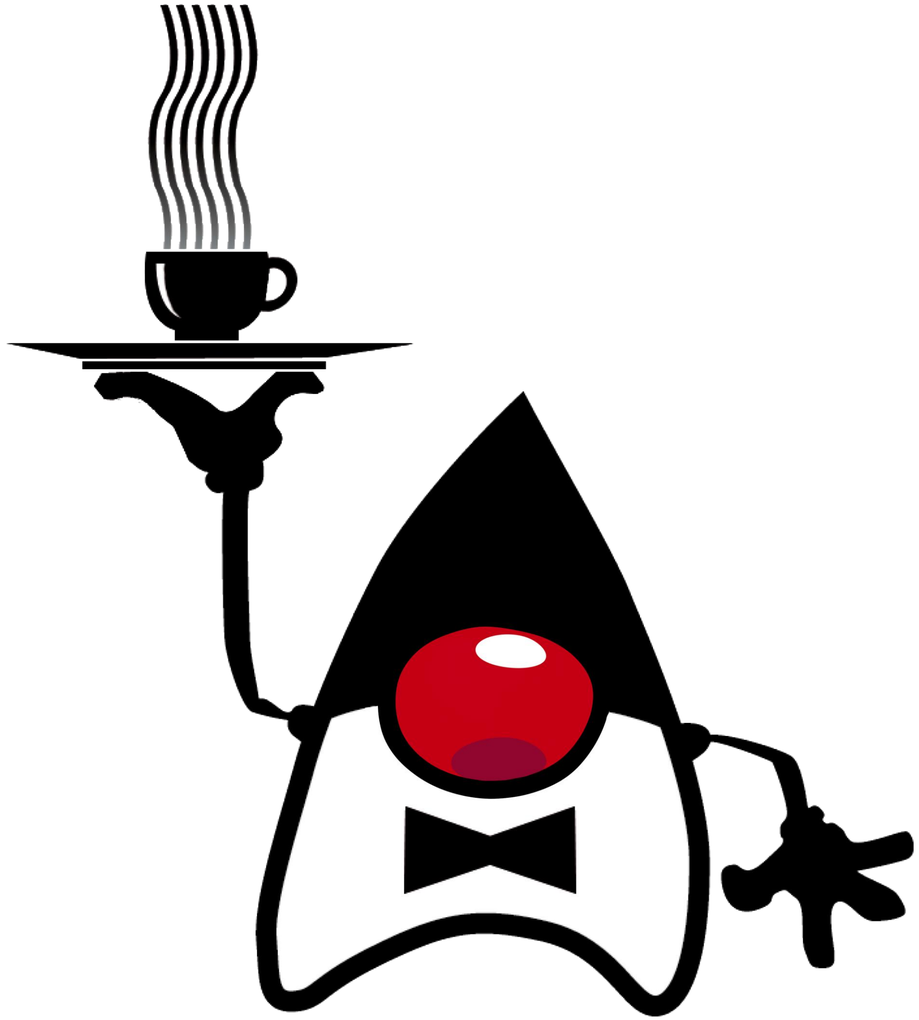
Trilha Java

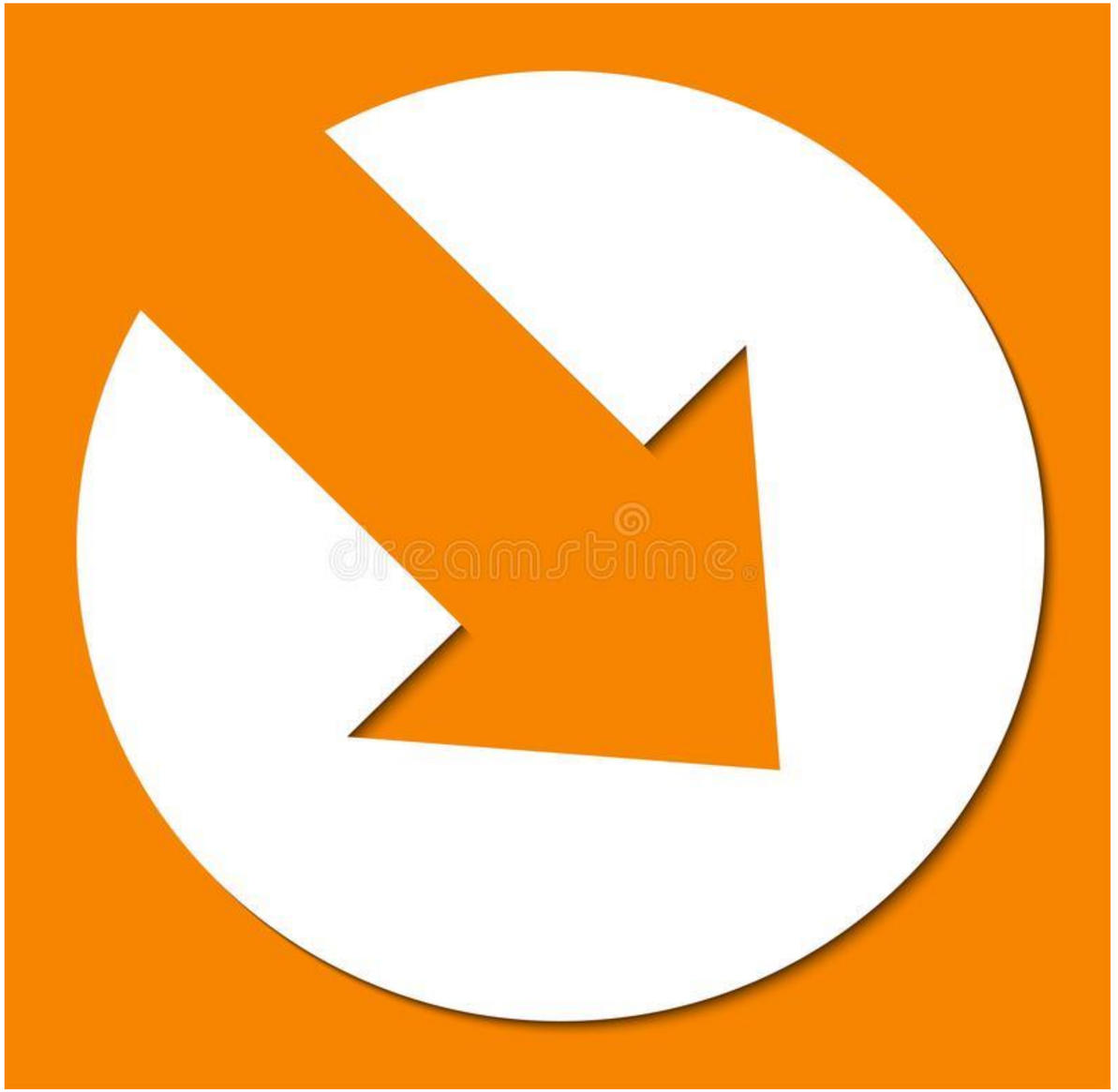
Encontro 18 – Exemplo e Atividade 9



vnt/school
powered by venturus

Recapitulação

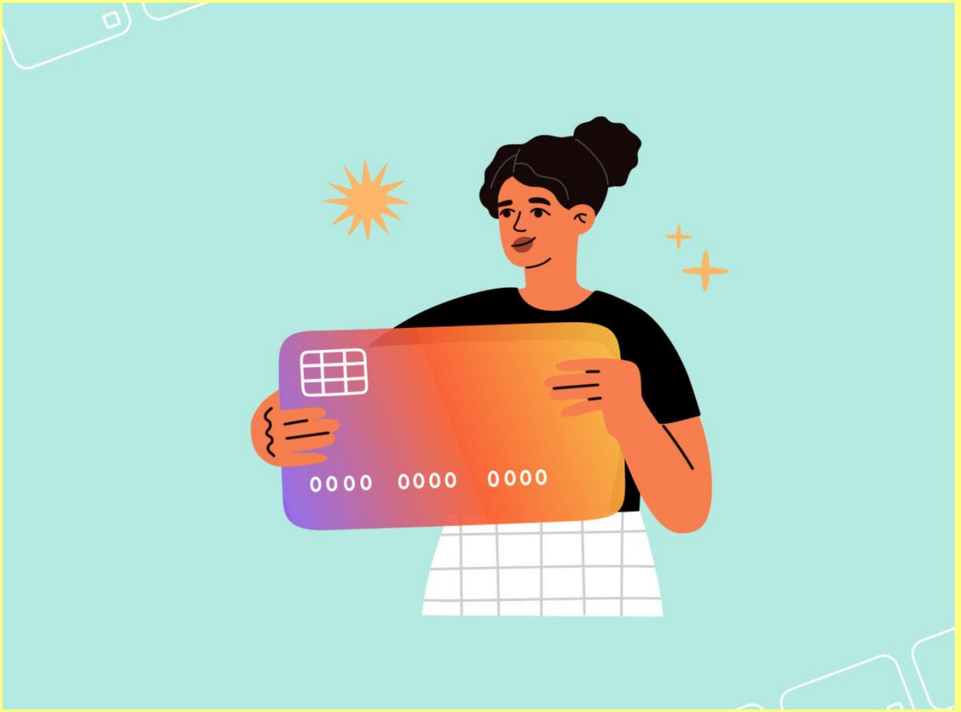
1. Encapsulamento.
2. Relacionamento entre classes.
3. Agregação



vnt/school
powered by venturus

Atividade 1

Em um banco, para se cadastrar uma conta bancária, é necessário informar o **número da conta**, o **nome do titular** da conta, e o valor de **depósito inicial** que o titular depositou ao abrir a conta.
Este valor de depósito inicial, entretanto, é opcional, ou seja: se o titular não tiver dinheiro a depositar no momento de abrir sua conta, o depósito inicial não será feito e o saldo inicial da conta será, naturalmente, zero.

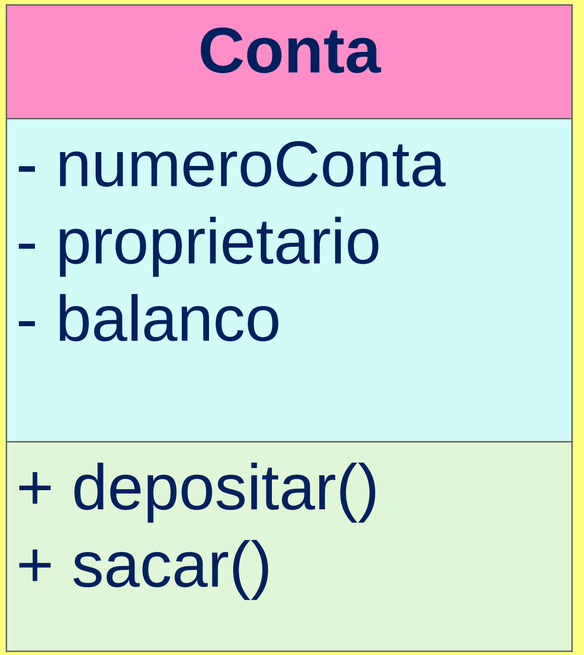


vnt/school
powered by venturus

Atividade 1

Importante: uma vez que uma conta bancária foi aberta, o número da conta nunca poderá ser alterado. Já o nome do titular pode ser alterado (pois uma pessoa pode mudar de nome por ocasião de casamento, por exemplo).

Além disso, o saldo da conta não pode ser alterado livremente. É preciso haver um mecanismo para proteger isso. O saldo só aumenta por meio de depósitos, e só diminui por meio de saques.



Atividade 1

Leve em consideração a seguinte regra:

Para cada saque realizado, o banco cobra uma taxa de $ 5.00.

Nota: a conta pode ficar com saldo negativo se o saldo não for suficiente para realizar o saque e/ou pagar a taxa.

Conta
- numeroConta - proprietario - balanco
+ depostar() + sacar()

vnt/school
powered by venturus

Atividade 1

Diante dos dados apresentados, crie um programa que realize o cadastro de uma conta, dando opção para que seja ou não informado o valor de depósito inicial. Em seguida, realize um depósito e depois um saque, sempre mostrando os dados da conta após cada operação.

Use o conceito de interface/encapsulamento e considere os métodos depositar() e sacar() como uma interface. Isso é importante para o contrato.

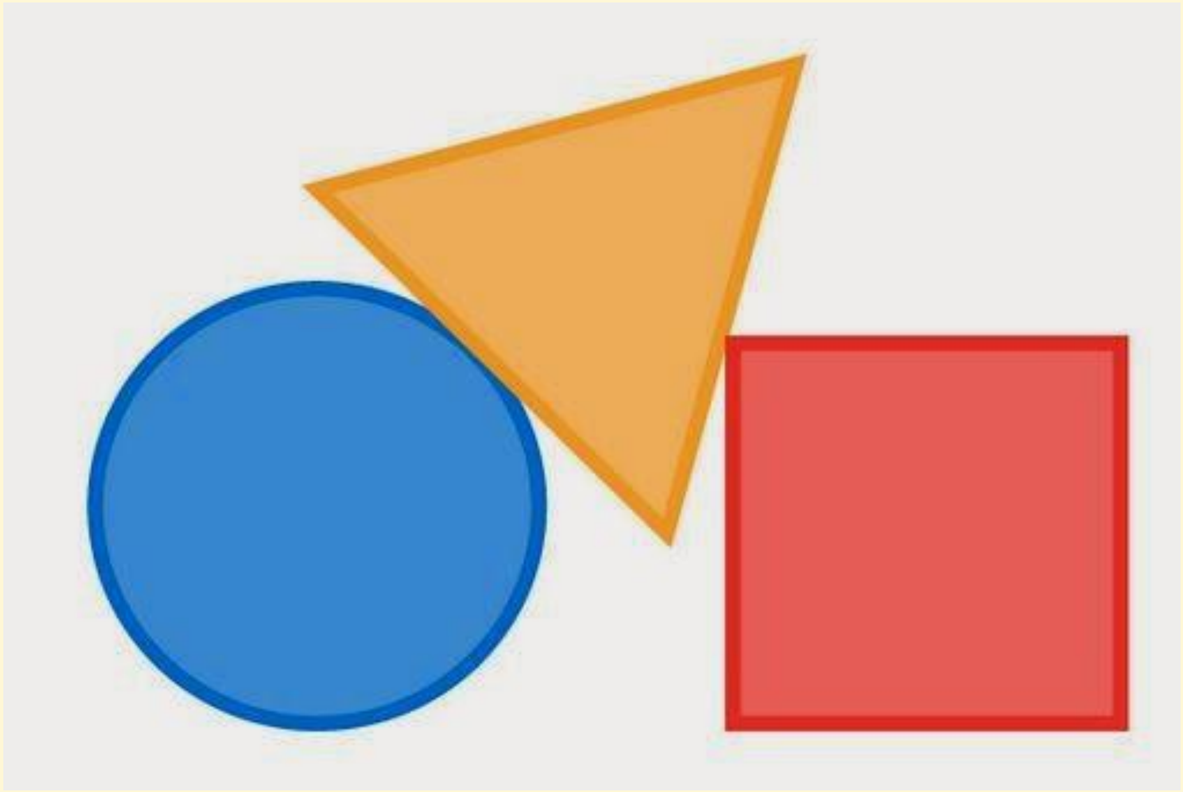
Conta
- numeroConta - proprietario - balanco
+ depostar() + sacar()

<inteface> **ControleConta**
public abstract depositar(); public abstract sacar();

vnt/school
powered by venturus

Atividade 2

O quadrado e o triângulo são figuras geométricas que partilham de algumas características semelhantes. Eles possuem perímetro, área, nome e etc.



Estes valores muitas das vezes são essenciais para o cálculo de muitas outras atividades.

Pensando nestes cálculos, é proposto o seguinte problema:

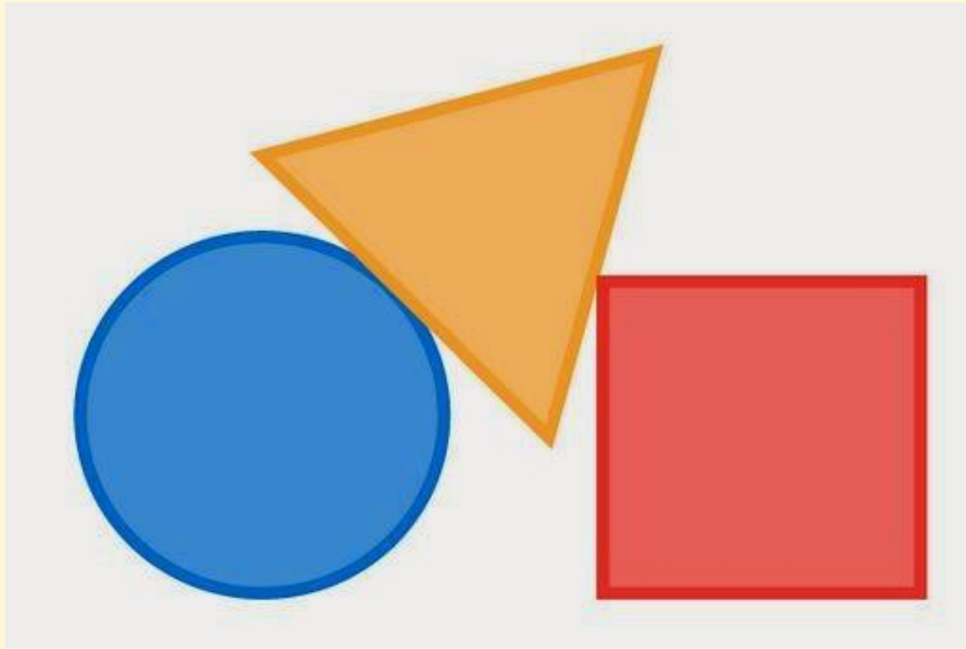
vnt/school
powered by venturus

Atividade 2

Crie um programa que contenha duas classes "Quadrado" e "Triangulo". Estas classes devem possuir suas principais características, como: nome, lado, altura, base e etc.

Faça um contrato através da interface e para tal use os seguintes métodos:

public abstract void **getNomeFigura()**;
public abstract void **getArea()**;
public abstract void **getPerimetro()**;



vnt/school
powered by venturus

Atividade 2

public abstract void **getNomeFigura()**;
public abstract void **getArea()**;
public abstract void **getPerimetro()**;

Além da interface citada acima, construa as classes apresentadas ao lado. Acrescente todos os getters e setters e qualquer outra informação que achar relevante nas respectivas classes.

Por fim, instancie os objetos "quadrado" e "triangulo". Apresente o resultado.

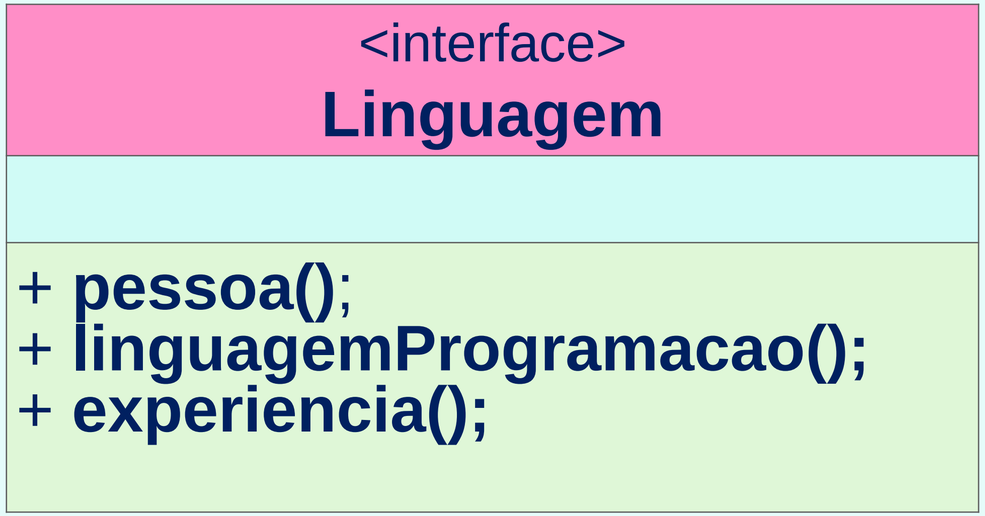
vnt/school
powered by venturus

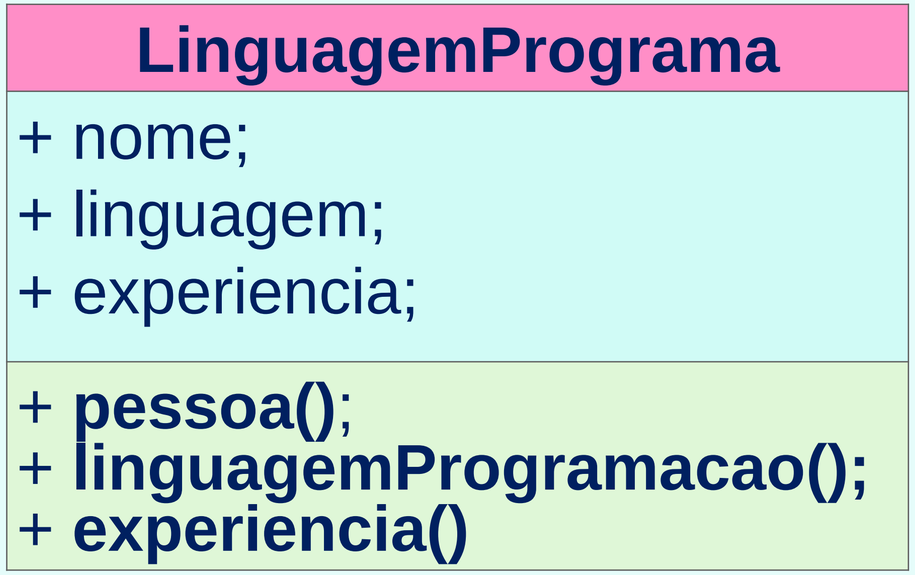
Quadrado
- lado
+ **getNomeFigura()**; + **getArea()**; + **getPerimetro()**;

Triangulo
- base; - altura; - ladoA; - ladoB; - ladoC;
+ **getNomeFigura()**; + **getArea()**; + **getPerimetro()**;

Atividade 3

Crie um programa que contenha três classes conforme apresentado no diagrama ao lado. O programa deve apresentar duas pessoas contendo nome, a linguagem de programação que trabalha e quantos anos de experiência. Faça um contrato através da interface e depois implemente os métodos. Instancia o objeto Maria que trabalha a 15 anos com Python e João que trabalha a 10 anos com Java.





vnt/school
powered by venturus

Atividade 4

Nos slides a seguir serão apresentados os tratamentos de exceções, que é dado pela estrutura **try-catch**. Como atividade para o seu aprendizado, leia o conteúdo com atenção e **reproduza os códigos apresentados**.

vnt/school
powered by venturus

Estrutura Try-Cacth: Tratamento Exceções

Bloco try:
Contém o código que representa a execução normal do trecho de código que pode acarretar em uma exceção.

Bloco catch:
Contém o código a ser executado caso uma exceção ocorra.

Deve ser especificado o tipo da exceção a ser tratada (**upcasting é permitido**).

Estrutura Try-Cacth: Tratamento Exceções

Sintaxe:

```
try {

} catch (ExceptionType e) {
        <comandos>
 } catch (ExceptionType e) {
        <comandos>
 } catch (ExceptionType e) {
        <comandos>
}
```

vnt/school
powered by venturus

Estrutura Try-Cacth: Tratamento Exceções

Em **destaque**, a primeira linha cria uma **vetor do tipo String**. Nomes devem ser inseridos com espaços.

Em seguida, a **variável position** captura o conteúdo da posição digitada.

A posição tem a sua impressão na linha seguinte.

```
public class Exemplo1 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

            String[] vect = sc.nextLine().split(" ");
            int position = sc.nextInt();
            System.out.println(vect[position]);

        System.out.println("Fim odo programa");
        sc.close();
    }
```

Esse código pode levar a algumas exceções.

vnt/school
powered by venturus

Estrutura Try-Cacth: Tratamento Exceções

Sem Exceção:
Digite 3 Strings seguidas de espaços: **Maria Jose Joao**
Digite Position: **1**
Resultado: O nome Jose deve ser impresso.
Exceção 1:
Digite 3 strings seguidas de espaços: **Maria Jose Joao**
Digite Position: **e (**digite uma letra**)**
Resultado: InputMismatchException. (erro de entrada)
Exceção 2:
Digite 3 strings seguidas de espaços: **Maria Jose Joao**
Digite Position: **5**
Resultado: ArrayIndexOutOfBoundsException. (erro de indice inexistente)

vnt/school
powered by venturus

Estrutura Try-Cacth: Tratamento Exceções

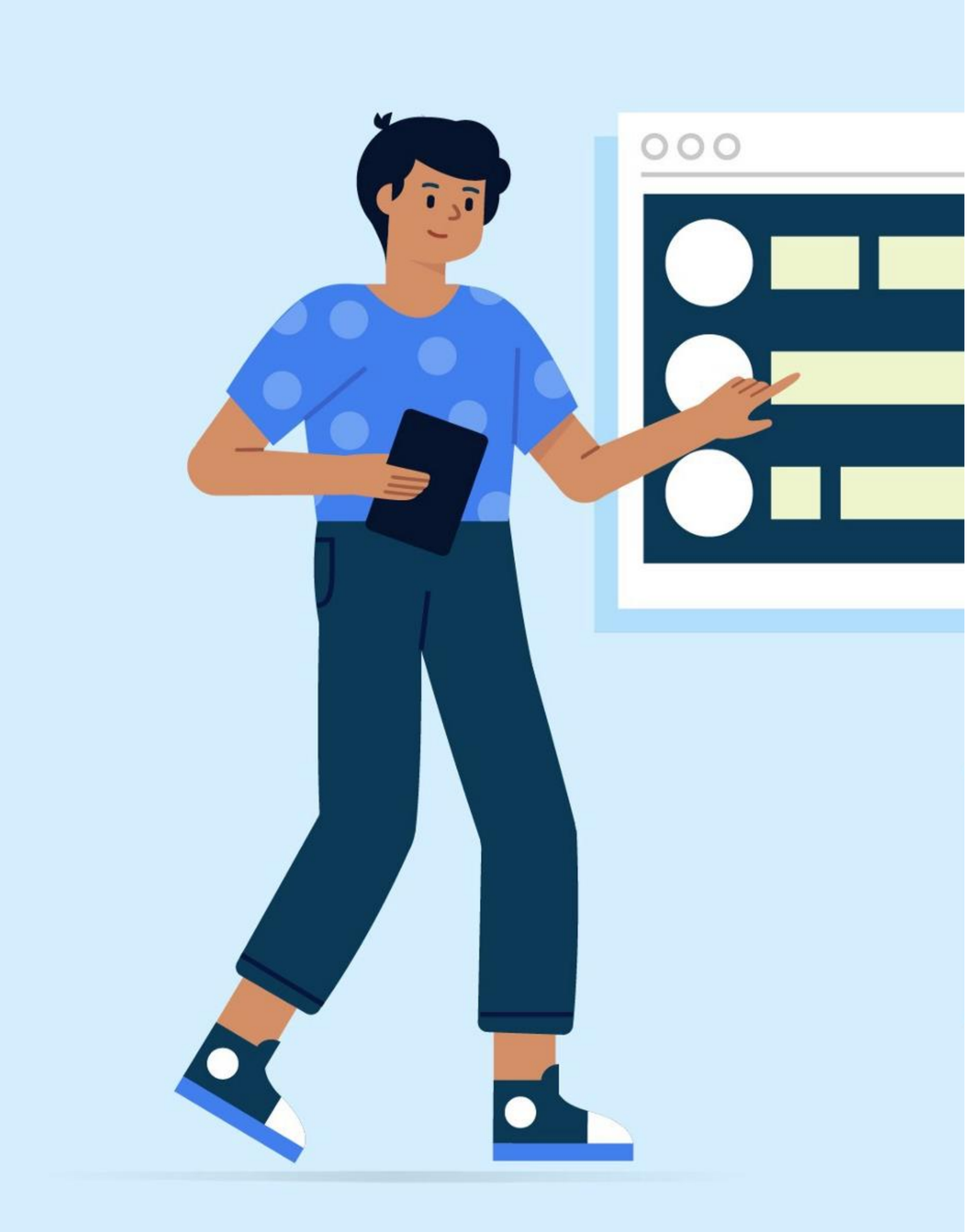
Exemplo:
A solução é tratar as exceções conforme o código ao lado.

As exceções agora irão enviar uma mensagem mais explicativa, caso elas ocorram.

Refaça o exercício anterior com esse código.

```java
public class Exemplo1 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        try {
            String[] vect = sc.nextLine().split(" ");
            int position = sc.nextInt();
            System.out.println(vect[position]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Invalid position!");
        } catch (InputMismatchException e) {
            System.out.println("Input error");
        }
        System.out.println("End of program");
        sc.close();
    }
```

vnt/school
powered by venturus



Comunidade VNT

Dica de hoje

Toda classe em Java é uma subclasse da classe **Object.**
Object possui os seguintes **métodos**:

getClass - retorna o tipo do objeto
equals - compara se o objeto é igual a outro
hashCode - retorna um código hash do objeto
toString - converte o objeto para string

Pesquisa um pouco mais sobre!!
Boa pesquisa!!

Referências

[1] A. Goldman, F. Kon, Paulo J. S. Silva; Introdução à Ciência da Computação com Java e Orientação a Objetos (USP). 2006. Ed. USP.
[2] Algoritmo e lógica de programação. Acessado julho/2022: https://visualg3.com.br/
[3] G. Silveira; Algoritmos em Java; Ed. Casa do Código.
[4] M. T. Goodrich, R. Tamassia; Estrutura de dados e algoritmos em Java. Ed Bookman. 2007.
[5] Algoritmo e lógica de programação. Acessado julho/2022: https://www.cursoemvideo.com/
[6] P. Silveira, R. Turini; Java 8 Pratico: lambdas, streams e os novos recursos da linguagem. Ed. Casa do Código.
[7] Linguagem Java: Curso acessado em agosto/2022: https://www.udemy.com/
[8] Linguagem Java: Curso acessado em setembro/2022: https://www.cursoemvideo.com/