PROGRAMAÇÃO DESKTOP

Por Huilson J. Lorenzi
Aula Introdutória - 2025

CARREIRA ACADÊMICA/PROFISSIONAL

2010 / 2015- Graduação em Ciência da Computação - UNOCHAPECÓ.

2012 / 2022 - Professor de Ensino Fundamental e Médio - São Lourenço do Oeste.

2013 - Estágio no Centro de Residência de Software - UNOCHAPECÓ.

2018 - Pós-Graduação em Desenvolvimento de Jogos Digitais pela UNYLEYA - Curitiba.

2023 - Pós-Graduação em Programação para Dispositivos Móveis pela UTFPR - Pato Branco

2022 / 2024 - Professor Substituto pela UTFPR de Pato Branco.

2024 - Programador Web pela Supera Sistemas de Pato Branco.

PLANO DE ENSINO

Objetivo

Aprender os passos necessários para buscar soluções visando, não só a informatização de processos, mas também a readequação dos mesmos, sempre visando a maximização de resultados e a redução dos recursos necessários, tornando assim esses processos sustentáveis do ponto de vista econômico, social e em relação ao meio-ambiente.

PLANO DE ENSINO

Ementa

Histórico e cenário atual da POO;
Programação estruturada e POO;
Paradigma de programação orientada a objetos;
Classes, Objetos;
Polimorfismo;
Sobrecarga de Métodos;
Herança; Encapsulamento;
Interface gráfica;
Persistência de dados.

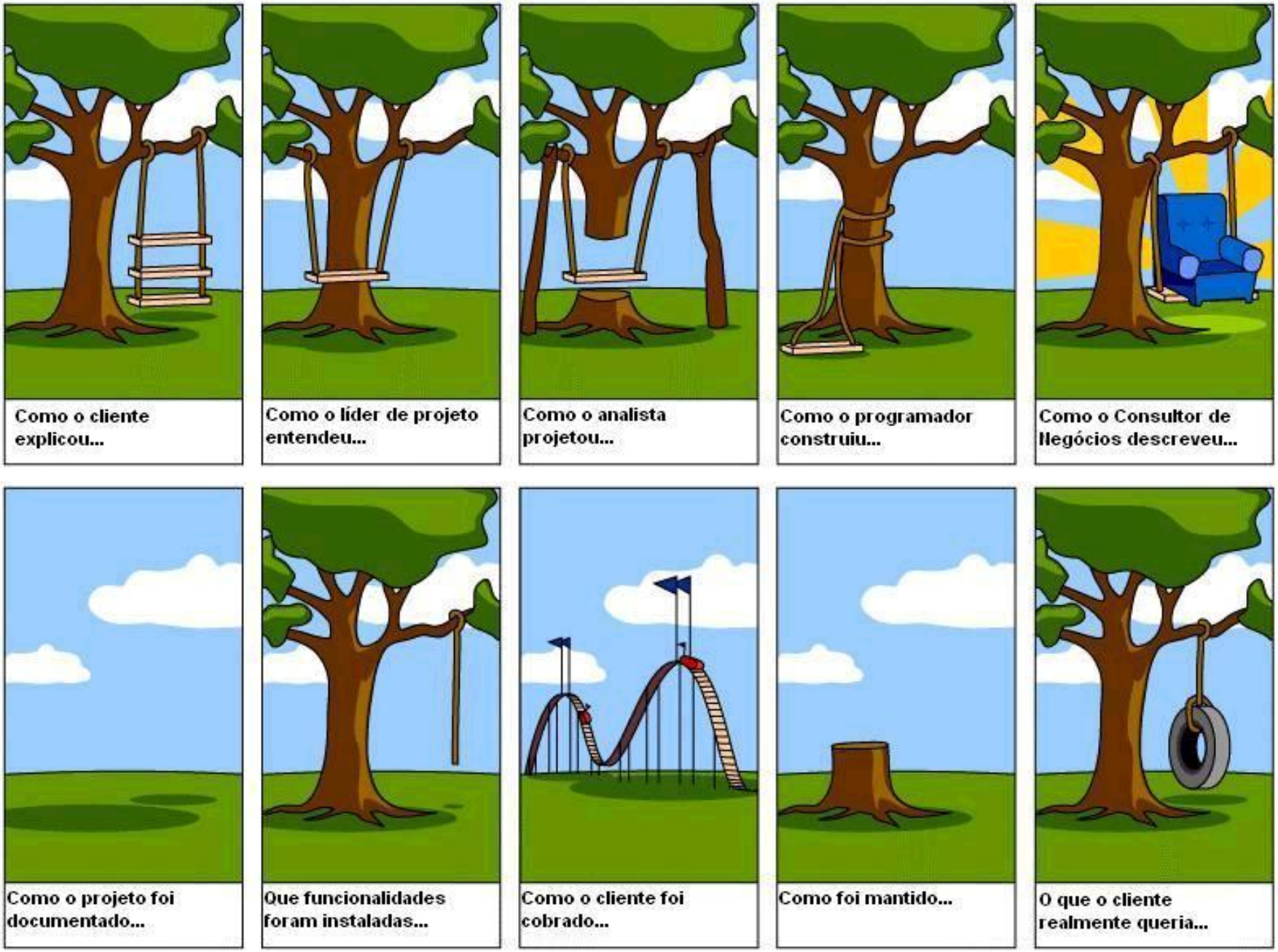
PLANO DE ENSINO

Conteúdo

- Conhecendo o Kotlin (ou Java)
- Pilares da Orientação a Objetos (Encapsulamento, Herança, Polimorfismo)
- Acesso a base de dados utilizando mapeamento objeto-relacional.
- Desenvolvimento de aplicações em camadas (MVC).
- Desenvolvimento de relatórios e gráficos utilizando frameworks.
- Testes unitários.

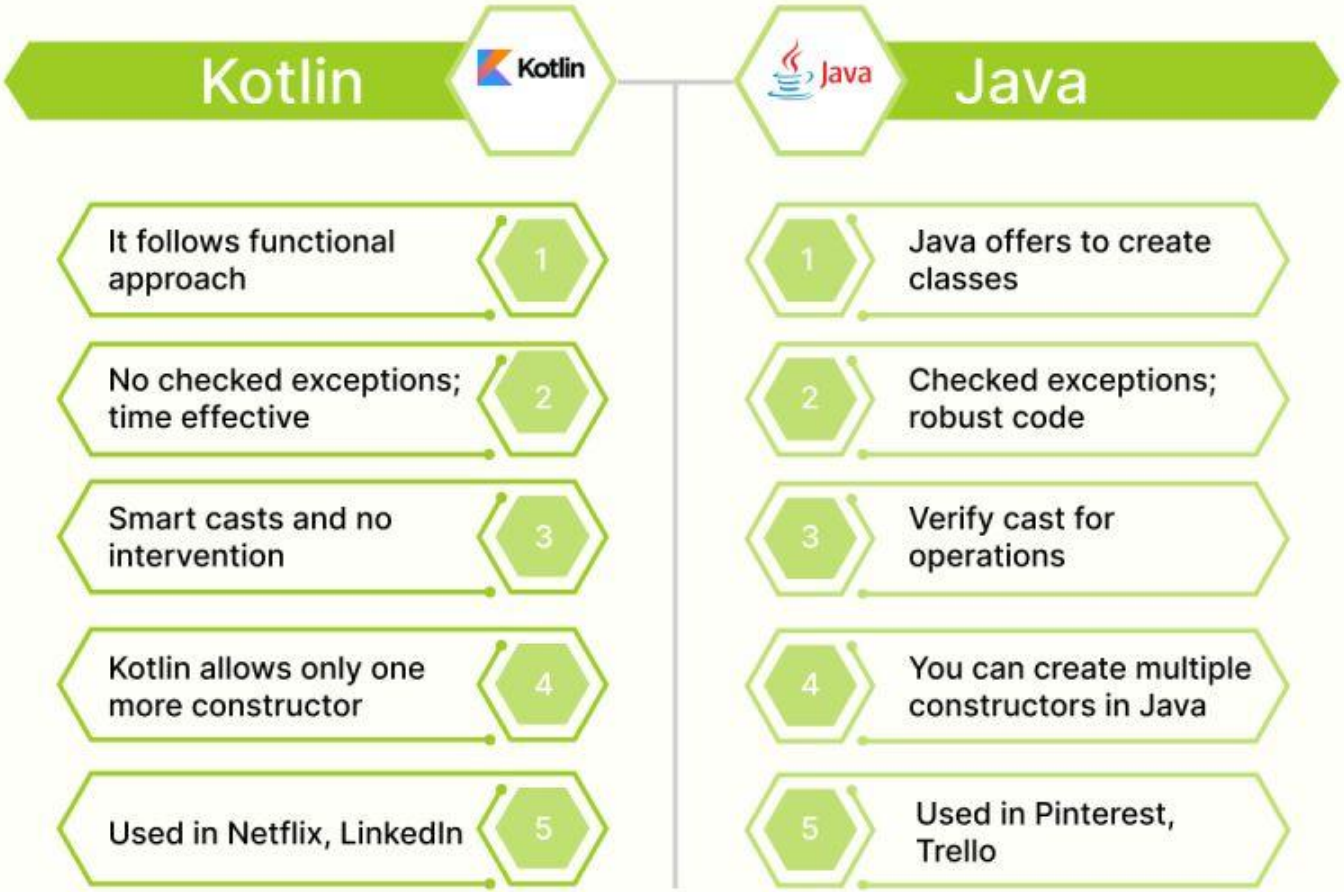
AVALIAÇÕES

- EX: Exercícios serão aplicados de forma de pesquisa, aplicações práticas e teóricas. (Máximo de 50% da nota final)
- TF: Uma prova teórica e um trabalho. (Mínimo de 50% da nota final)
- SM: Um Seminário apresentado no final da disciplina, com o intuito de fixar o conhecimento adquirido em sala.


Como o cliente explicou...
Como o líder de projeto entendeu...
Como o analista projetou...
Como o programador construiu...
Como o Consultor de Negócios descreveu...
Como o projeto foi documentado...
Que funcionalidades foram instaladas...
Como o cliente foi cobrado...
Como foi mantido...
O que o cliente realmente queria...

ANDROID DEVELOPER

Kotlin vs Java 2023



SINTAXE

- Não é necessário especificar certos diretórios e pacotes: os arquivos de origem podem ser colocados arbitrariamente no sistema de arquivos.
- Toda classe ou interface começa com letra maiúscula e todo atributo, método, propriedade, variável com letra minúscula.
- Um ponto de entrada de uma aplicação Kotlin é a função **main**.
- **NOTA:** JAVA é *case-sensitive*: “MinhaClasse" e "minhaclasse" tem diferentes significados, essa técnica é chamada CamelCase.

IMPRIMIR NO CONSOLE

- Em Kotlin é possível usar os métodos *println/print* para imprimir textos no console:
  - *println(“Esse é um exemplo com println que pula de linha”);*
  - *print(“O print não pula para uma nova linha”);*
  - *println(“Uma variável fora do escopo”, + var);*
  - *println(“Uma variável dentro do escopo $var”);*

TIPOS DE VARIÁVEIS

- O Kotlin usa duas palavras-chave diferentes para declarar variáveis: **val** e **var**.
  - **VAL:** para uma variável cujo valor nunca muda. Não é possível reatribuir um valor a uma variável que tenha sido declarada usando val.
  - **VAR:** para uma variável cujo valor possa ser mudado.

TIPOS DE VARIÁVEIS

- O Kotlin pode inferir o tipo com base no tipo do valor atribuído.
  - **INTEIROS:** Byte, Short, Int, Long.
  - **BOOLEAN:** true, false.
  - **CARACTERES:** Char.
  - **PONTO FLUTUANTE:** Float, Double.
  - **STRINGS:** String

CONVERSÃO DE VARIÁVEL

- A **conversão** em Kotlin é muito simples, bastando adicionar a função encadeada para o tipo de variável pretendido. *Exemplo: .toInt(), .toString(), etc.*

  NOTA: É preciso ter um cuidado extra na conversão para não tomar uma exception.

CONVERSÃO DE VARIÁVEL (JAVA)

- **Widening Casting** (automático) – convertendo um tipo menor para um tipo maior.
  byte -> short -> char -> int -> long -> float -> double

- **Narrowing Casting** (manual) – convertendo um tipo maior para um tipo menor.
  double -> float -> long -> int -> char -> short -> byte

OPERADORES

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

CLASSE STRING

- Strings são usadas para armazenar pequenos textos
- Uma String recebe uma coleção de caracteres cercadas por aspas duplas.
- Exemplo: *val frase = “Exemplo de string!”;*

MÉTODOS EM STRING

- *.uppercase()* //para deixar a string em maiúsculo
- *.lowercase()* //para deixar a string em minúsculo
- *.indexOf()* //retorna a primeira ocorrência especificada
- *.length()* //retorna o tamanho da string

PARTICULARIDADES

- O sinal de **+** é usado tanto para concatenar (juntar duas strings) como para somar.

- **\”** Insere aspas duplas em uma string, e **\’** insere aspas simples em uma string, e **** insere uma contrabarra em uma string.

CLASSE MATH

- A classe Math é usada para realizar operações matemáticas em números.
  - *Math.max(x,y)* //retorna o maior valor das variáveis.
  - *Math.min(x,y)* //retorna o menor valor das variáveis.
  - *Math.sqrt(x)* //retorna a raiz quadrada da variável.

KOTLIN RANGE

- Assim como a classe Math do **JAVA**, a biblioteca **RANGE** do Kotlin possui diversos métodos para operações matemáticas.
  - *x.coerceAtLeast(y)* //retorna o maior valor das variáveis.
  - *x.coerceAtMost(y)* //retorna o menor valor das variáveis.
  - *sqrt(x)* //retorna a raiz quadrada da variável.

KOTLIN RANGE - INTERVALOS

- O Kotlin permite que você crie facilmente intervalos de valores usando as funções:
  - *5 in 1..5* //intervalo fechado, inclui 1 e o 5

    retorna true
  - *5 in 1..<5* // intervalo aberto, inclui 1 mas exclui o 5

    retorna false

KOTLIN RANGE - INTERVALOS

- Mais funções:
  - *for (i in 1..5)* // para usar em laços
  - *for (i in 5 downTo 1)* // iteração inversa
  - *for (i in 1..6 step 2)* // incremento customizado

LEITURA VIA CONSOLE

- Em **JAVA**, é possível usar o método *System.in* (*java.util.Scanner*) para fazer a leitura do teclado via console.

- Em **Kotlin** usamos o *readln()* para fazer a leitura do teclado via console, porém pode se usar o *readLine()* para ler algo nulo *(útil quando for usar em leitura de arquivos)*.

ARRAYS

- **Arrays** são usados para armazenar vários valores em um única variável, ao invés de declarar em variáveis separadas cada valor.
- Para declarar um **array**, use a palavra chave **arrayOf()**.
  Exemplo: *val nomes = arrayOf(“Huilson”, “Tais”, “Guilherme”, “Lucas”, “Eduarda”)*;
- Você pode acessar um elemento pelo seu índice.
  Exemplo: *println(nomes[0])*;
- Para retornar o número de elementos de um **array** você pode usar *.size*

ARRAYS

- Você pode criar um loop através de um **array** usando **for** junto com o uso do **.size**

```
val cars = arrayOf("Volvo", "BMW", "Ford", "Mazda")
for (i = 0; i < cars.size; i++) {
  println(cars[i]);
}
```

- Você também pode usar um **for-each**

```
for (car in cars) {
  System.out.println(cars[i]);
}
```

EXEMPLO WHEN KOTLIN

```
val day = 4
    when (day) {
        1 -> println("Monday")
        2 -> println("Tuesday")
        3 -> println("Wednesday")
        4 -> println("Thursday")
        5 -> println("Friday")
        6 -> println("Saturday")
        7 -> println("Sunday")
    }
```

OPERADOR TERNÁRIO

- Um operador ternário faz parte da sintaxe para uma expressão condicional básica. (wikipédia)

- Operador ternário em **if/else**:
  - variável = (condição) **?** expressãoVerdadeira **:** expressãoFalsa;

- Exemplo em JAVA:
  - *int time = 20;*
  - *String result = (time < 18) ? "Good day." : "Good evening.";*
  - *System.out.println(result);*

VALORES NULOS EM KOTLIN

- Para permitir valores nulos, declare uma variável com um sinal de interrogação (*?*) logo após o tipo da variável. Por exemplo, você pode declarar uma string anulável escrevendo String?. Esta expressão cria uma String de um tipo que pode aceitar valores nulos.

ELVIS OPERATOR

- Ao trabalhar com tipos anuláveis (nulos), você pode verificar *null* e fornecer um valor alternativo. Por exemplo, se *b* não for *null*, acesse b.length. Caso contrário, retorne um valor alternativo:

```
val b: String? = null
val l = b?.length ?: 0
println(l)
```

SAFE CALL OPERATOR

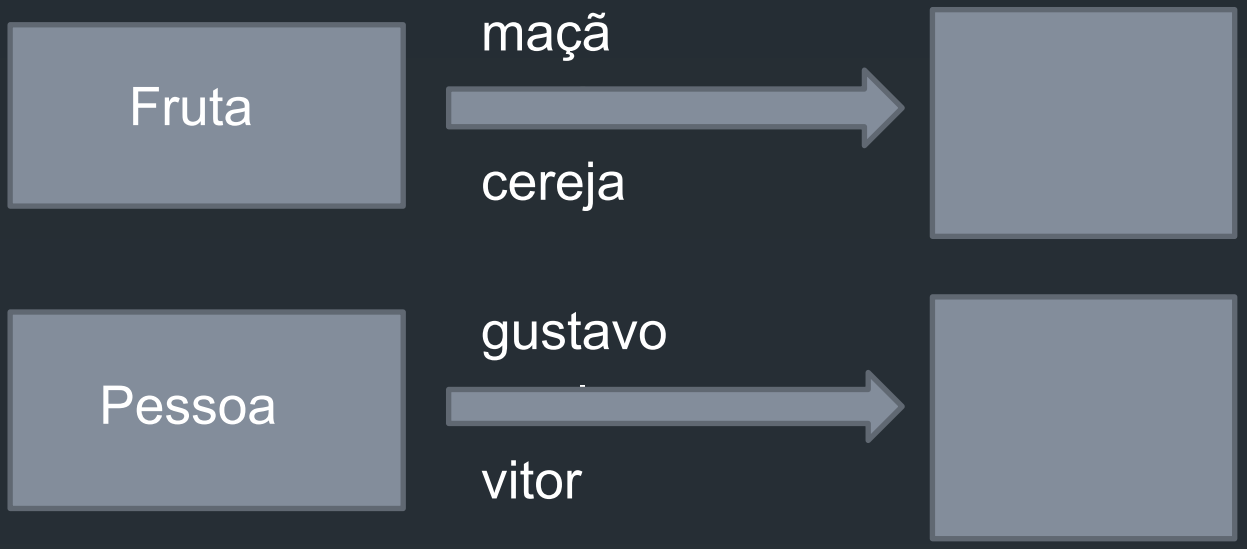
- O operador *?.* permite que você manipule a nulidade com segurança em um formato mais curto. Em vez de lançar um NPE, se o objeto for null, o ?.operador simplesmente retorna *null*.

```
val a: String? = "Kotlin"
val b: String? = null

println(a?.length)
println(b?.length)
```

CLASSES E OBJETOS

- **CLASSE** **OBJETO**



CLASSE E OBJETO

Uma classe é um modelo que define um tipo de objeto, enquanto um objeto é uma instância desse tipo.

- **Classe**
  - É um modelo ou especificação que define as propriedades e métodos dos objetos
  - É uma forma de definir um tipo de dado
  - É formada por dados e comportamentos
  - Os dados são definidos por atributos
  - Os comportamentos são definidos por métodos

CLASSE E OBJETO

- **Objeto**
  - É uma instância personalizada da classe
  - É algo concreto e físico
  - Existe na memória durante a execução da aplicação
  - Possui características próprias, denotadas por atributos
  - Pode ser categorizado, agrupado

ENCAPSULAMENTO

- O significado de Encapsulamento, é ter a certeza que a informação “sensível” esteja escondido do usuário. Para alcançar isso, deve-se:
  - Declarar as variáveis/atributos da classe como privado.
  - Gerar métodos **getters** e **setters** para acessar e atualizar os valores das variáveis privadas.

  - O método **get** retorna o valor da variável.
  - O método **set** pega um parâmetro e envia a variável.

HERANÇA

- Em JAVA, é possível herdar atributos e métodos de uma classe para outra. O conceito de herança é dividido em dois grupos:
  - Superclasse (Classe mãe/raiz): A classe do qual é herdada.
  - Subclasse (Classe filha/folha): A classe que herda de outra classe.

- Para herdar algo de uma classe é usada a palavra chave, **extends**.

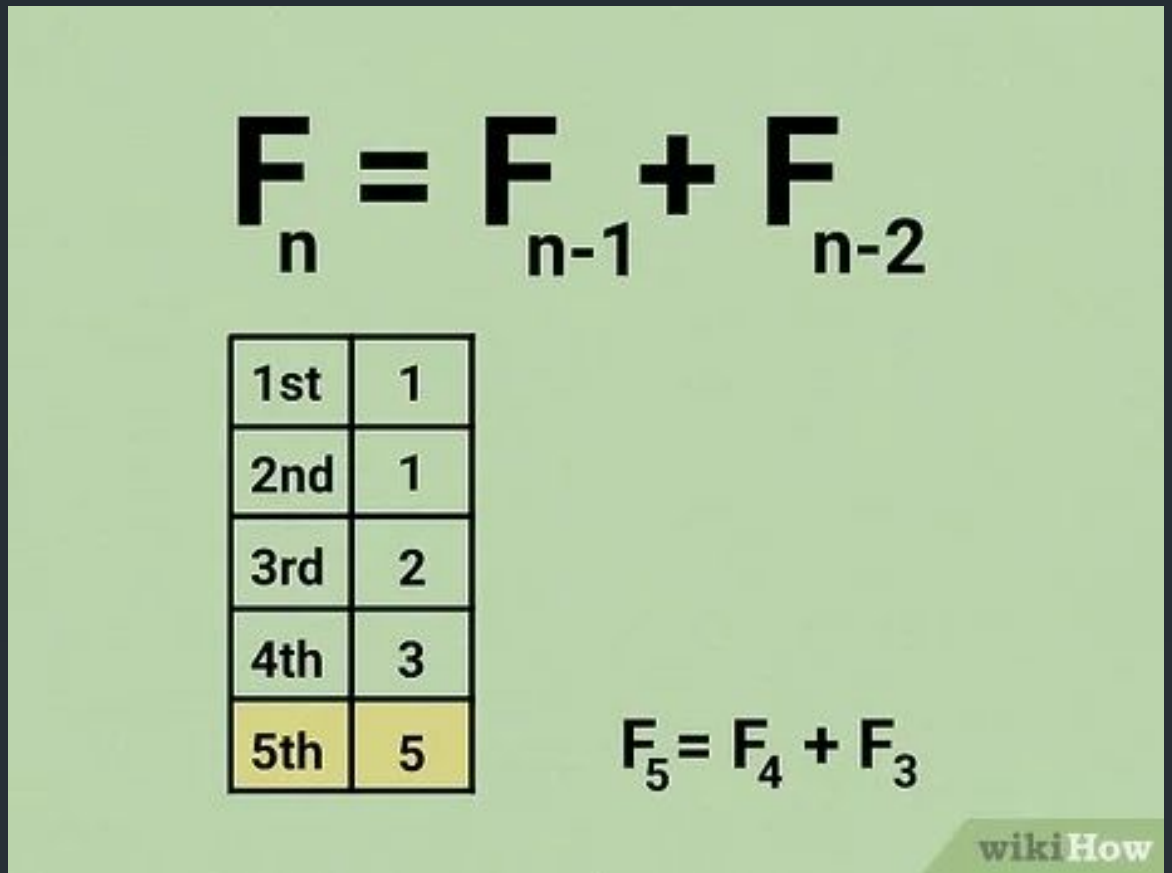
POLIMORFISMO

- O polimorfismo significa “muitas formas”, e ocorre quando temos muitas classes que estão relacionadas uma à outra por herança.

- Uma superclasse chamada **Animal** possui um método chamado *somAnimal()*. As Subclasses chamadas **Porco**, **Gato** e **Cachorro**, possuem o mesmo método, mas cada uma possui características diferentes.

EXERCÍCIO DE FIXAÇÃO

- Crie uma aplicação que resolva a sequência de Fibonacci baseado em um número de casas especificado (mínimo 3) pelo usuário.

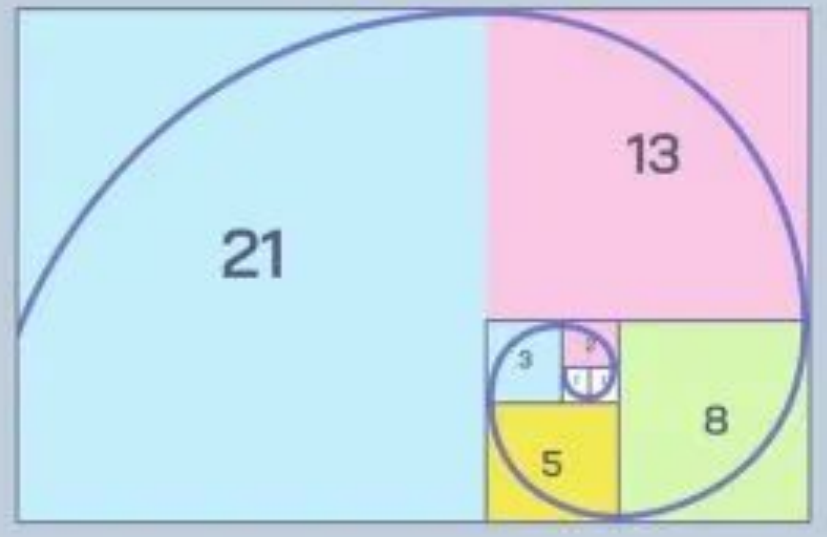


A sequência de Fibonacci

Cada número é a soma dos seus dois antecessores

1 1 2 3 5 8 13 21

1 + 1 = 2
1 + 2 = 3
2 + 3 = 5
3 + 5 = 8
5 + 8 = 13
8 + 13 = 21



REFERÊNCIAS

- W3schools, **Java Tutorial**. Link: https://www.w3schools.com/, 2025.
- Rodrigues, Érick Oliveira. **ORIENTAÇÃO A OBJETOS** - *Revisão e Particularidades Java, 2021.*
- Kotlin. **Documentação**. *Link: https://kotlinlang.org/docs/, 2025*
- Developers. **Guides**. *Link: https://developer.android.com/kotlin/, 2025*
- Guanabara, Gustavo. **Curso de Java**. Link: https://www.youtube.com/watch?v=sTX0UEpIF54&list=PLHz_AreHm4dkI2ZdjTwZA4mPMxWTfNSpR, 2015.