**Matheus Luiz Massuda**

**Java – Programação Orientada a Objetos**

**PROGRAMAÇÃO ORIENTADA A OBJETOS (POO) – *Object Oriented Programming*** *-* Objetivo da POO é aproximar o mundo digital do mundo real.

**HISTÓRIA – Baixo nível** **(**Na década de 40 a programação era de baixo nível (binário ou decimal).) > **Programação** **Linear** (alto nível, mas modestas sem rotinas e desvios, como se fosse lista de compras) > **Programação** **estruturada** (permitia escrever em linear, mas serem executadas fora de ordem. Pequenos programas evoluíram para pequenos sistemas) > **Programação** **Modular** (módulos estruturas colocados em capsulas) > **POO**.

**Alan Kay 1970 (Matemático e Biólogo) –** Pai da POO. Criou o Smalltalk (primeira linguagem Orientada a Objetos) para poder criar o Dynabook.

**Proposta –** Transformar grandes conglomerados de dados em objetos menores e específicos com métodos que requisitem apenas objetos relacionados (não precisa haver filtro adicional pois esses métodos estão relacionados com determinados objetos). Além disso é possível relacionar objetos entre si, caso seja necessário.

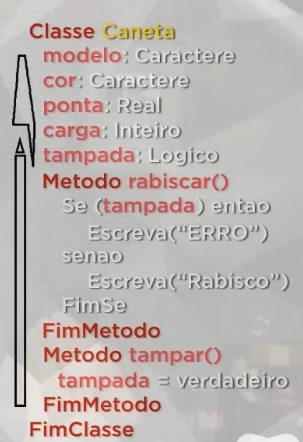
**LINGUAGENS DE PROGRAMAÇÃO OO –** C++; Java; PHP; Python; Ruby; VB; Swift.

**Vantagens da POO – (COMERN**ada)

* **CONFIÁVEL:** Isolamento entre as partes gera software seguro. Ao alterar uma parte, nenhuma outra é afetada.
* **OPORTUNO:** Ao dividir tudo em partes, várias delas podem ser desenvolvidas em paralelo.
* **MANUTENÍVEL:** Atualizar um software é mais fácil. Uma pequena modificação vai beneficiar todas as partes que usarem o objeto.
* **EXTENSÍVEL:** Software não é estático, ele deve crescer para permanecer útil.
* **REUTILIZÁVEL:** Podemos usar objetos de um sistema que criamos em um outro sistema futuro. (Exemplo criei a classe aluno para um colégio e quero reaproveitar essa classe em uma academia de ginástica que também terá alunos).
* **NATURAL:** Mais fácil de entender. Você se preocupa mais com a funcionalidade do que nos detalhes de implementação.

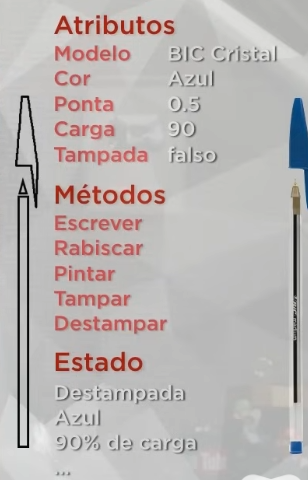
**JDK (JAVA DEVELOPMENT KIT) –** Kit de Desenvolvimento Java (Oracle).

**CLASSE –** É um conceito abstrato que define um objeto. É um molde / forma.

**OBJETO (O QUE É) –** É uma coisa **material** ou **abstrata** que pode ser percebida pelos sentidos e descrita por meio de **características, comportamento e estado** atual. Todos os objetos devem possuir **atributos (características, propriedades, dados). Métodos (processos, comportamentos, rotinas internas). Estado (estado atual).**

1. **ATRIBUTO - Coisas que eu tenho?** Modelo / cor / tipo de ponta / carga / tampa / botão
2. **MÉTODO - Coisas que eu faço?** Escreve / rabiscar / pintar / Tampar / Destampar
3. **ESTADO - Como estou agora?** 50% carga / Sem carga

Exemplo: caneta (embora de várias cores, possuem uma mesma classificação “forma”)

**INSTANCIAMENTO:** é a geração de um objeto, a partir da classe “forma”. Estou instanciando uma classe em forma de um objeto. **Caneta1 = nova Caneta**

**Atributos** definem características (valores) para os objetos, exemplo: caneta1.cor=”Azul” ou caneta1.tampada=”falso”.

**Método** definem uma rotina ou uma função para o objeto (são demarcados por parênteses). Exemplo: caneta1.rabiscar()

**DEFINIÇÃO (BIBLIOGRAFIA) – CLASSE X OBJETO**

* **CLASSE:** é um conceito abstrato que define **atributos** e **métodos** comuns que serão compartilhados por um objeto.
* **OBJETO:** é uma **instância** de uma classe.
* **ABSTRAÇÃO:** A partir de objetos muito complexos, relacionar os atributos que são importantes para aquele **momento ou contexto.**

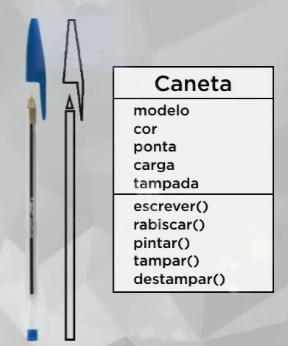
Exemplo: podemos ter uma classe (forma de biscoitos de árvore de natal), podemos fazer e usar glace ver para colorir uma árvore e glace azul para colorir outra árvore. Assim o atributo seria:

**biscoitoArvore.corGlace=”verde”** e  **biscoitoArvore.corGlace=”azul”**

**biscoitoArvore.baseMassa=”baunilha”** e **biscoitoArvore.baseMassa=”chocolate”**

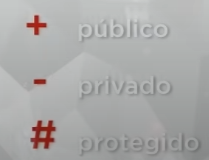
**This.cor == This é uma referência ao próprio objeto que chamou.**

**UML (LINGUAGEM DE MODELAGEM UNIFICADA): DIAGRAMA DE CLASSES –** Classes são retângulos com 3 divisões: Nome, Atributos e Métodos.

**Nome (Classe):** Começa com a primeira letra maiúscula.

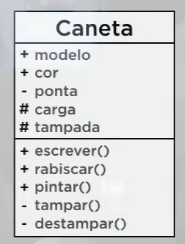
**Atributos:** características / propriedades (começa com minúsculas).

**Métodos():** Começam com minúsculas e possuem parênteses.

****

**VISIBILIDADE DE UM OBJETO –** Indica o nível de acesso aos componentes internos (atributos e métodos) de uma classe.

* **Público (+) (atributo / método):** A classe atual e todas as outras classes podem ter acesso à ela.
* **Privado (-) (atributo / método):** Somente a classe atual pode acessar.
* **Protegido (#) (atributos / métodos):** Permite acesso à classe atual e suas filhas (sub-classes).

**\*IMPORTANTE:** É possível acessar atributos e métodos privados e protegidos por meio de métodos públicos. É como se esse método público fizesse uma ponte para acessar o método privado.

**MÉTODOS ESPECIAIS (SUPORTADO POR TODAS LINGUAGENS ORIENTADAS A OBJETOS) –**

1. **MÉTODOS ACESSORES (Getters):** Pega / acessa determinada informação. Exemplo, estou no caixa eletrônico e quero consultar meu saldo: **get.Saldo** (irá pegar meu saldo). Ou seja, get (método Getter) acessando o atributo Saldo.

O método get que **protege** um objeto. Ele não da acesso direto a uma requisição, mas funciona como um intermediário. Permitindo então acesso ao atributo, mas sem ser um acesso direto ao atributo.



1. **MÉTODOS MODIFICADORES (Setters):** Passam / entregam determinadas informações.Modificam informações dentro do sistema. **e = setTotDoc(doc);**
2. **MÉTODOS CONSTRUTORES (Construct):** Permite automatizar a construção (instanciamento) de um objeto de forma mais fácil. **C1 = new Caneta (“BIC”, “Azul”, 0.5);**

**\*IMPORTANTE =** Para cada atributo de um objeto teremos um método get e outro set para ele.

Exemplo de caso (classe estante foi criada) e vamos instanciar uma nova instante com comando **e = new Estante.** Chega uma pessoa e quer saber quantos documentos pertencentes a ela estão na estante. **t = e.totDoc.** get.totDoc.