTANDO

**Tópico 3 – A Programação Orientada a Objetos usando JAVA**

Os quatro conceitos chaves da Programação Orientada a Objetos são implementados em Java de maneira simples.

**Abstração e Encapsulamento**

A **Abstração** em java é implementada na forma de classes, possuindo atributos e métodos.

EX:

class Pessoa{

//atributos de Pessoa:

String nome;

String idade;

String rg;

void anda(){ //método de Pessoa

...

}

void fala(){ //método de Pessoa

...

}

}

**Modificadores de acesso**

Java possui modificadores de acesso, ou modificadores de visibilidade. Ou seja, é possível limitar quem pode acessar determinado **método** ou **atributo**.

Por exemplo, se eu declarar um atributo como private, ele só poderá ser acessado dentro da sua classe de origem.

Além do modificar de acesso private, existe o public, que permite que um determinado **método** ou **atributo** seja acessada de qualquer classe.

Quando não declarado, o método ou atributo fica em um estado intermediário entre private e public.

O **Encapsulamento** é feito por meio de propriedades privadas, fazendo o uso dos métodos "get", para pegar um valor e "set" para inserir um valor.

EX:

|  |
| --- |
| class Pessoa { //declaração de classe em java  private int rg; //variável pertencente a classe Pessoa  public int GetRg() //Esse método é usado para acessar o valor da variável privada "rg"  {  return rg;  {    public void SetRg(int rg) // Esse método é usado para inserir um valor na variável "rg", uma vez que a mesma é privada. A palavra "this" serve para indicar que o rg referenciado é o da classe atual.  {  this.rg = rg;  }  } |

**Construtor da classe**

**Destrutor de classe**

**Herança em Java**

**Polimorfismo em Java**

**Tópico 4 – Exercícios**

Fontes:

Os 4 pilares da Programação Orientada a Objetos