

# Programação Orientada a Objetos

Prof. Alexandre Krohn





# Roteiro

- Programação Orientada a Objetos
- Princípios Básicos
- Introdução a Linguagem Java
- Exercícios

# Programação Orientada a Objetos

**Programação Orientada a Eventos**

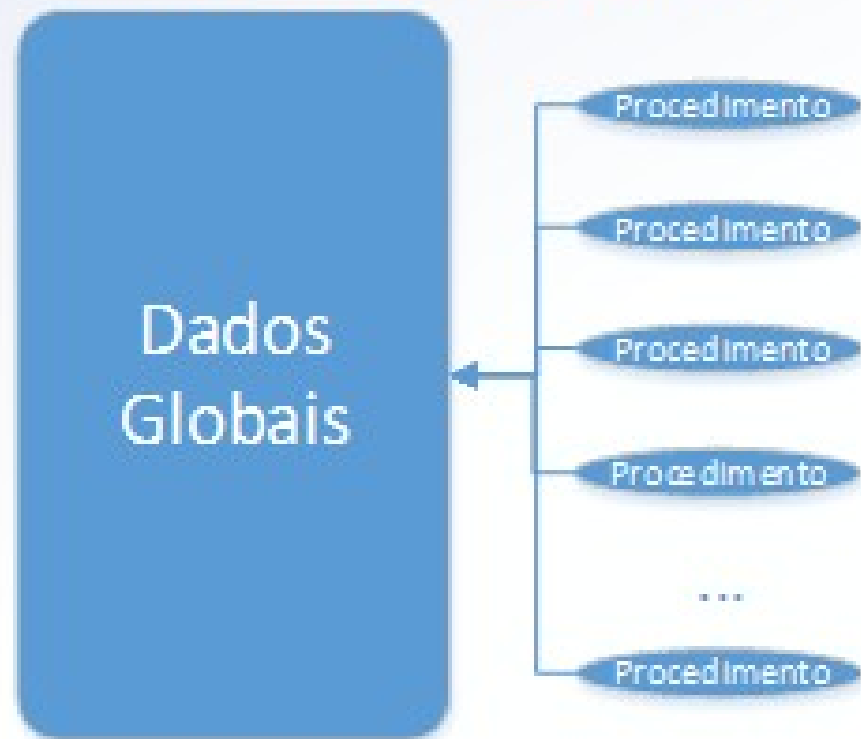
**Programação Orientada a Objetos**

**Programação Estruturada**

**Programação Procedural**

# Comparando...

## Programação Estruturada



## Programação Orientada a Objetos



# Um exemplo em C

```
#include <stdio.h>
#include <stdlib.h>
```

```
int n, i, divisores;
```



Variáveis Globais !!!

```
int main()
{
```

```
    printf("Informe um número:\n");
    scanf("%d",&n);
```

```
    divisores = 0;
    for(i=1;i<=n;i++) {
        if(n%i==0) {
            divisores++;
        }
    }
```

```
    if(divisores==2) {
        printf("O número %d é primo",n);
    } else {
        printf("O número %d não é primo",n);
    }
```

```
    return 0;
```

```
}
```

# Um exemplo em Java

```
public class Primos {
```

```
    public boolean ehPrimo(int n) {
```

```
        int divisores, i;
```

```
        divisores = 0;
```

```
        for(i=1; i<=n; i++) {
```

```
            if(n%i==0) {
```

```
                divisores++;
```

```
            }
```

```
        }
```

```
        return divisores==2;
```

```
    }
```

```
    public static void main(String[] args) {
```

```
        int numero;
```

```
        Primos p = new Primos();
```

```
        Scanner in = new Scanner(System.in);
```

```
        System.out.print("Informe um número: ");
```

```
        numero = in.nextInt();
```

```
        if(p.ehPrimo(numero)) {
```

```
            System.out.println(numero + " é primo!");
```

```
        } else {
```

```
            System.out.println(numero + " NÃO é primo!");
```

```
        }
```

```
        in.close();
```

```
    }
```

```
}
```

Variáveis

Aqui as variáveis  
pertencem às  
funções

# Outro exemplo em C : Estruturas

```
typedef struct Elemento
{
    char nome [100];
    char rua [100];
    char cidade [100];
    char estado [2];
    char cep [10];
} Elemento;
```

Estruturas são uma maneira de tratar dados de forma agrupada.

Objetos também, mas com vantagens

```
Elemento el;

printf ( "%s \n", el.nome);
printf ( "%s \n", el.rua);
printf ( "%s \n", el.cidade);
printf ( "%s \n", el.estado);
printf ( "%s \n", el.cep);
```



# Objetivos da OO

- Diminuir o **tempo e o custo** de desenvolvimento através da **reutilização de componentes** de software na forma de classes, empregando **solução incremental** de problemas e usando subclasses.
- Diminuir o custo da **manutenção** de software através da habilidade de **localizar alterações** para implementação de uma ou mais classes





# Orientação a Objetos

Para uma linguagem ser **Orientada a Objetos** ela **precisa**:

- Ser **baseada em objetos**, ou seja, deve ser fácil programar objetos que **encapsulam dados e operações**;
- Ser baseadas em classes, ou seja, **cada objeto** pertence a (ou **é fabricado a partir de**) **uma classe**; e
- Permitir **herança**, ou seja, deve ser fácil agrupar classes em **hierarquias** de subclasses e superclasses

Peter Wegner, OOPSLA, 1987



# Vantagens

- Os métodos estruturados aplicam **funções e procedimentos ativos** a **dados passivos**. Os métodos orientados a objetos encapsulam dados e procedimentos.
  - Representação de **cada elemento como objeto**
  - **Reutilização de código**
  - **Independência** entre partes do código
  - Facilidade de leitura e **manutenção**
  - Possibilidade de **criação de bibliotecas** de classes
  - Melhor **organização** da aplicação : **Padronização**
  - **Segurança** do software



# Orientação a Objetos

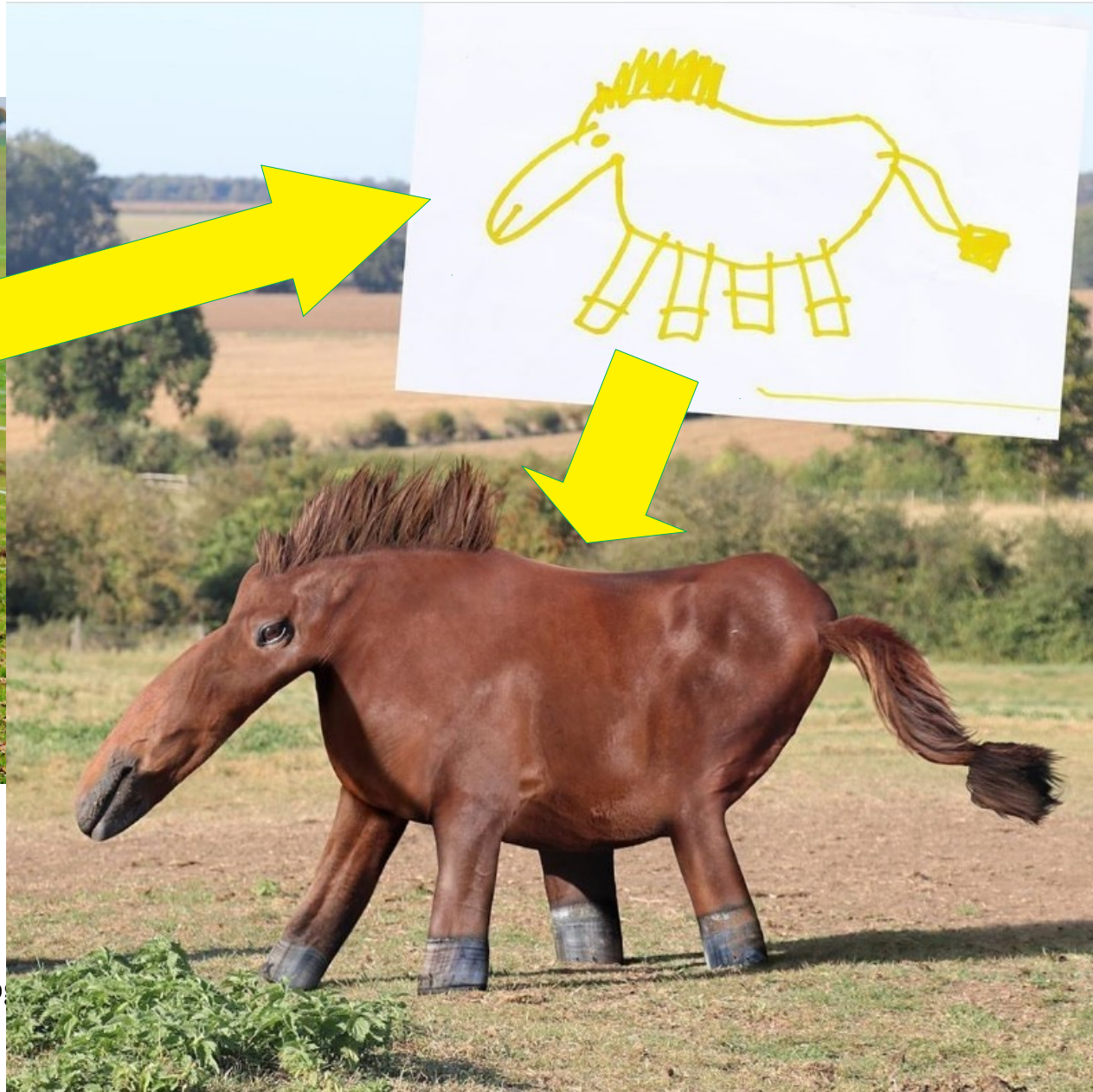
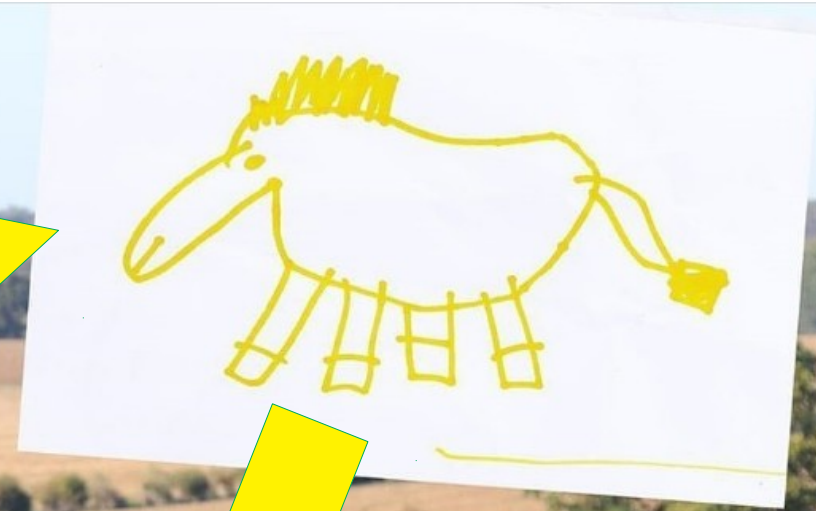
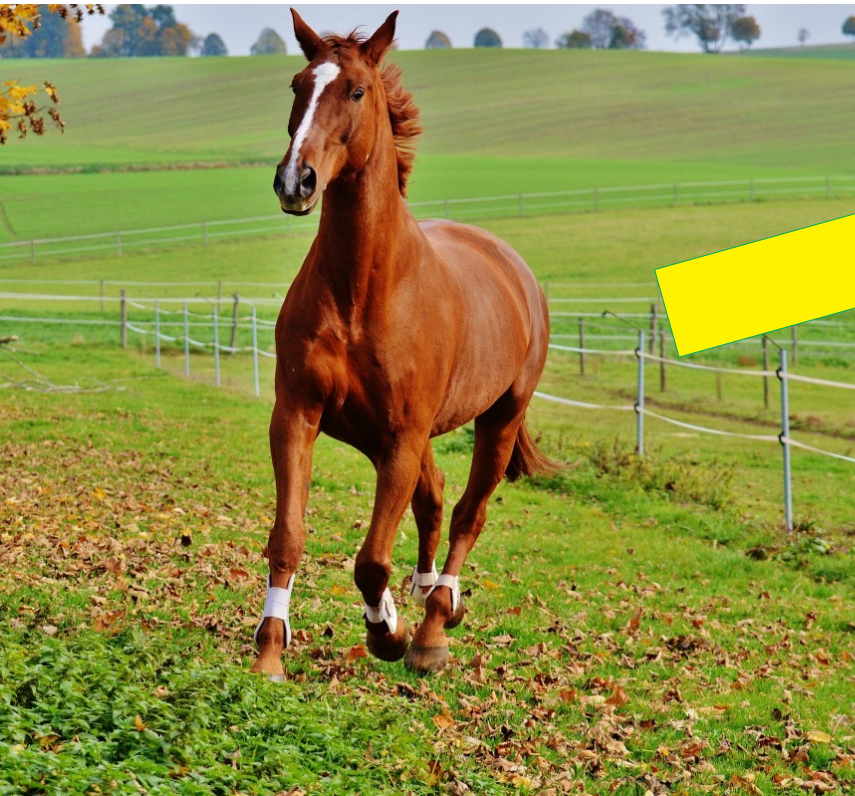
- Quatro pilares da OO
  - **Abstração** : Identidade, propriedades e métodos
  - **Encapsulamento** : Caixa Preta
  - **Herança** : Menos código-fonte
  - **Polimorfismo** : Mesma chamada, objeto diferente, comportamento diferente + Mesmo objeto, diferentes chamadas, diferentes comportamentos



# Abstração

- Forma de retirar as características **relevantes** de algo do mundo real para incluir em um projeto
- Classes são abstrações
- **Abstração** é a habilidade de concentrar nos aspectos **essenciais** de um **contexto** qualquer, ignorando características menos importantes ou acidentais. (Wikipedia)

# Abstração : armadilhas







# Encapsulamento

- Em Orientação a Objetos, diz-se que **dados e comportamentos** são “encapsulados” dentro dos objetos
- Dados são **atributos**
- Comportamentos são **métodos**
- Objetos são a composição de seu estado e seu comportamento



# Encapsulamento

- **Encapsulamento** é a técnica que faz com que **detalhes internos** do funcionamento dos métodos de uma classe permaneçam **ocultos** para os objetos.
- Por conta dessa técnica, **o conhecimento a respeito da implementação interna da classe é desnecessário do ponto de vista do objeto**, uma vez que isso passa a ser responsabilidade dos métodos internos da classe.



# Encapsulamento

- **Esconder** o como as coisas funcionam é a função do encapsulamento
- É separar o programa em partes, o mais **isoladas** possível.
- A ideia é tornar o software mais flexível, fácil de modificar e de criar novas implementações.

# Encapsulamento



# Encapsulamento







# Encapsulamento

- Encapsulamento **inibe acessos não autorizados** aos dados.

O encapsulamento **restringe os erros** a escopos menores

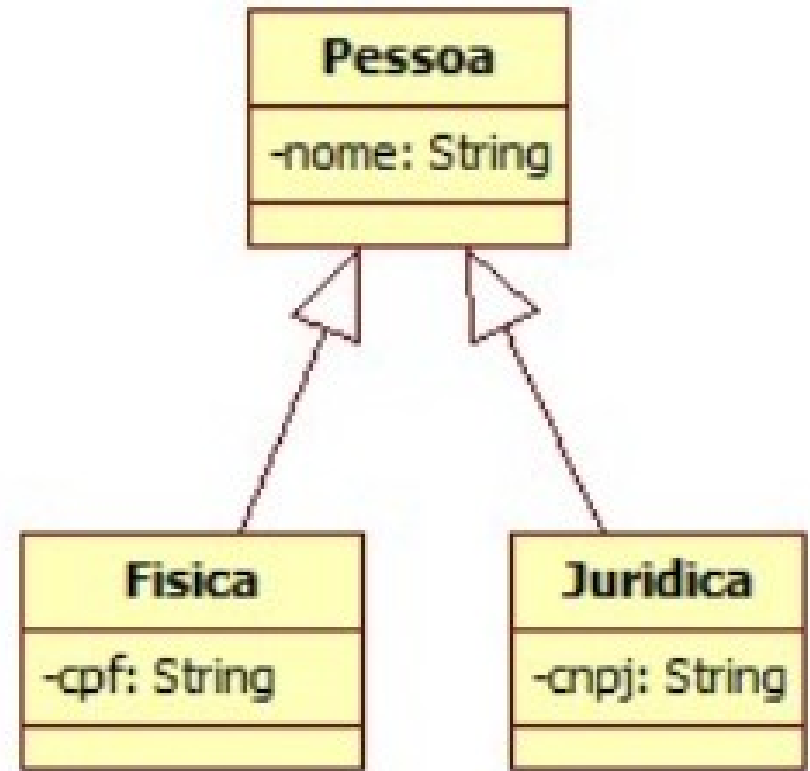


# Herança

- **Herança** é um princípio de orientação a objetos, que permite que classes compartilhem atributos e métodos, através de "heranças".
- Ela é usada na intenção de reaproveitar código ou comportamento generalizado ou especializar operações ou atributos.

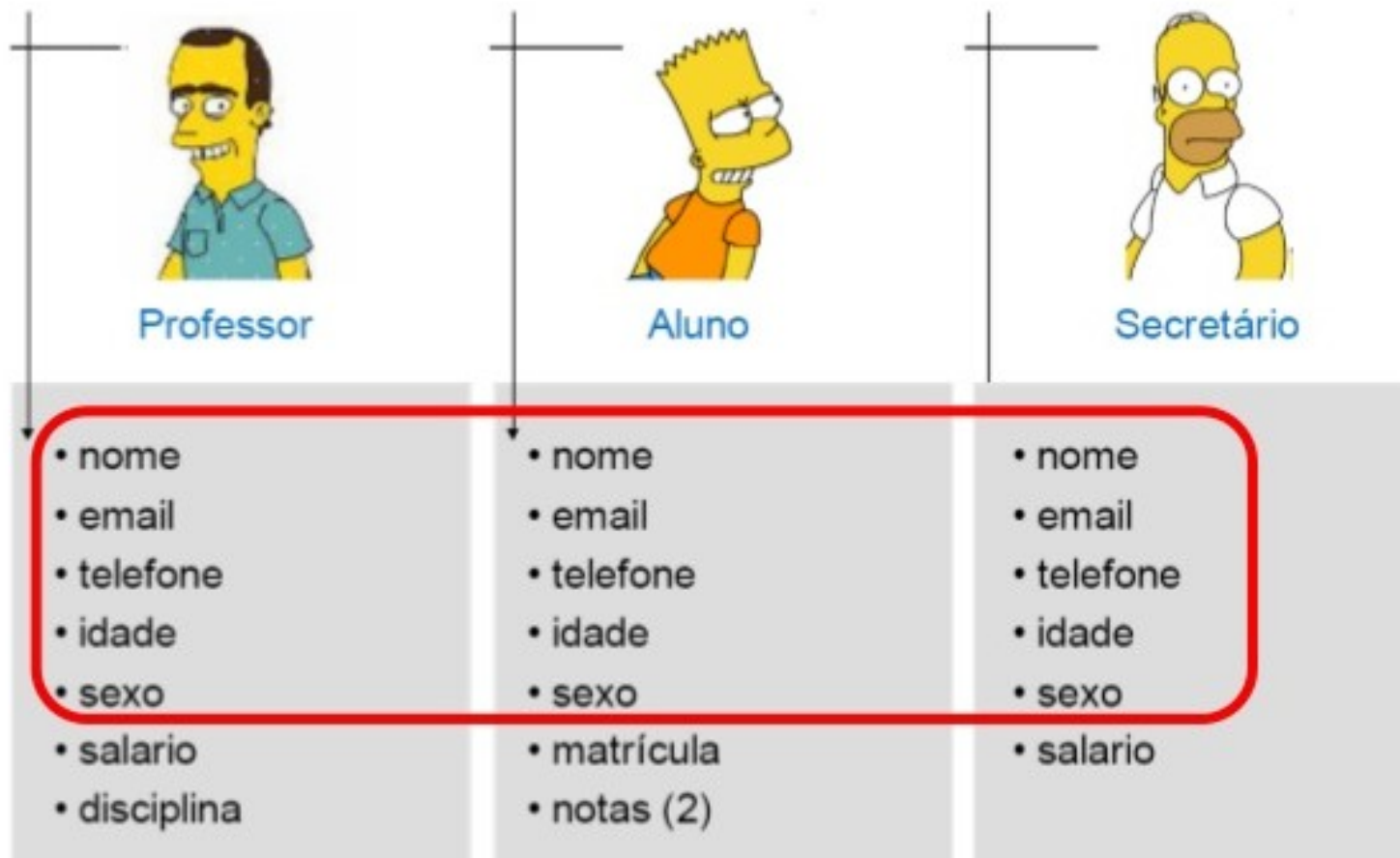
# Herança

- A ideia de **herança** é facilitar a programação. Uma **classe A** deve **herdar** de uma **classe B** quando podemos dizer que **A é um B**.



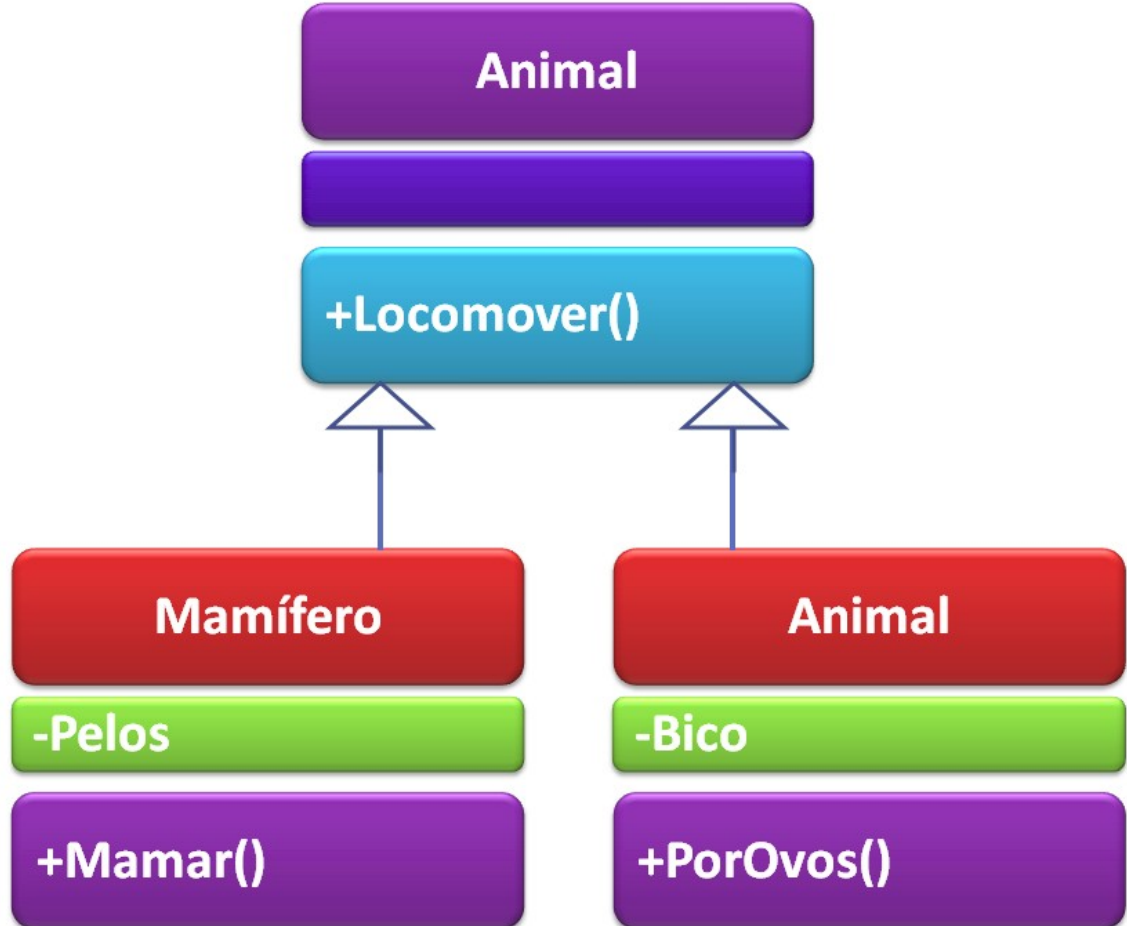
# Herança

- Dados (atributos) comuns



# Herança

- Comportamentos (métodos) comuns







# Herança

- Reduz a quantidade de código escrito
- Diminui o tamanho dos códigos-fonte
- Distribui a complexidade da aplicação



# Polimorfismo

- O Polimorfismo permite que um programador invoque um método **sem se preocupar** em como a ação foi implementada
- Isso é útil quando se quer que o mesmo método seja implementado de **forma diferente** em **classes diferentes**.



# Polimorfismo





# Polimorfismo

- Em um jogo como o **Fortnite**, todos os alvos implementam um método que é disparado quando são atingidos
- Cada alvo terá um comportamento diferente
- Mas o método “atingir()” será sempre o mesmo

# Classe ???





# Estrutura fundamental da O.O. : Classe

- Uma classe é um elemento do código O.O. que utilizamos para representar objetos do mundo real.
- Dentro dela é comum declararmos **atributos** e **métodos**, que representam, respectivamente, as **características** e **comportamentos** desse objeto.

# Classes x Objetos

- Classes são os “moldes”
- Objetos, ou instâncias, são os produtos gerados a partir desses moldes



Classes



Objetos





# Histórico : Linguagens

- **Simula**, criada em 1962 na Noruega por Ole-Johan Dahl e Kristen Nygaard foi a primeira linguagem OO
- **Smalltalk**, criada em 1970 por Alan Kay, no XEROX PARC, para, dentre outras coisas, as primeiras interfaces gráficas
- Em 1983, no Bell Labs, Bjarne Stroustrup estendeu a linguagem C com alguns conceitos de objetos e criou **C++**
- O **Objective-C** foi criado por Brad Cox e Tom Love em 1986 na empresa deles, a Stepstone
- Em 1991, Guido Van Rossum, do CWI (Centrum Wiskunde & Informatica, Centro de Matemática e Ciência da Computação) em Amsterdã, Holanda, lança a linguagem **Python**
- Em 1995, a Sun lança o ambiente **Java**, que inclui a linguagem Java, altamente influenciada por Smalltalk mas com sintaxe parecida com C++. Criador: James Gosling
- **Ruby** foi lançada em 1995 também, criada por Yukihiro Matsumoto, mas só atingiu aceitação em 2006
- Em 2000, Anders Hejlsberg, da Microsoft, cria o **C#**

# Linguagem de Programação Java

- Suporte à orientação a objetos;
- Portabilidade;
- Segurança;
- Linguagem Simples; (Derivada de C, mas sem ponteiros!)
- Alta Performance;
- Interpretada (o compilador pode executar os bytecodes do Java diretamente em qualquer máquina);
- Independente de plataforma;
- Tipada (detecta os tipos de variáveis quando declaradas);



# Linguagem de Programação Java



- Além disso:
- Linguagem utilizada amplamente
- Implementa todos os conceitos de O.O. necessários para a disciplina
- Boa empregabilidade (\$\$)
- Amplo conjunto de bibliotecas e ferramentas disponíveis
- Open Source e gratuita

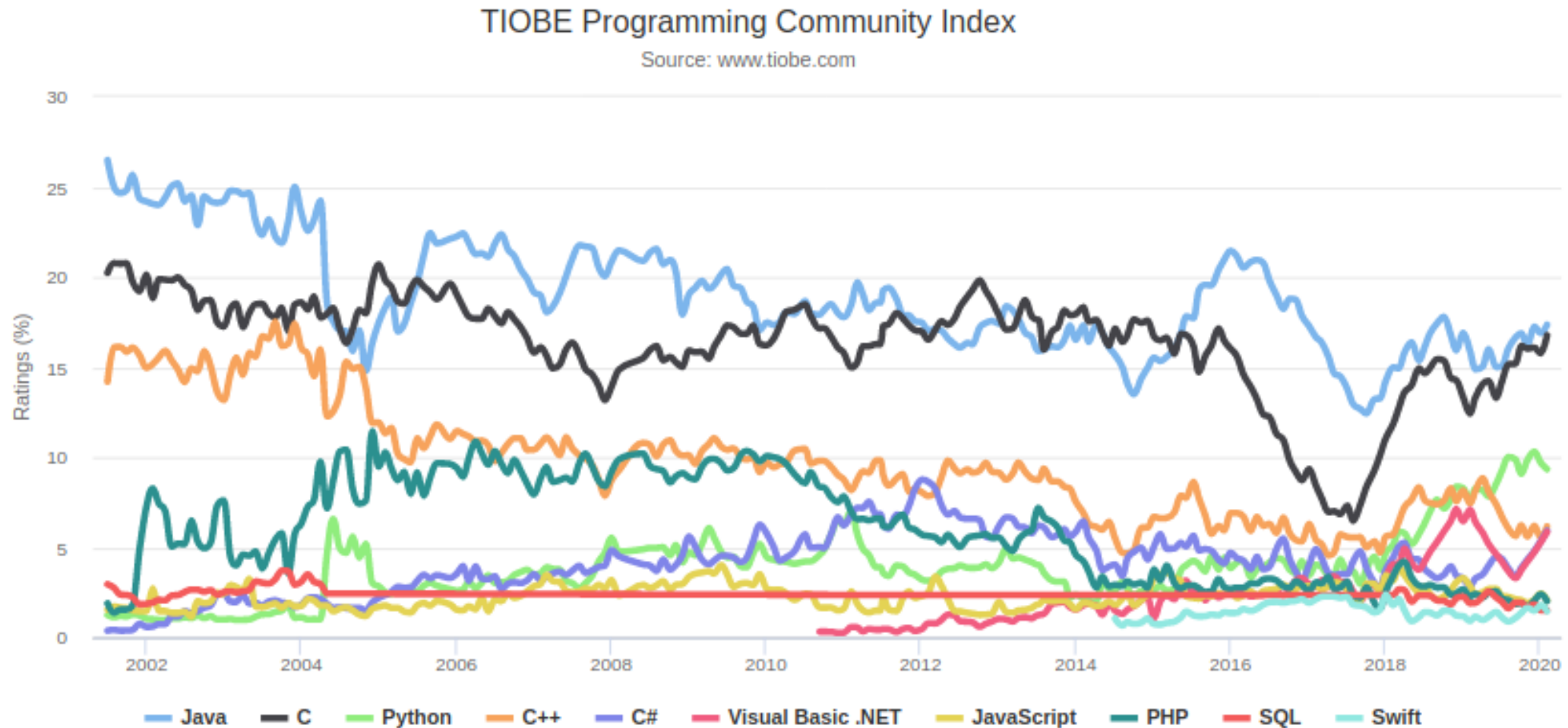


# Programação em Java

Feb 2020	Feb 2019	Change	Programming Language	Ratings	Change
1	1		Java	17.358%	+1.48%
2	2		C	16.766%	+4.34%
3	3		Python	9.345%	+1.77%
4	4		C++	6.164%	-1.28%
5	7	▲	C#	5.927%	+3.08%
6	5	▼	Visual Basic .NET	5.862%	-1.23%
7	6	▼	JavaScript	2.060%	-0.79%
8	8		PHP	2.018%	-0.25%
9	9		SQL	1.526%	-0.37%
10	20	▲▲	Swift	1.460%	+0.54%

Fonte : <https://www.tiobe.com/tiobe-index/>

# Programação em Java



Fonte : <https://www.tiobe.com/tiobe-index/>





# Java na UCS



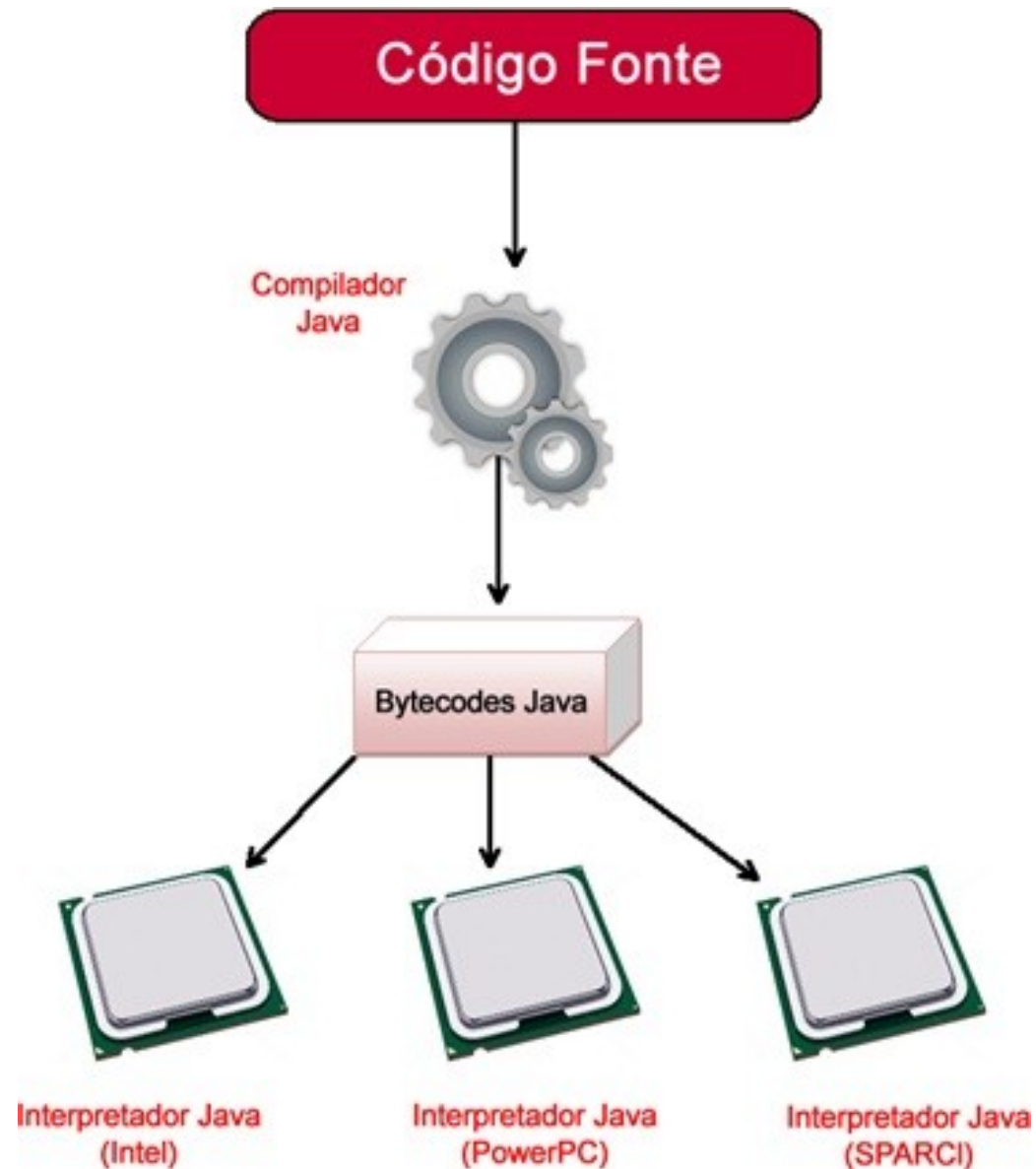
- Nos cursos de TI da UCS, Java aparece nas disciplinas de:
  - Programação OO
  - Projeto e Arquitetura de Software
  - Dispositivos Móveis (Android)
  - Programação Distribuída e Concorrente
  - Projetos Temáticos



# Java : Máquina Virtual

- Programas Java executam sobre uma máquina virtual, chamada JVM - Java Virtual Machine
- Uma máquina virtual é um software que funciona como se fosse um computador, executando programas. Para isso, ele intermedia chamadas ao sistema operacional real do computador hospedeiro

# Java: Compilado e Interpretado



# Estrutura de uma classe em Java

```
public class NomeDaClasse {  
  
    private int atributo1;  
    private double atributo2;  
    private String atributo3;  
    private boolean atributo4;  
    private char atributo5;  
  
    public NomeDaClasse() {  
        // implementação do método  
    }  
  
    public void metodo1(String a, int b) {  
        // implementação do método  
    }  
  
    public int metodo2() {  
        // implementação do método  
        return atributo1;  
        // exemplo de retorno  
    }  
  
    public String metodo3() {  
        // implementação do método  
        return atributo3;  
        // exemplo de retorno  
    }  
}
```

# Estrutura de uma classe em Java

```
public class NomeDaClasse {
```

Início da classe

```
    private int atributo1;  
    private double atributo2;  
    private String atributo3;  
    private boolean atributo4;  
    private char atributo5;
```

Definição da classe

```
    public NomeDaClasse() {  
        // implementação do método  
    }
```

```
    public void metodo1(String a, int b) {  
        // implementação do método  
    }
```

```
    public int metodo2() {  
        // implementação do método  
        return atributo1;  
        // exemplo de retorno  
    }
```

Fim da classe

```
    public String metodo3() {  
        // implementação do método  
        return atributo3;  
        // exemplo de retorno  
    }
```

```
}
```

# Estrutura de uma classe em Java

```
public class NomeDaClasse {  
  
    private int atributo1;  
    private double atributo2;  
    private String atributo3;  
    private boolean atributo4;  
    private char atributo5;  
  
    public NomeDaClasse() {  
        // implementação do método  
    }  
  
    public void metodo1(String a, int b) {  
        // implementação do método  
    }  
  
    public int metodo2() {  
        // implementação do método  
        return atributo1;  
        // exemplo de retorno  
    }  
  
    public String metodo3() {  
        // implementação do método  
        return atributo3;  
        // exemplo de retorno  
    }  
}
```

## Método construtor:

possui o mesmo nome da classe e não possui tipo de retorno.

É chamado na criação de um novo objeto.

Neste método, inicializam-se os atributos da classe.

# Estrutura de uma classe em Java

```
public class NomeDaClasse {  
  
    private int atributo1;  
    private double atributo2;  
    private String atributo3;  
    private boolean atributo4;  
    private char atributo5;  
  
    public NomeDaClasse() {  
        // implementação do método  
    }  
  
    public void metodo1(String a, int b) {  
        // implementação do método  
    }  
  
    public int metodo2() {  
        // implementação do método  
        return atributo1;  
        // exemplo de retorno  
    }  
  
    public String metodo3() {  
        // implementação do método  
        return atributo3;  
        // exemplo de retorno  
    }  
}
```

## Demais métodos:

métodos são módulos de uma classe que realizam determinada função.

Métodos têm basicamente a seguinte estrutura:

visibilidade retorno nome(parâmetros)  
{ //esta chave indica início do método  
  
} //esta chave indica final do método

Os parâmetros são declarados na forma: tipo nome (ex. String a).

Se existir mais de um parâmetro de entrada, estes são separados por vírgulas.

Mesmo sem parâmetros de entrada, os parêntesis são obrigatórios, indicando que se trata de um método.





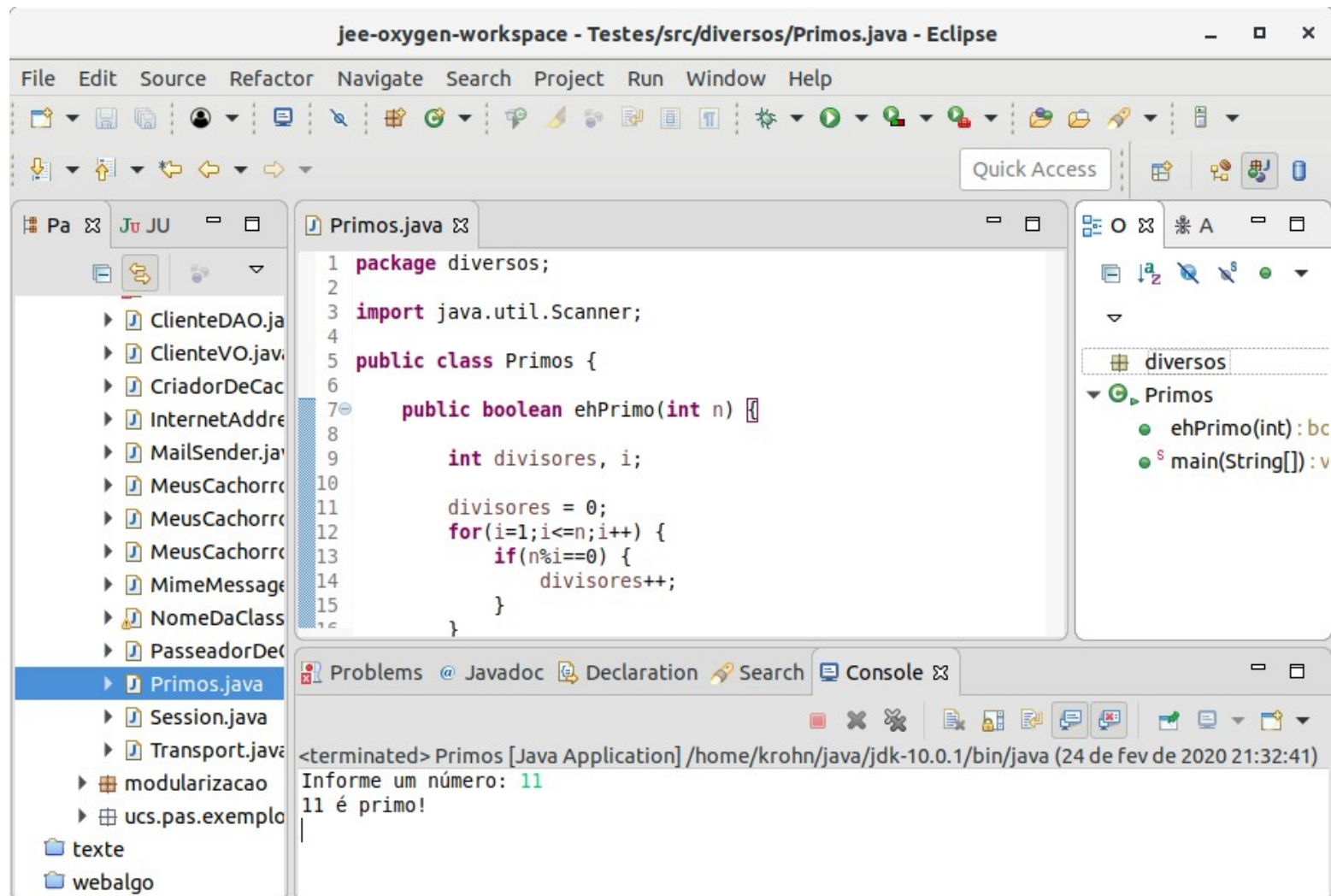
# Como programar Java

- O código-fonte Java é um arquivo texto comum, e pode ser editado com qualquer editor de texto.
- Mas existem ferramentas **IDE** que facilitam muito esse trabalho.

# Ferramentas IDE

- Eclipse

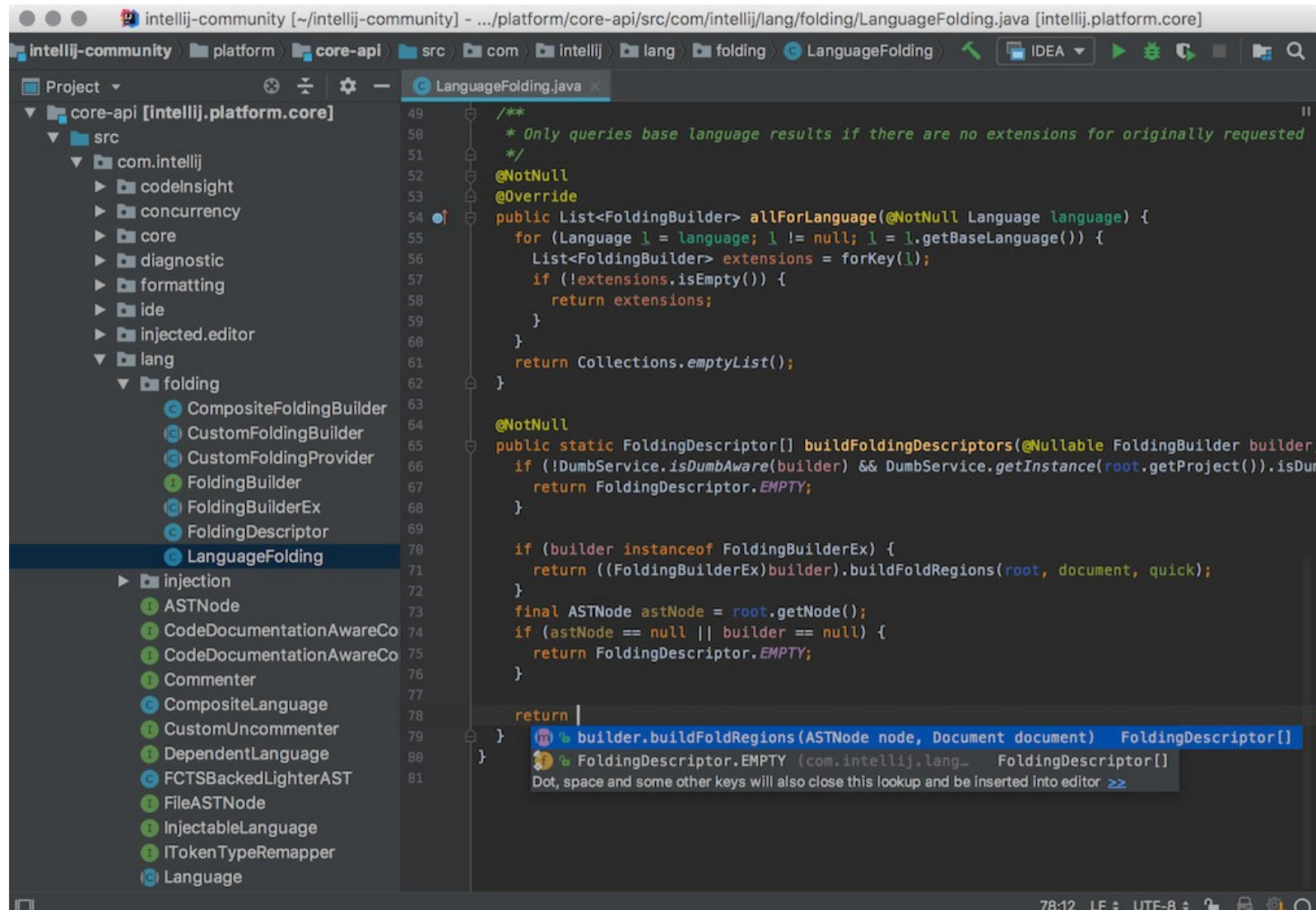
<https://www.eclipse.org/eclipseide/>



# Ferramentas IDE

- IntelliJ

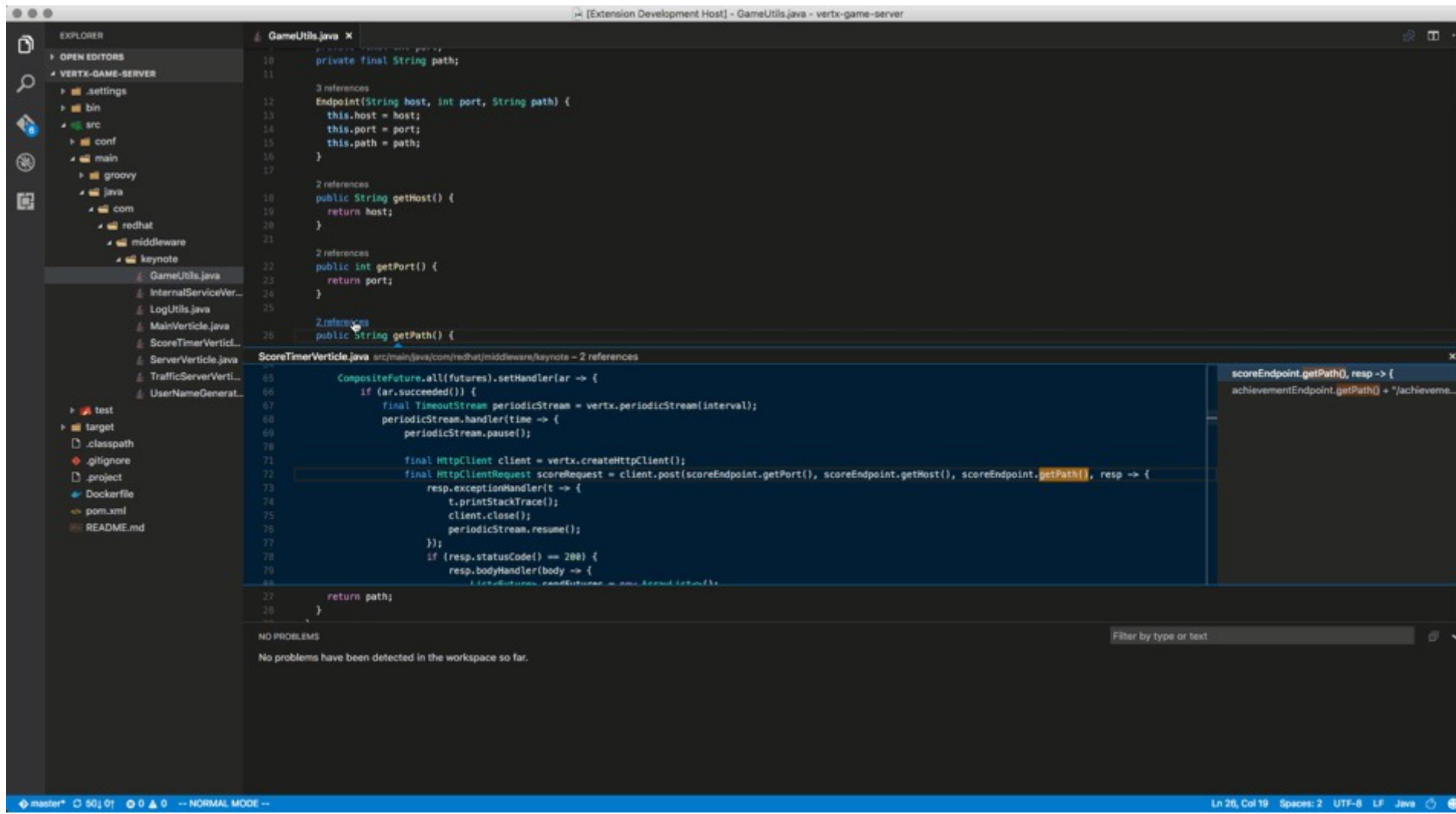
<https://www.jetbrains.com/idea/>



# Ferramentas IDE

- Visual Studio Code (com extensão Red Hat)

<https://code.visualstudio.com/>



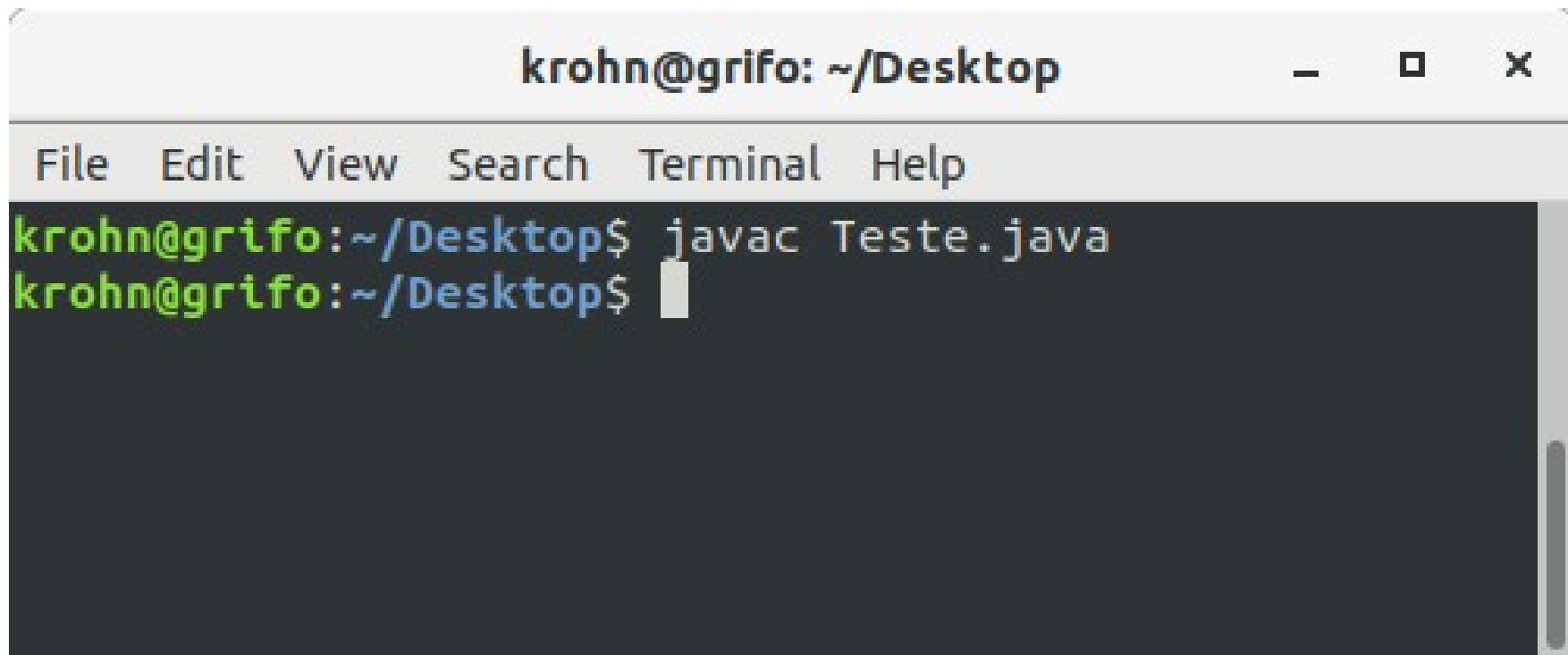
# E em linha de comando?

- Supondo um arquivo chamado Teste.java, com o seguinte conteúdo:

```
public class Teste {  
  
    public static void main(String[] args){  
  
        System.out.println("Oi Mundo!");  
    }  
  
}
```

# Compilando

- Para compilar o código-fonte Java, utiliza-se o comando **javac**:

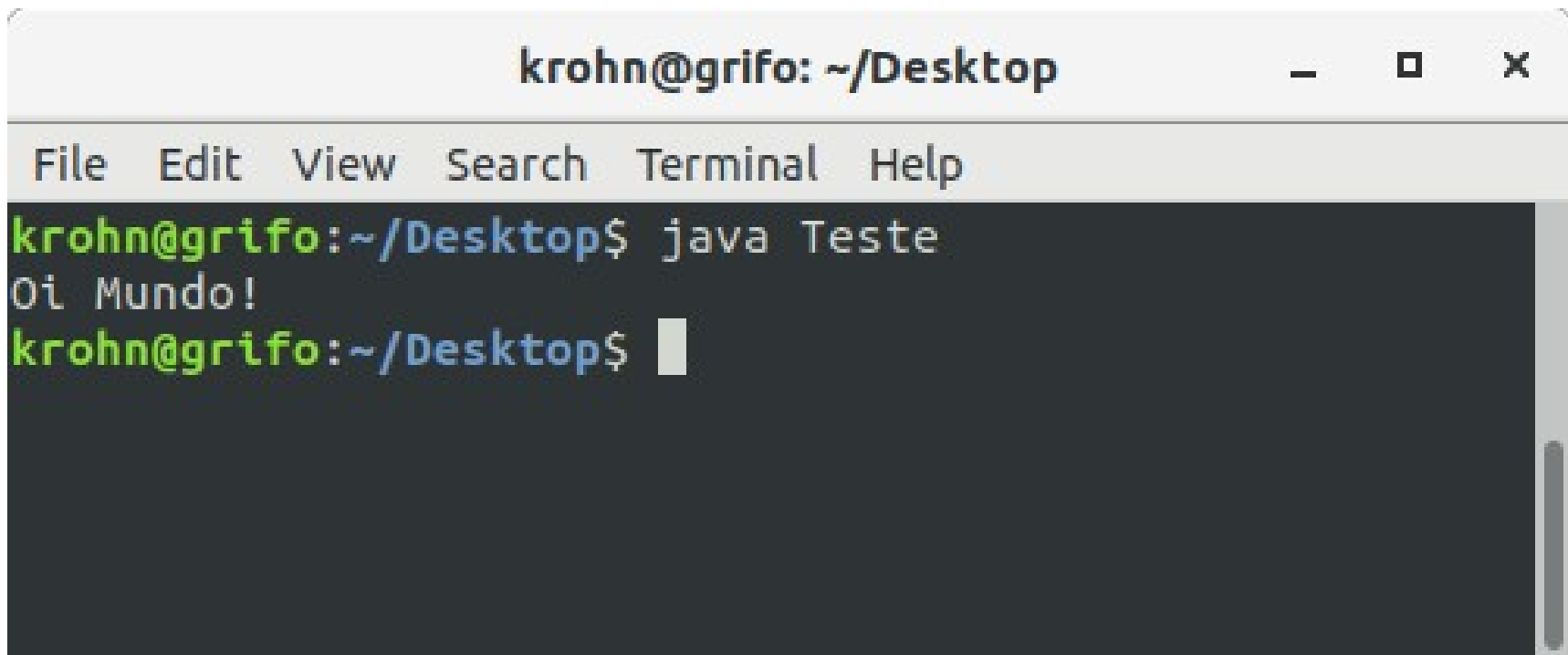


A screenshot of a terminal window titled "krohn@grifo: ~/Desktop". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal shows two lines of text: "krohn@grifo:~/Desktop\$ javac Teste.java" and "krohn@grifo:~/Desktop\$ " with a cursor. The background is dark gray, and the text is in a monospaced font with green and blue highlights for the prompt and path.

```
krohn@grifo: ~/Desktop
File Edit View Search Terminal Help
krohn@grifo:~/Desktop$ javac Teste.java
krohn@grifo:~/Desktop$
```

# Executando

- Para executar o código-fonte Java, utiliza-se o comando **java**:



A screenshot of a terminal window titled "krohn@grifo: ~/Desktop". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal shows the command "java Teste" being executed, which outputs "Oi Mundo!". The prompt "krohn@grifo:~/Desktop\$" is visible at the bottom.

```
krohn@grifo: ~/Desktop
File Edit View Search Terminal Help
krohn@grifo:~/Desktop$ java Teste
Oi Mundo!
krohn@grifo:~/Desktop$
```





# Tipos de dados em Java

- Tipos primitivos
- Tipos objeto

# Tipos primitivos

Classificação	Tipo	Descrição
Lógico	boolean	Pode possuir os valores true (verdadeiro) ou false (falso)
Inteiro	byte	Abrange de -128 a 127 (8 bits)
	short	Abrange de -32768 a 32767 (16 bits)
	int	Abrange de -2147483648 a 2147483647 (32 bits)
	long	Abrange de $-2^{63}$ a $(2^{63})-1$ (64 bits)
Ponto Flutuante	float	Abrange de $1.40239846^{-46}$ a $3.40282347^{+38}$ com precisão simples (32 bits)
	double	Abrange de $4.94065645841246544^{-324}$ a $1.7976931348623157^{+308}$ com precisão dupla (64 bits)
Caracter	char	Pode armazenar um caracteres unicode (16 bits) ou um inteiro entre 0 e 65535

# Tipos Objeto

- Java possui uma série de tipos Objeto prontos para usar:

Primitive Data Type	Wrapper Class
char	Character
byte	Byte
short	Short
long	Integer
float	Float
double	Double
boolean	Boolean



# Tipo Objeto

- As “**wrapper classes**” são classes que permitem que tipos primitivos sejam utilizados como objetos.
- Além disso, há outros dois tipos objetos muito usados:
  - **String**
  - **vetores**



# Tipo Objeto

- E todas as outras classes que o **programador** criar passam a ser tipos objeto, pois a partir deles objetos serão criados.

# Declaração de variáveis

- Para declarar variáveis em Java, basta colocar o tipo da variável e seu nome.

Exemplo:

```
int idade; //armazena números inteiros
float peso; //armazena números reais
//Declarando e inicializando as variáveis
char letra = 'A';
double saldo = 132.65;
```



# Constantes

- **Constante** é uma variável imutável. Para declarar constantes em Java utilizamos a palavra chave **final**.

Exemplo:

```
final int idade = 18;
```

```
final float peso = 70.8;
```





# Entrada e saída na console

- `System.out.println(...)`
- `Scanner`



# System.out.println(...)

- “A instrução `System.out.println()`, gera uma saída de texto entre aspas duplas significando uma `String`, criando uma nova linha e posicionando o cursor na linha abaixo, o que é identificado pela terminação “`\n`””
- Tudo isso para dizer que é a forma de escrever na console



# System.out.println(...)

- System.out.println("teste");

teste

- System.out.println("5");

5

- System.out.println(5);

5

- System.out.println(5 + 5);

10

- System.out.println("5" + "5");

55

- System.out.println("5" + 5);

55 // O número inteiro é automaticamente convertido para String



# Classe Scanner

- Para lermos a partir da console, utilizamos a classe Scanner.
- Ela abre um “fluxo” de dados a partir do teclado, que deve ser fechado depois.

# Classe Scanner

```
int numero;
```

```
//abre a entrada a partir do teclado  
Scanner in = new Scanner(System.in);
```

```
System.out.print("Informe um número: ");  
numero = in.nextInt();
```

```
// faz algo com a informação
```

```
in.close(); // fecha a entrada
```



# Classe Scanner

- Usa-se o método `nextInt()` para ler um número **inteiro** do teclado
- Para lermos outros tipos de dados, há outros métodos:
  - `next()` // lê Strings de uma palavra
  - `nextLine()` // lê frases completas até o “enter”
  - `nextFloat()` // lê floats
  - `nextDouble()` // lê doubles



# Modificadores de Acesso

- Em O.O., usamos modificadores de acesso para definir o **que será “visível”** para outras partes do programa.
- Modificadores são a maneira de implementarmos **Encapsulamento**





# Modificadores de Acesso

- Em java, há quatro modificadores de acesso:
  - `private`
  - `protected`
  - `public`
  - `default`



# Modificadores de Acesso

## private

- É o modificador de acesso mais restritivo que existe. Atributos e métodos declarados como private são acessíveis somente pela classe que os declara.
- Métodos e atributos com o modificador private não são herdados.



# Modificadores de Acesso

## protected

- É um modificador de acesso um pouco mais permissivo que o private. Atributos e métodos declarados como protected **são acessíveis pela classe que os declara, suas subclasses em outros pacotes e outras classes dentro do mesmo pacote.**
- Métodos e atributos declarados com o modificador protected numa superclasse devem ser definidos como protected ou public em suas subclasses e nunca private.



# Modificadores de Acesso

## public

- Modificador de acesso **mais permissivo** que existe. Atributos, métodos e classes declarados como public são **acessíveis por qualquer classe** do Java.
- **Todos** os métodos e atributos declarados como **public são herdados pelas subclasses**.
- Métodos e atributos declarados como public devem se manter public em todas as subclasses.



# Modificadores de Acesso

## default

- Modificador de acesso padrão, **usado quando nenhum for definido.**
- Neste caso os atributos, métodos e classes são visíveis por todas as classes dentro do mesmo **pacote.**



# Modificadores e Encapsulamento

- Tudo o que o usuário externo precisa conhecer a respeito de uma classe encontra-se em propriedades ou métodos declarados como públicos (**public**).
- Somente os códigos membros da classe são capazes de acessar seus métodos e variáveis privados (**private**). Isso garante que não ocorrerão ações inadequadas, mas exige que a interface pública seja planejada com cautela para que o funcionamento interno da classe não seja muito exposto.
- Quer encapsular? Declare **privado**!



# Operadores

- Atribuição
- Aritméticos
- Incremento / Decremento
- Relacionais
- Lógicos



# Operadores de Atribuição

- Com esses atributos podemos alterar e manipular o valor de nossas variáveis.
- A atribuição é feita com **UM** sinal de igual ( = )

Exemplo:

```
int v = 10;    // v = 10
v += 10;       // v = 20
v -= 5;        // v = 15
v *= 2;        // v = 30
v /= 3         // v = 10
v %= 3;        // v = 1 (resto de divisão inteira)
```





# Operadores Aritméticos

+ Adição

- Subtração

\* Multiplicação

/ Divisão

% Módulo ou resto da divisão inteira

# Incremento / Decremento

- Os operadores de incremento e decremento incrementam ou decrementam em 1 o valor da variável.
- São dois : `++` e `--`, e quais podem ser declarados antes ou depois da variável.

```
int a = 5;
```

```
a++; // a = 6
```

```
a++; // a = 7
```

```
a--; // a = 6
```



# Incremento / Decremento

- Quanto vale **c** ?

```
int a = 5;
```

```
int b = 6;
```

```
int c = (a++) + (--b);
```

# Incremento / Decremento

- Quanto vale **c** ?

```
int a = 5;
```

```
int b = 6;
```

```
int c = (a++ + --b);
```

Primeiro lê o valor,  
depois incrementa

Primeiro decrementa,  
depois lê o valor



# Incremento / Decremento

- Quanto vale **c** ?

```
int a = 5;
```

```
int b = 6;
```

```
int c = (a++) + (--b);
```

```
// a = 6
```

```
// b = 5
```

```
// c = 10
```



# Operadores Relacionais

- Utilizamos os operadores relacionais quando precisamos determinar a relação entre uma variável e/ou um valor. A operação realizada com operadores relacionais retornam valores do tipo primitivo boolean (true/false).
- Note que para compararmos valores o símbolo de igualdade são **DOIS** sinais de igual ( `==` )

Exemplo:

`==` (Igual) `!=` (Diferente) `>` (Maior) `<` (Menor) `>=` (Maior ou igual) `<=` (Menor ou igual)

```
int v = 10;
```

```
boolean x = false;
```

```
x = (v == 10); // x = true
```

```
x = (v > 10); // x = false
```

```
x = (v < 10); // x = false
```

```
x = (v != 10); // x = false
```

```
x = (v >= 10); // x = true
```

```
x = (v <= 10); // x = true
```

# Operadores Lógicos

- Os operadores lógicos são usados para analisarem expressões. Assim como os operadores relacionais, os lógicos também retorna um tipo primitivo boolean.
- **&&** (E)      **||** (OU)

Exemplo:

```
int v = 10;
```

```
boolean x = false;
```

```
x = v > 10 && v < 100; // x = false
```

```
x = v > 10 || v < 100; // x = true
```



# Instanciando um objeto

- Para criar objetos, usamos o comando ***new***
- ***new*** faz a alocação da memória e cria o objeto invocando o construtor chamado



# Instanciação : Exemplo

Primos p = *new* Primos();

Tipo  
(nome da classe)

Palavra-chave  
*new*

Variável  
(referência ou ponteiro  
para o objeto)

Método construtor a ser  
chamado

# Método *main()*

- Ao contrário de programação em C, Java permite que cada classe possua um método *main()*
- O método *main()* é sempre o primeiro método a ser chamado quando a classe é executada.

# Método *main()*

- A assinatura (estrutura) do método *main()* é sempre a mesma:

```
public static void main(String[ ] args) {  
    // corpo do método  
}
```

# Hello World!

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

# Hello World!

```
public class HelloWorld {  
  
    public static void main(String[] args) {  
        HelloWorld hw = new HelloWorld();  
        hw.sayHello();  
    }  
  
    public void sayHello() {  
        System.out.println("Hello World!");  
    }  
  
}
```

Aqui podemos ver como  
um método é chamado:

Usa-se o nome da variável “ponto”  
o nome do método

# Próximos passos



- Controle de fluxo
- Primeiras classes
- Herança



# Dúvidas?





# Atividades

- Execute as atividades presentes no documento

01.Lista.de.Exercícios.POO.pdf





# Material online

- O livro do Rafael Santos está disponível em:
  - <https://pt.slideshare.net/Dodobernardo/rafael-santos-introducao-poo-usando-java>
- Também há uma apostila, muito boa, de uma empresa de cursos chamada Caelum:
  - <https://www.caelum.com.br/download/caelum-java-objetos-fj11.pdf>



# Referências

- Richard Weiner e Lewis Pinson, **Programação Orientada para Objeto e C++**
- Ann Winblad e outros, **Software Orientado ao Objeto**
- Rafael Santos, **Introdução A Programação Orientada A Objetos: USANDO JAVA**
- <https://www.devmedia.com.br/os-4-pilares-da-programacao-orientada-a-objetos/9264>