Google olabi

<ORIENTAÇÃO A OBJETOS EM JAVA>

{ Orientação a objetos em Java

Nos desenvolvimentos de sistemas, existem alguns fatores importantes como: entendimento do código, fácil manutenção, reaproveitamento, entre outros. Para isso, a Programação Orientada a Objetos (POO) tem a intenção de ajudar nesses fatores, dando tempo e agilidade no desenvolvimento de um sistema para o programador.

A POO é formada por alguns itens, dentre os quais se destacam: classes, objetos, atributos, métodos e construtores.

Tudo em Java está associado a classes e objetos, juntamente com seus atributos e métodos. Assim, é primordial saber a orientação a objeto, para que seja possível aproveitar ao máximo o que o Java tem a oferecer.

<classes>

As classes são projetos de um objeto. Possuem características e comportamentos que permitem o armazenamento de atributos e métodos dentro delas.

Características das classes:
- Toda classe possui um nome.
- Possuem visibilidade. Exemplo: public, private, protected.
- Possuem membros como: Características e Ações.

Para criar uma classe, basta declarar conforme demonstrado a seguir:

```
public class Teste{
 //Atributos ou propriedades
 //Métodos
}
```

<Objetos>

Os objetos são características definidas pelas classes e neles é permitido instanciar objetos da classe para inicializar os atributos e invocar os métodos.

Todo objeto é algo que existe, é uma coisa concreta; enquanto a classe é considerada como um modelo ou projeto de um objeto, sendo algo que não se consegue tocar.

 = Objeto

 = Classe

<Atributos>

Os atributos são as propriedades de um objeto e também são conhecidos como variáveis ou campos. Essas propriedades definem o estado de um objeto, fazendo com que esses valores possam sofrer alterações.

<Métodos>

Os métodos são ações ou procedimentos que podem interagir e se comunicar com outros objetos. A execução dessas ações se dá através de mensagens, tendo como função o envio de uma solicitação ao objeto para que seja efetuada a rotina desejada.

Como boas práticas, é indicado sempre utilizar o nome dos métodos declarados como verbos. Assim, quando for efetuada alguma manutenção, essa será de fácil entendimento.

Algumas nomenclaturas de nomes de métodos estão listadas a seguir:

- acaoVoltar
- voltar
- correr
- resgatarValor
- pesquisarNomes

<Construtores>

Os construtores, ou constructors, são os responsáveis por criar o objeto em memória, ou seja, instanciar a classe que foi definida. O construtor de um objeto é um método especial, pois inicializa seus atributos toda vez que é instanciado (inicializado).

Toda vez que é digitada a palavra reservada new, o objeto solicita para a memória do sistema armazená-lo, onde chama o construtor da classe para inicializar o objeto. A identificação de um construtor em uma classe é sempre o mesmo nome da classe.
Os construtores são obrigatórios e são declarados conforme demonstrado a seguir:

```
public class Teste{
// Construtor da classe Teste
public Teste(){
//Faça o que desejar na construção do objeto
    }
}
```

{ Pilares da Programação Orientada a Objetos

Pilar 1: Abstração

A abstração de dados é a propriedade pela qual apenas os detalhes essenciais são exibidos ao usuário. As unidades triviais ou não essenciais não são exibidas ao usuário. Ex: um carro é visto como um carro e não como seus componentes individuais.
A abstração de dados também pode ser definida como o processo de identificar apenas as características necessárias de um objeto ignorando os detalhes irrelevantes. As propriedades e comportamentos de um objeto o diferenciam de outros objetos de tipo semelhante e auxiliam na classificação/agrupamento dos objetos.
Em Java, a abstração é alcançada por interfaces e classes abstratas. A seguir, é possível observar exemplos de abstrações:

Entidade	Características	Ações
Carro, Moto	tamanho, cor, peso, altura	acelerar, parar, ligar, desligar
Elevador	tamanho, peso máximo	subir, descer, escolher andar
Conta Banco	saldo, limite, número	depositar, sacar, ver extrato

Pilar 2: Encapsulamento

É definido como o agrupamento de dados em uma única unidade. É o mecanismo que une o código e os dados que ele manipula. Outra maneira de pensar sobre o encapsulamento é que ele é um escudo protetor que impede que os dados sejam acessados pelo código fora desse escudo.

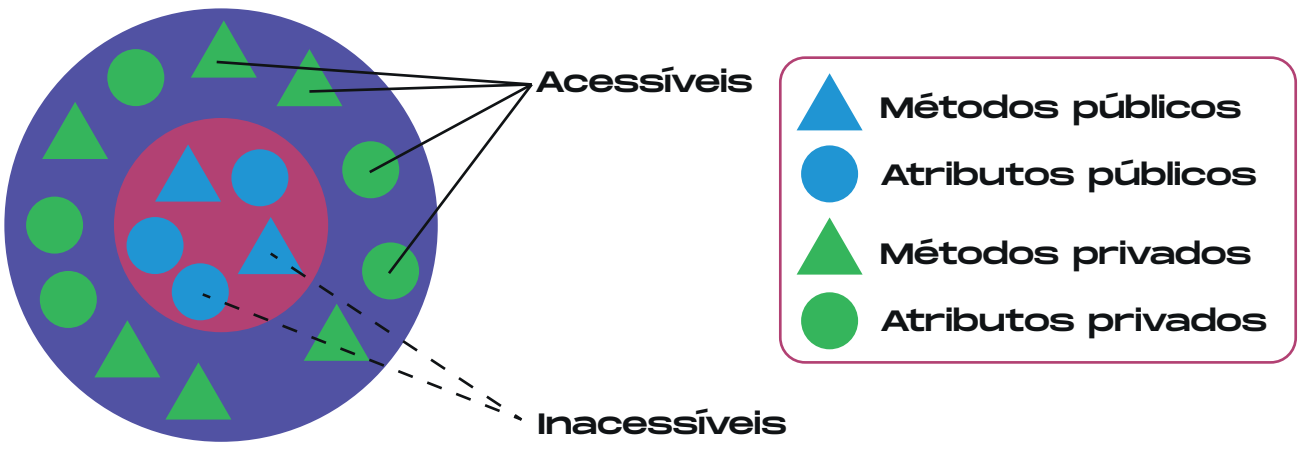
Tecnicamente, no encapsulamento as variáveis ou dados de uma classe são ocultados de qualquer outra classe e podem ser acessados somente através de qualquer função membro da própria classe em que são declaradas.

Assim como no encapsulamento, os dados de uma classe ficam ocultos de outras classes, por isso também é conhecido como ocultação de dados.

O encapsulamento pode ser obtido declarando todas as variáveis da classe como privadas e escrevendo métodos públicos na classe para definir e obter os valores das variáveis.

Em Java, existem 4 tipos de especificadores de acesso: public, protected, private e default.

Public	acessível em todas as classes da sua aplicação.
Protected	acessível dentro do pacote em que está definido e em sua (s) subclasse(s) (incluindo subclasses declaradas fora do pacote)
Private	acessível apenas dentro da classe em que está definido.
Default	(declarado/definido sem usar qualquer modificador): acessível dentro da mesma classe e pacote dentro do qual sua classe está definida.



Para acessar esses tipos de modificadores inacessíveis é necessário criar métodos setters e getters. Por entendimento, os métodos setters servem para alterar a informação de uma propriedade de um objeto. E os métodos getters para retornar o valor dessa propriedade.

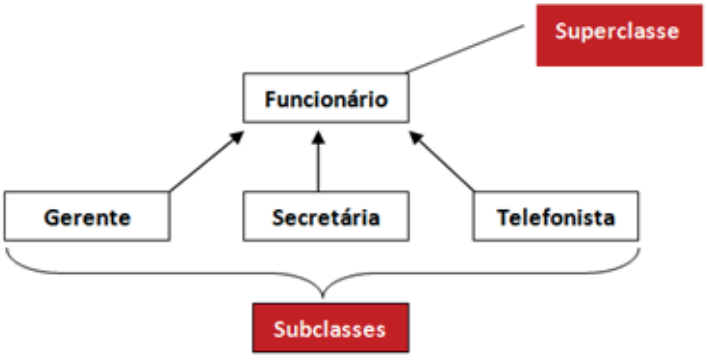
Métodos getters	Métodos setters

public String getNome() {
return nome;
}
 	
public double getSalario() {
return salario;
}
	public void setNome(String nome) {
	this.nome = nome;
	}
	
	public void setSalario(double salario) {
	this.salario = salario;
	}

Pilar 3: Herança

Herança é o mecanismo pelo qual uma classe permite herdar os recursos (campos e métodos) de outra classe. Algumas terminologias importantes usadas em relação à herança estão listadas a seguir:

- **Superclasse:** A classe cujos recursos são herdados é conhecida como superclasse (ou classe base ou classe pai).
- **Subclasse:** A classe que herda a outra classe é conhecida como subclasse (ou uma classe derivada, classe estendida ou classe filha). A subclasse pode adicionar seus próprios campos e métodos além dos campos e métodos da superclasse.
- **Reusabilidade:** A herança suporta o conceito de "reusabilidade", ou seja, quando queremos criar uma nova classe e já existe uma classe que inclui parte do código que queremos, podemos derivar nossa nova classe da classe existente. Ao fazer isso, estamos reutilizando os campos e métodos da classe existente.



Pilar 4: Polimorfismo

Polimorfismo refere-se à capacidade das linguagens de POO de diferenciar entidades com o mesmo nome de forma eficiente. Isso é feito pelo Java com a ajuda da assinatura e declaração dessas entidades.

Em outras palavras, o polimorfismo denota uma situação na qual um objeto pode se comportar de maneiras diferentes ao receber uma mensagem. Isso nos permite realizar uma única ação de maneiras diferentes.

Os principais tipos de polimorfismo são: Polimorfismo estático ou sobrecarga e Polimorfismo dinâmico ou sobreposição.

O Polimorfismo Estático se dá quando temos a mesma operação implementada várias vezes na mesma classe. A escolha de qual operação será chamada depende da assinatura dos métodos sobrecarregados.

O Polimorfismo Dinâmico acontece na herança, quando a subclasse sobrepõe o método original. Agora o método escolhido se dá em tempo de execução e não mais em tempo de compilação. A escolha de qual método será chamado depende do tipo do objeto que recebe a mensagem.

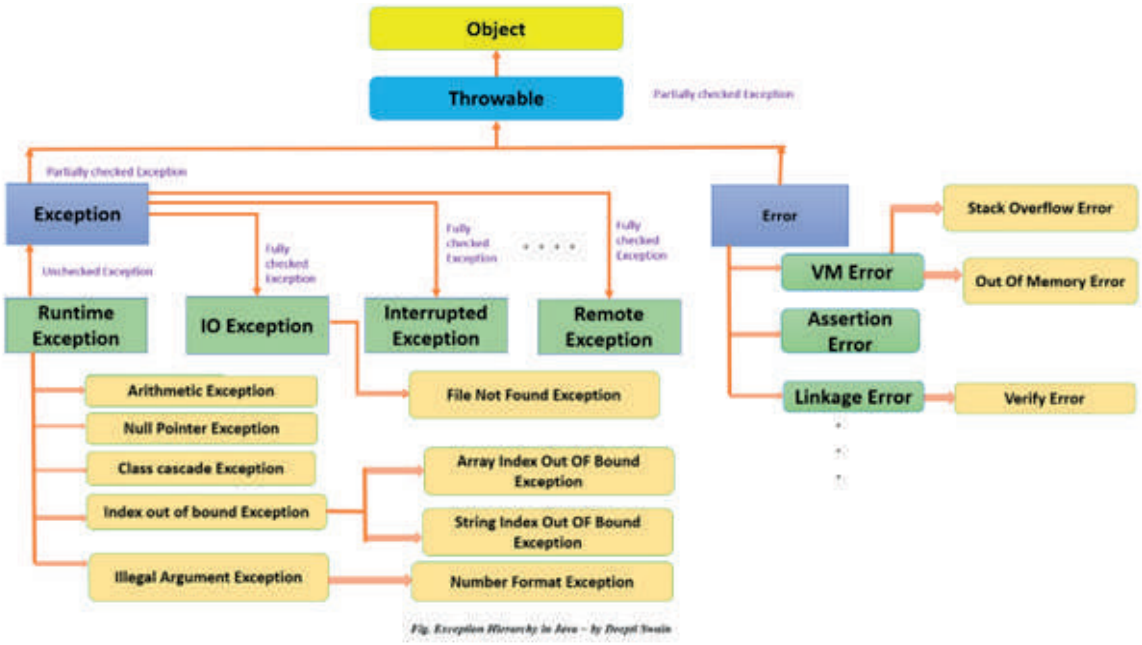
Ex.:

```
class Animal {
  public void animalSound() {
    System.out.println("The animal makes a sound");
  }
}
class Pig extends Animal {
  public void animalSound() {
    System.out.println("The pig says: wee wee");
  }
}
class Dog extends Animal {
  public void animalSound() {
    System.out.println("The dog says: bow wow");
  }
}
```

{ Java Exceptions

Exceção é um evento indesejado ou inesperado que ocorre durante a execução de um programa, ou seja, em tempo de execução e que interrompe o fluxo normal das instruções do programa. Exceções podem ser capturadas e tratadas pelo programa. Quando ocorre uma exceção dentro de um método, ele cria um objeto. Esse objeto é chamado de objeto de exceção. Ele contém informações sobre a exceção, como o nome e a descrição da exceção e o estado do programa quando a exceção ocorreu.

Exceção	Erros
Indica condições que um aplicativo razoável pode tentar capturar	Representam condições irrecuperáveis, como Java Virtual Machine (JVM) ficando sem memória, vazamentos de memória, erros de estouro de pilha, incompatibilidade de biblioteca, recursão infinita, etc. Erros geralmente estão além do controle do programador e não devemos tentar lidar com eles.



Tipos de Exceções

Java define vários tipos de exceções relacionadas às suas várias bibliotecas de classes. Também permite que os usuários definam suas próprias exceções.

As exceções podem ser categorizadas de duas maneiras: incorporadas e definidas pelo usuário.

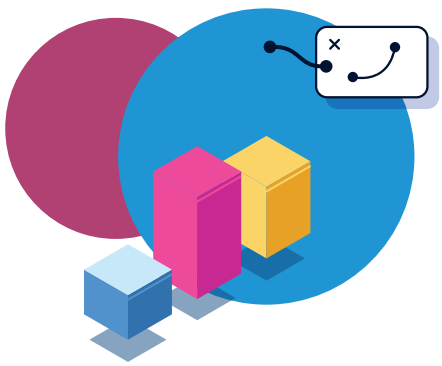
Exceções incorporadas

Exceções incorporadas são as exceções que estão disponíveis em bibliotecas Java. Essas exceções são adequadas para explicar certas situações de erro e podem ser classificadas em verificadas (Checked Exceptions) e não verificadas (Unchecked Exceptions).

Exceções verificadas	Exceções não verificadas
Chamadas de exceções em tempo de compilação porque são verificadas em tempo de compilação pelo compilador.	O compilador não verificará essas exceções em tempo de compilação. Em palavras simples, se um programa lançar uma exceção não verificada, mesmo que não a tratemos ou a declaremos, o programa não apresentará um erro de compilação.

Exceções definidas pelo usuário

Às vezes, as exceções internas em Java não são capazes de descrever uma determinada situação. Nesses casos, os usuários também podem criar exceções que são chamadas de 'Exceções definidas pelo usuário'.



Tratamento de exceção personalizado

O tratamento de exceção Java é gerenciado por meio de cinco palavras-chave: try, catch, throw, throws e finally.

As instruções de programa que você acha que podem gerar exceções estão contidas em um bloco **try**. Se ocorrer uma exceção dentro do bloco **try**, ela será lançada. Seu código pode capturar essa exceção (usando o bloco **catch**) e tratá-la de maneira racional.

Exceções geradas pelo sistema são lançadas automaticamente pelo sistema de tempo de execução Java. Para lançar manualmente uma exceção, use a palavra-chave **throw**. Qualquer exceção lançada de um método deve ser especificada como tal por uma cláusula **throws**. Qualquer código que absolutamente deve ser executado após a conclusão de um bloco **try** é colocado em um bloco **finally**.

<Leituras complementares>

https://www.devmedia.com.br/abstracao-encapsulamento-e-heranca-pilares-da-poo-em-java/26366

https://www.devmedia.com.br/uso-de-polimorfismo-em-java/26140#:~:text=Polimorfismo%20significa%20%22muitas%20formas%22%2C,diferentes%20ao%20receber%20uma%20mensagem

https://www.w3schools.com/java/java_try_catch.asp

<Referências>

https://www.devmedia.com.br/introducao-a-programacao-orientada-a-objetos-em-java/26452

https://www.geeksforgeeks.org/object-oriented-programming-oops-concept-in-java/

https://www.geeksforgeeks.org/exceptions-in-java/?ref=lbp

Google olabi

<ORIENTAÇÃO A OBJETOS EM JAVA>

/exercícios/

1 ➔ Identifique as classes e implemente um programa para a seguinte especificação: “O supermercado vende diferentes tipos de produtos. Cada produto tem um preço e uma quantidade em estoque. Um pedido de um cliente é composto de itens, onde cada item especifica o produto que o cliente deseja e a respectiva quantidade. Esse pedido pode ser pago em dinheiro, cheque ou cartão.”

2 ➔ Faça um programa de agenda telefônica com as classes Agenda e Contato.

3 ➔ Crie uma classe para representar uma pessoa com os atributos privados de nome, data de nascimento e altura. Crie os métodos públicos necessários para sets e gets e um método para imprimir todos os dados de uma pessoa. Crie um método para calcular a idade da pessoa.

4 ➔ Crie uma classe Agenda que pode armazenar 10 pessoas e que seja capaz de realizar as seguintes operações:

- void armazenaPessoa(String nome, int idade, float altura);
- void removePessoa(String nome);
- int buscaPessoa(String nome); // informa em que posição da agenda está a pessoa
- void imprimeAgenda(); // imprime os dados de todas as pessoas da agenda
- void imprimePessoa(int index); // imprime os dados da pessoa que está na posição “i” da agenda

5 ➔ Faça uma classe Ex2Sorteio para:

- Sortear um número de 0 a 1000 (dica: usar Math.random())
- Pedir um palpite ao usuário. Se ele errar, informe se o palpite é maior ou menor do que o número sorteado.
- Pedir novos palpites até que o usuário acerte e, depois disso, mostrar em quantas tentativas ele acertou: lançar a exceção MenorQueException caso o número arriscado seja menor do que o sorteado; lançar a exceção MaiorQueException caso o número arriscado seja maior do que o sorteado.
- Capturar essas exceções e tratá-las, mantendo a lógica original.