O conteúdo digital começa falando de **particionamento**, nos primeiros exemplos não temos nada de novo, apenas declarações de classe e agregação, o que objetivamente é a reunião de classes para formar novas classes. Como descrito no exemplo foi agregado a classe Desktop as demais classes que formariam um desktop comum como; placa de vídeo, memória, hd e etc.

O conceito de agregação é novamente aplicado mais a frente, dessa vez é descrito um conceito que até então era novo pra mim, o conceito de validadores, segue exemplo:

public void setMarca( String ma ) {

if(!ma.isEmpty()) {

marca = ma;

Tendo praticado mais agregação, é apresentada a aplicação do conceito de herança que é uma forma de criar novas classes a partir de classes que já existem, se aproveitando das características que já existem na classe que será herdada:

public class Armazenamento extends Identificacao{

public String tipo;

public int capacidade;

public Armazenamento( ) {

super();

Ao ler o código, foram criadas todas as classes de forma similar aos exemplos anteriores e encontramos uma nova chamada, a super() que representa uma chamada de método que invoca o construtores dentro das superclasses “Armazenamento e Identificação”.

A estrutura do código como um todo é muito semelhante com as que temos trabalhado em sala de aula.

ENCAPSULAMENTO-

O encapsulamento é um processo que permite uma classe encapsular atributos e seus métodos, ocultando os detalhes de implementação dos objetos. Logo abaixo surge um exemplo de diversos métodos para calculaimposto()

No primeiro método foi criaca a classe calcularImposto ifs e elses para verificações e mais abaixo foram usados operadores aritméticos para se calcular os impostos

public double calcularTipoProduto1(double valor, int faixa) {

// verificar as faixas e realizar o cálculo correto

if(faixa == 0) {

imposto = valor \* 12.5 / 100;

}

else {

if(faixa == 1) {

imposto = valor \* 12.8 / 100;

}

else {

imposto = valor \* 13.1 / 100;

}

}

return imposto;

e no ultimo exemplo é criada uma classe main e importado o Scanner para entrada de dados e uma série de System.out.println para exibir a execução do programa.

Classes definidas como public podem ser acessadas por qualquer classe ou aplicação sem qualquer restrição.

Classes definidas como PRIVATE só pode ser acessadas por membros da própria classe.

Um membro definido como PROTECTED pode ser acessado apenas por membros da própria classe, por outras classes, aplicações que estejam no mesmo package ou das suas subclasses.

Quando não define sua classe com nenhuma das opções acima a classe é definida como default e com isso ela se torna publica.

Logo abaixo é exibido três scripts mostrando bem esses conceitos.

No ultimo momento do conteúdo digital é apresentado os modificadores static e final.

Nesse primeiro código é criada uma classe chamada RefThis. Um script muito parecido com os anteriores, declaração dos atributos e logo depois é usado o this no retorno dos métodos. O this, no caso, faz referencia a si mesmo, alterando a identificação do objeto pelo seu identificador (nome, idade e peso).

**Nos scripts Classe Cotacao e Classe Exemplo1** é importado o Scanner para entrada de dados, então é criada a classe Cotacao, a declaração de seus atributos, mais classes para verificações, seus getters and setters e também o uso do método (Double.parseDouble(entrada.nextLine) que converte a entrada em string para double, o entrada.nextLine indica que o scanner lerá a entrada até encontrar uma quebra de linha.

Então no Classe Exemplo 1 é importada a classe cotação que está no mesmo pacote, essa classe é a main desse programa que logo no inicio já é criado objetos(cot\_1, cot\_2) com a classe (Cotacao) descrita anteriormente, o programa manipula as informações escritas no script anterior e dá sua saída (System.out.println).

As classes seguintes seguem um padrão parecido com os outros scripts. Na **Classe Teste** é criada uma classe e um método com a adição do método Teste() com um atributo, mais incremento (contador++).

**Já na Classe Exemplo2,** foram criados cinco objetos mas não foi feito nenhum acesso direto para o *contador,* cada objeto quando criado, incrementava o valor.

No script **MetodosCompartilhados** vemos a aplicação de mais métodos estáticos. Nesse primeiro momento o método Math.pow foi identificado pela classe *valor* para ser referenciado no método main presente na classe Exemplo3. No **Exemplo3** é a classe main, onde é escrita a saída do programa e declaração desses métodos (Math.pow , Double.parseDouble e Integer.parseInt).

O exemplo seguinte mostra o uso da Constante (método final)

Public final int valorDobro(){

Return valor\*2;

}

Dessa forma, impedindo que a classe SubClasse pudesse sobescrever esses métodos.

O ultimo Script que aborda o conceito de classes abstratas. Uma classe abstrata é uma classe que funciona como modelo para outras classes. No exemplo, temos a classe pai **AssaAquece** com seus setters and getters, criado seus métodos. Depois são criadas as subclasses e ligadas usando herança *extend*, e então é criado um novo pacote **aplicação** , uma classe **Comida**, é importado então o pacote contendo as subclasses e a classe pai, um método main e criação de novos objetos para a saída.

Nome: Elvis Moises Melo de Oliveira

Matricula: 202002410931