<https://github.com/azold6?tab=repositories>

fazer um bate papo online

criando chat online em java

“Na prática, a teoria é outra.”

**Sumário {** 1 – Modulo 1

1 – Conteúdo Base

2 - POO

2 – Modulo 2

1 - Método construtor

2 - Modificadores de acesso

3 – Static

3 – Modulo 3

1 - Heranças

2 - Sobrecarga de métodos

3 - Hierarquia de classes

4 – Modulo 4

1 - Sobrescrita de métodos

2 - FINAL

3 - Classe abstract

4 - Interface

5 – Polimorfismo

5 – Modulo 5

1 - Collections

2 - Arraylist

3 - Interface List

4 - Herança dupla

5 - Equals e HashCode

6 - InterfaceMap

7 - Tratamento de execeções

6 - Modulo 6

1 - Padrões de Projeto

2 - Singleton

3 - Simple Factory

4 - Abstract Factory

5 – Adapter

7 - LEITURA COMPLEMENTAR

**}**

**Conteúdos Base**

Fala muito sobre a preparação do ambiente em java, da sua peculiaridade em criação de um projeto e deixar bem organizado em pastas, da importância da nomenclatura das pastas e documentos em java, que seja necessário o nome das pastas estarem em letras minusculas e os documentos em java com letras maiusculas.

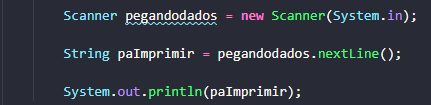
A seguir o conteúdo base de qualquer iniciação em uma nova linguagem de programação, passando pelos conteúdos de declaração de variável, condicionais e estruturas de repetição, que por sinal a sintaxe se assemelha muito com a linguagem “C”. Além desses conteúdos bases também menciona vetores e matrizes(que também são chamadas de vetores bidimensionais).

**Lendo dados - vulgo scanner:**

ePrimeiro é necessário exportar o scanner, dessa forma:

****

logo em seguida:

****

**Obs**: Você cria o Scanner da seguinte forma como é mostrado na 1° linha, na 2° linha é passado o tipo de dado que você quer receber e o nome que vai ter esse dado, junto com o scanner sendo executado, para diferentes tipos de dado vai ser diferentes tipos de método, dados do tipo String são usados o nextLine(); , pra dados do tipo inteiro são usados o tipo nextInt();

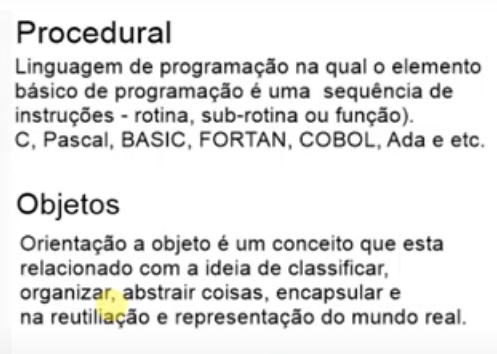
**POO**

É uma forma de deixar o código um pouco mais organizado, criando classes que se referem a um objeto, esse objeto vai ter **atributos** e **métodos**. Também se torna mais fácil dar manutenção no código programando dessa forma.

Sendo os **atributos** às **características** desse objeto.

E métodos sendo as **ações** desse objeto.

**\*Paradigma OO vs Procedural:** Na programação orientada a objeto o atributos e métodos estão dentro do mesmo objeto, sendo que na procedural estão normalmente separados



**Leitura complementar:** <https://blog.betrybe.com/tecnologia/poo-programacao-orientada-a-objetos/#:~:text=Foi%20ent%C3%A3o%20que%20surgiu%20a,%2C%20objetos%2C%20atributos%20e%20m%C3%A9todos>.

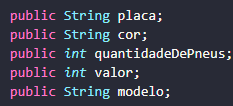
**Método construtor:** Todo método contrutor tem o mesmo nome da classe, e todo método contrutor não tem tipo de retorno



**obs:** Nesse exemplo foi aplicado se baseando na aula dele, sendo a classe Carro que possuia todos os atributos de carro e estava sendo construída na classe principal. Sendo o new ali responsável por instanciar um objeto em memória

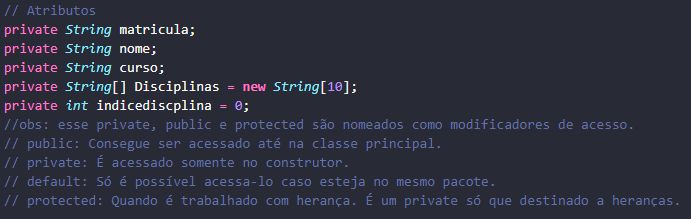


**obs:** Essa parte é tão importante quanto a criação, é aqui que você tem que se atentar onde está a classe que você queira buscar, sem o import é impossível construir ela em outro lugar, sendo ali os “componentes” a pasta em que está a classe “Carro”.



**obs:** Esse são os atributos de carro, começando com o primeiro elemento “public”, que é o **modificador de acesso**, responsável por designar como esse atributo vai ser acessado posteriormente. Como segundo elemento vem o **tipo** dessa varíavel e como terceiro o **nome** dela. Isso é instanciar !!

**\*Tipos de modificadores de acesso:**

****

**Obs:** private: É acessado semente pelo\*\*\* construtor.

**Static:** Tanto variável quanto método, passado o static aquilo se torna somente e exclusivamente da classe.

**Heranças**

**Definição:** É um reaproveitamento de atributos e métodos “herdados” de uma classe pai definida pelo desenvolvedor, suas classes filhos herdam todas caracterísicas e ações da classe pai.

**obs:** Na aula seguinte ele apresenta como passar alguns elementos instanciados como obrigatoriedade no contrutor da classe pai, isso é algo simples, é só criar um método passando o mesmo nome da classe e nos seus parâmetros o que é pra ser setado como obrigatoriedade.(a sintaxe de como é feito está no código)

**\*Sobrecarga de Métodos:** A criação de dois ou mais métodos com nomes totalmentes iguais em uma classe no entanto com parâmetros diferentes. Esse conjunto de parâmetros que vai definir qual método está sendo chamado.

**Sobrescrita(@override):**

**\*Hierarquia de Classes:**

**\*Final:** Uma vez que você atribui o final em algo, é aquilo e acabou, é impossível de ser modificada. Impossível de alteração, sendo assim, se é usado em um método ou atributo isso fica como **impossível de alterações**. Usando o final em classes... fica impossível de outras classes herdarem seus atributos e métodos.

**Classe abstract**: Um modelo do que está propondo, **não é possíver ser instanciado**, **apenas** servir como herança e outros objetos herdar seus atributos e métodos.

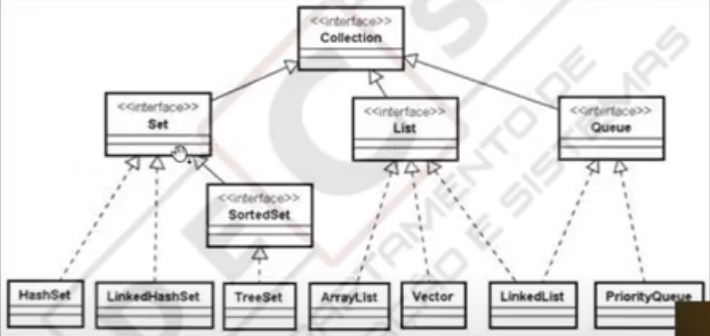
**Método abstract:** Todos os objetos que herdarem de uma classe necessariamente vão ter que criar o método que está sendo passado como abstract.

**\*Interfaces:** É uma classe que possui todos os seus métodos e atributos como obrigatoriedade, funciona exatamente igual a classe abstract no entanto com a obrigatoriedade de crianção dos seus métodos nas classes que herdam dela.

**\*Polimorfismo:** Na definição literal significa várias formas. Usar o mesmo elemento de formas diferentes.

**Collections**

**Definição:** Uma estrutura de dados que permite armazenar vários objetos. Sendo dividido em três que são: Listas, Conjuntos, Mapa(ou dicionário).

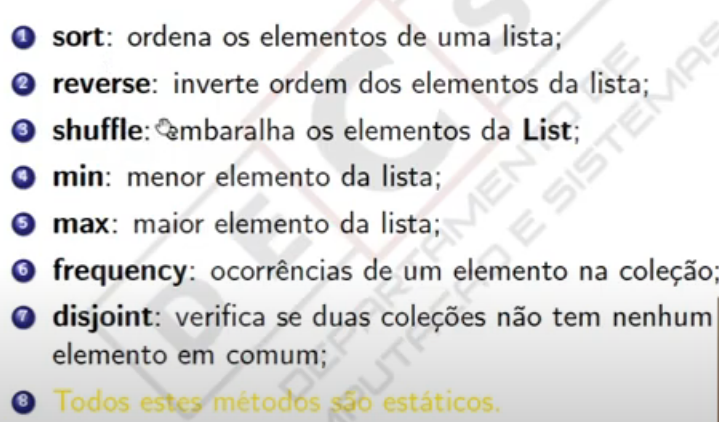


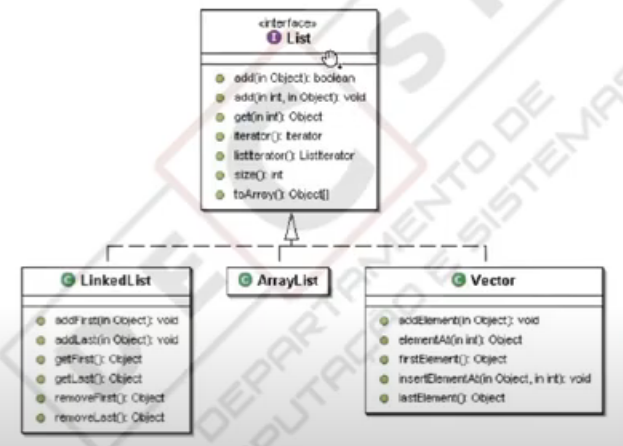
**Listas:**

**ArrayList:** Serve para conter uma lista de objetos.

**LinkedList:** A única diferença entre ele e ArrayList vai estar no desempenho computacional

**Obs:** Podendo assim usar a ideia de polimorfismo, já que ambos implementam a interface List, você só passa como List no inicío e durante a codificação vai aplicando em casos específicos qual é a melhor alternativa (ArrayList ou LinkedList). Um usa vetores e outro listas ligadas.





**Interface List:** Coleção de objetos que não garante ordenação e também não permite elementos duplicados.

**Herança Dupla:** Uma classe que está sendo herdeira de duas classes distintas

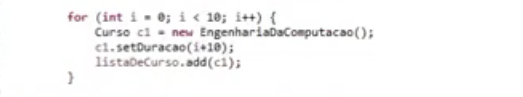


**Equals & HashCode**: Necessários suas implementações para saber quando dois objetos são iguais.

**Interface Map:** Um map é uma associação de chaves e valores, chaves possuem a necessidade

Dica: Sempre que criar uma classe por precaução já defina os métodos, equals, hashcode e

**Tratamento de Exceções:** Uma forma de contorna alguns erros durante a execução do código, usando o **try** {Vai tentar executar esse bloco}, **cath**{Caso não consiga executar o bloco try, o bloco cath irá ser executado logo em seguida}



Criando e alocando objetos em arrays de maneira dinâmica

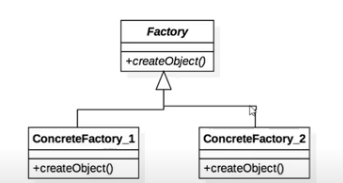
**Padrões de Projeto**

**Singleton:** Faz parte de um padrão de projeto que deve ser seguido, no caso criando um objeto que impede que o contrutor seja repetido, garantir que exista uma única instância.

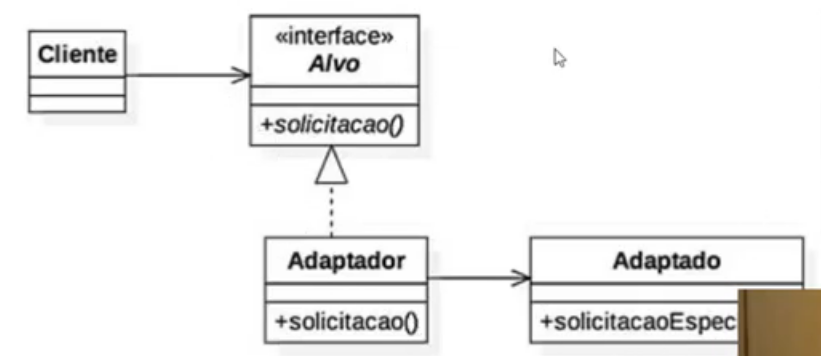
Impede que exista duas instâncias, apenas uma...

**Static:** torna o método algo da classe e não do objeto, sendo assim, não precisa iniciar o construtor e só passar ele como referencia.

**Factory Method:** Pra diferentes tipos de situações

****

**Adapter:**



Adaptação de um projeto que possui uma determinada interface, pra um outro projeto que possui uma outra interface, usando a ideia de polimorfismo.

**LEITURAS COMPLEMENTARES:**

Modificadores de Acesso {  
<https://www.devmedia.com.br/modificadores-de-acesso-do-java/27065>

<https://www.alura.com.br/apostila-java-orientacao-objetos/modificadores-de-acesso-e-atributos-de-classe#getters-e-setters>

<https://medium.com/trainingcenter/modificadores-de-acesso-3f87133611c8>   
}

Static {  
<https://www.web-dev-qa-db-br.com/pt/java/diferenca-entre-metodos-estaticos-e-metodos-de-instancia/1067631353/>   
}

Herança {

<https://www.alura.com.br/apostila-java-orientacao-objetos/heranca-reescrita-e-polimorfismo#repetindo-codigo>

<https://medium.com/caiquefortunato/heran%C3%A7a-em-java-teoria-e-pr%C3%A1tica-2ca7d9b0f3de>

}

SobreCarga de Métodos {  
<https://www.devmedia.com.br/introducao-sobrecarga-de-metodos-e-tipos-genericos-em-java/22853>   
}

Class Object{  
<https://www.devmedia.com.br/java-object-class-entendendo-a-classe-object/30513>   
}

Sobrescrita de métodos {  
<https://cursos.alura.com.br/forum/topico-sobrescrita-e-sobrecarga-de-metodos-qual-a-diferenca-e-para-o-que-servem-73090>   
}

FINAL {  
<https://www.devmedia.com.br/modificadores-de-acesso-do-java/27065>   
}

Classes Abstratas {  
<https://www.alura.com.br/apostila-java-orientacao-objetos/classes-abstratas#classe-abstrata>   
}

Interface {  
<https://www.devmedia.com.br/java-interface-aprenda-a-usar-corretamente/28798>   
}

Polimorfismo {  
<https://www.alura.com.br/apostila-java-orientacao-objetos/heranca-reescrita-e-polimorfismo>

<https://www.devmedia.com.br/encapsulamento-polimorfismo-heranca-em-java/12991>   
}

Polimorfismo em interface {  
<https://medium.com/@danielchristofolli/heran%C3%A7a-polimorfismo-e-interfaces-java-4d2728b58924>   
}

Collections {  
<https://www.devmedia.com.br/java-collections-como-utilizar-collections/18450>   
}

ArrayList {  
<https://www.devmedia.com.br/explorando-a-classe-arraylist-no-java/24298>

<https://www.devmedia.com.br/usando-generics-em-java/28981>   
}

Ordenação de Objetos {  
<https://www.alura.com.br/artigos/ordenando-uma-lista-de-objetos-em-java>   
}

Interface set {  
<http://www.linhadecodigo.com.br/artigo/3669/trabalhando-com-a-interface-set-no-java.aspx>   
}

Interface Map {  
<https://www.devmedia.com.br/conhecendo-a-interface-map-do-java/37463>   
}

Tratamento de Exeções {  
<https://www.devmedia.com.br/conhecendo-a-interface-map-do-java/37463>   
}

Padrões de Projeto {  
<https://www.devmedia.com.br/conheca-os-padroes-de-projeto/957>   
}

Singleton {  
<https://www.devmedia.com.br/trabalhando-com-singleton-java/23632>   
}

Simple Factory {  
<https://www.thiengo.com.br/padrao-de-projeto-simple-factory>   
}

Factory Method {  
<https://www.thiengo.com.br/padrao-de-projeto-factory-method>   
}

Abstract Factory {  
<https://www.thiengo.com.br/padrao-de-projeto-abstract-factory>   
}

Adapter {  
<https://www.devmedia.com.br/padrao-de-projeto-adapter-em-java/26467>   
}