

# **Desenvolvimento de Software**

Cássio Seffrin

## Desenvolvimento de Software

- Metodologia Procedural (PE)
  - Metodologia Orientada o Objetos (POO)
  - Objeto (estrutura de dados que representam estado / comportamento)
- 
- Fortran IBM (1954)
  - Basic 65 Microsoft
  - Simula 67 POO
  - Pascal foi criada em 1970 pelo suíço Niklaus Wirth
  - Smalltalk 72-80 POO
  - C(73), C++ (80), C#, Objective C/Swift
  - Python 1991 POO
  - Java (1992)
  - PHP Rasmus Lerdorf (1995)
  - Ruby (1995)
  - Lua (nmap Scripting Engine) - The Lua language is designed, implemented, and maintained at PUC-Rio in Brazil since 1993 (Waldemar, Roberto, Luiz)

## Desenvolvimento de Software

O imperativo é um dos modos verbais, juntamente com o modo indicativo e o modo subjuntivo. No modo imperativo a pessoa falante leva o seu interlocutor a realizar uma ação, expressando o que quer que ele faça.

Assim, a ação transmitida por um verbo no imperativo é um pedido, convite, exortação, ordem, comando, conselho ou súplica.

O imperativo se divide em imperativo afirmativo e imperativo negativo, sendo conjugados de forma diferente. Em ambos não existe flexão na 1.<sup>a</sup> pessoa do singular (eu).

Exemplos de uso do modo imperativo

Pare com essa brincadeira.

Jogue o lixo fora, por favor.

Resolva esse problema rápido.

Vai fazer o dever de casa.

Sai da frente!

Para!

## Desenvolvimento de Software

“Na Ciência da Computação, programação imperativa é um paradigma de programação que descreve a computação como ações, enunciados ou comandos que mudam o estado (variáveis) de um programa. Muito parecido com o comportamento imperativo das linguagens naturais que expressam ordens, programas imperativos são uma sequência de comandos para o computador executar. O nome do paradigma, Imperativo, está ligado ao tempo verbal imperativo, onde o programador diz ao computador: faça isso, depois isso, depois aquilo... Este paradigma de programação se destaca pela simplicidade, uma vez que todo ser humano, ao se programar, o faz imperativamente, baseado na ideia de ações e estados, quase como um programa de computador.”  
Fonte wiki 2020

# Desenvolvimento de Software

Exemplos de linguagens de programação que baseiam-se no modelo imperativo:

Assembler

Basic

C/C++/C#

Cobol

Fortran

Java

Javascript

Lua

Pascal

PHP

Python

Ruby

Typescript

## Desenvolvimento de Software

Exemplos de linguagens de programação que baseiam-se no modelo funcional:

Javascript

Typescript

LISP

Clojure

Python

Erlang

Haskell

Linguagens funcionais puras - esses tipos de linguagens funcionais suportam apenas os paradigmas funcionais. Por exemplo - Haskell.

Linguagens Funcionais Impuras - Esses tipos de linguagens funcionais oferecem suporte aos paradigmas funcionais e à programação de estilo imperativo. Exemplo - LISP.

# Desenvolvimento de Software

Exemplos de linguagens de programação que baseiam-se no modelo POO:

Simula

Smalltalk

C++, C#, Objective C

Swift

Typescript

Java

Dart

Python

Clojure

## Desenvolvimento de Software

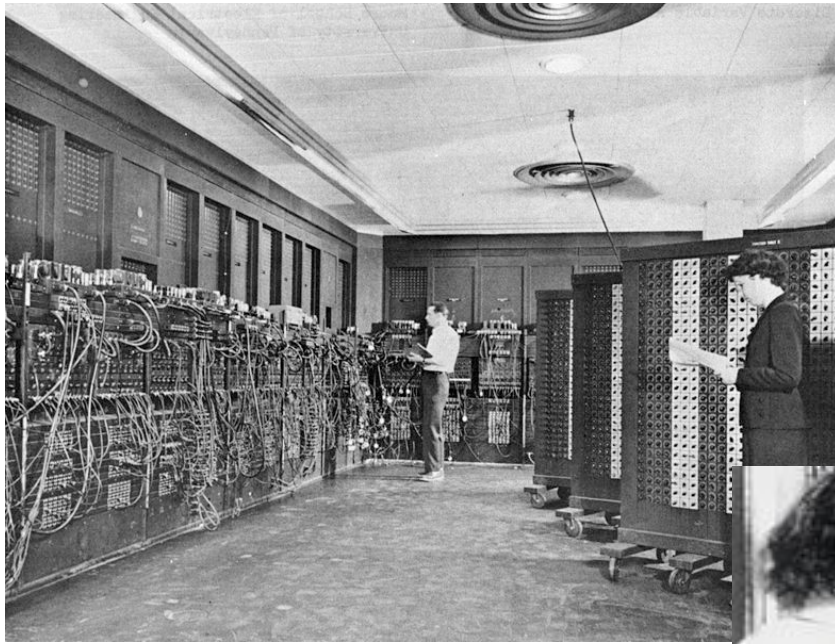
JavaScript é uma linguagem multi-paradigma, permite misturar e combinar paradigmas orientados a objetos, procedurais e funcionais. Recentemente, tem havido uma tendência crescente para a programação funcional. Em estruturas como Angular e React, você realmente obterá um aumento de desempenho usando estruturas de dados imutáveis. A imutabilidade é um princípio básico da programação funcional.

Uma função pura é uma função que possui as seguintes propriedades: Seu valor de retorno é o mesmo para os mesmos argumentos (sem variação com variáveis estáticas locais, variáveis não locais, argumentos de referência mutáveis ou fluxos de entrada de dispositivos de E / S )

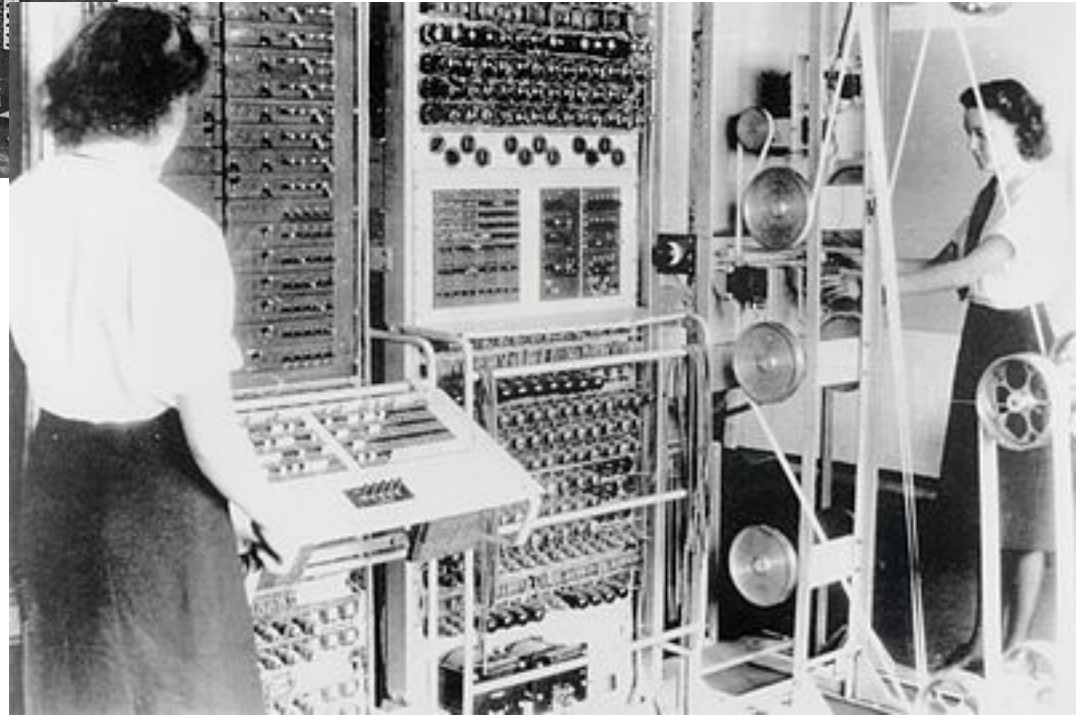
ImmutableJS

Immer





ENIAC 1945 - 1947 - US



Computador Colossus sendo operado UK  
Mark I- Dezembro de 1943 e  
Mark II – 1 de Junho de 1944

**Simula 67**, cuja primeira versão foi apresentada em 1966 foi a 1ª linguagem orientada a objetos e introduziu os conceitos de classes e herança.

Exemplo de Classe em Simula:

```
Glyph Class Line (elements);  
  Ref (Glyph) Array elements;  
  Begin  
    Procedure print;  
      Begin  
        Integer i;  
        For i:= 1 Step 1 Until UpperBound (elements, 1) Do  
          elements (i).print;  
        OutImage;  
      End;  
    End;  
  End;
```



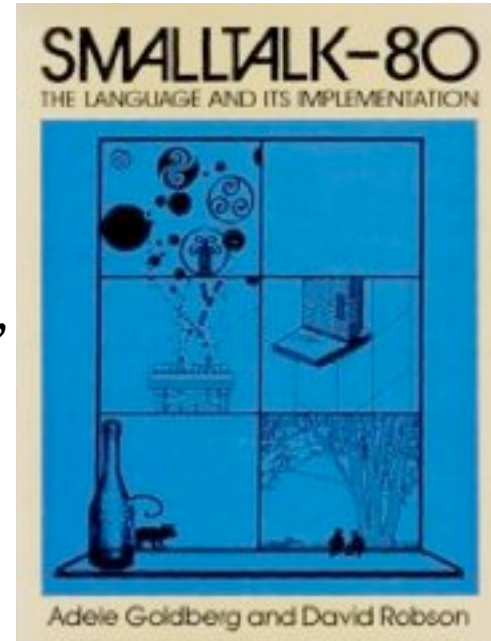
Ken Thompson e Dennis Ritchie (C 1972 - Unix 1983)

AT&T Bell Labs entre [1969](#) e [1973](#)

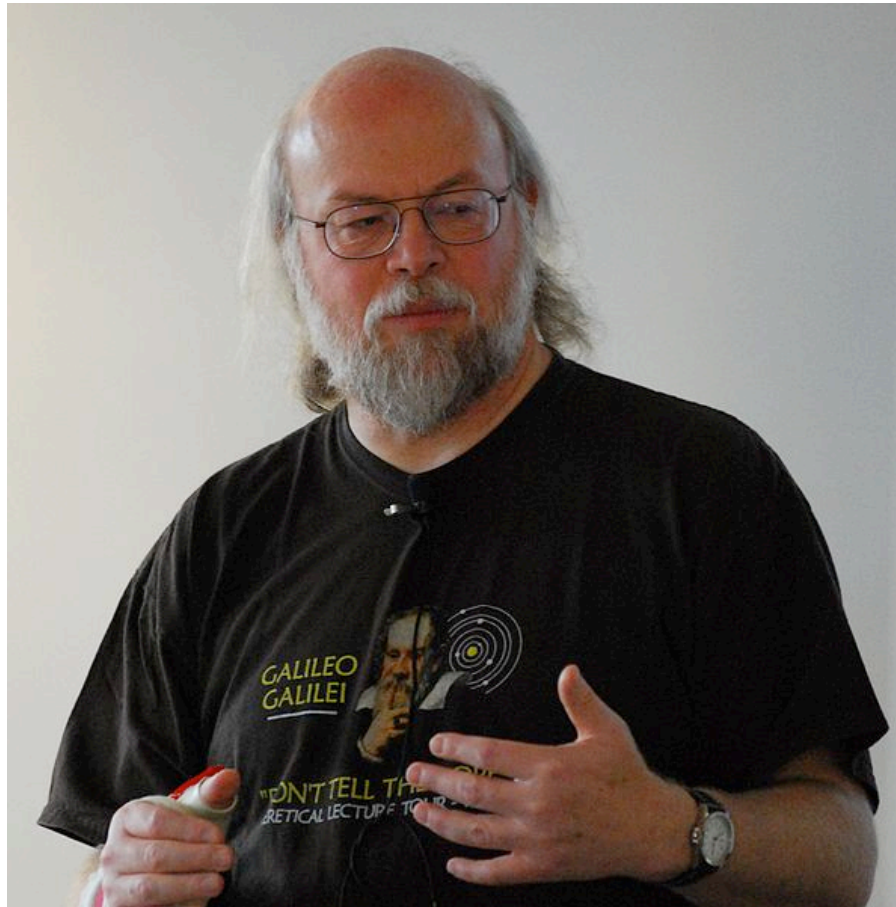


Richard Matthew Stallman: FSF e GNU

Smalltalk-80, ou simplesmente Smalltalk, é uma linguagem de programação orientada a objeto fracamente tipada.



Em *Smalltalk* tudo é objeto: os números, as classes, os métodos, blocos de código, etc. Não há tipos primitivos, ao contrário de outras linguagens orientadas a objeto; strings, números e caracteres são implementados como classes em *Smalltalk*, por isso esta linguagem é considerada puramente orientada a objetos. Tecnicamente, todo elemento de *Smalltalk* é um objeto de primeira ordem.



James Gosling

# Linguagem Java

- Oak (árvore de carvalho)
- Java
- POO
- Compilador Bytecode (javac Pessoa.java -> Pessoa.class)
- Tipagem forte
- Classes
- Atributos
- Construtores
- Métodos
- Modificadores de acesso (private, protected (subclasse), public)
- Outros Modificadores (final/static/abstract)
- Objeto



# Tipos Primitivos

Tipos	Primitivo	Valores possíveis		Valor Padrão	Tamanho	Exemplo
		Menor	Maior			
Inteiro	byte	-128	127	0	8 bits	byte ex1 = (byte)1;
	short	-32768	32767	0	16 bits	short ex2 = (short)1;
	int	-2.147.483.648	2.147.483.647	0	32 bits	int ex3 = 1;
	long	-9.223.372.036.854.770.000	9.223.372.036.854.770.000	0	64 bits	long ex4 = 1l;
Ponto Flutuante	float	-1,4024E-37	3.40282347E + 38	0	32 bits	float ex5 = 5.50f;
	double	-4,94E-307	1.79769313486231570E + 308	0	64 bits	double ex6 = 10.20d; ou double ex6 = 10.20;
Caractere	char	0	65535	\0	16 bits	char ex7 = 194; ou char ex8 = 'a';
Booleano	boolean	false	true	false	1 bit	boolean ex9 = true;



# Linguagem Java

## Exemplo de um método em java

```
class Matematica {
    int multiplica(int a, int b){
        return a*b;
    }
    double raiz(float n){
        return Math.sqrt(n);
    }
    // divisao
    // adicao
    // subtracao
}

class Principal{
    public static void main(String a[]){
        Matematica mat = new Matematica();
        System.out.println("resultado de 2*4: "+
mat.multiplica(2,4) );
        System.out.println("raiz de 9: "+mat.raiz(9) );
    }
}
```

# Linguagem Java

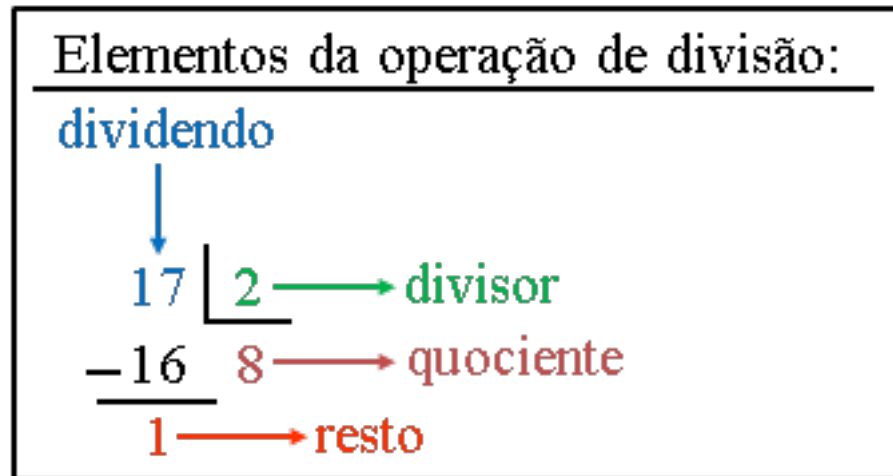
## recursividade

- Recursividade é quando uma função/método chama ele mesmo.

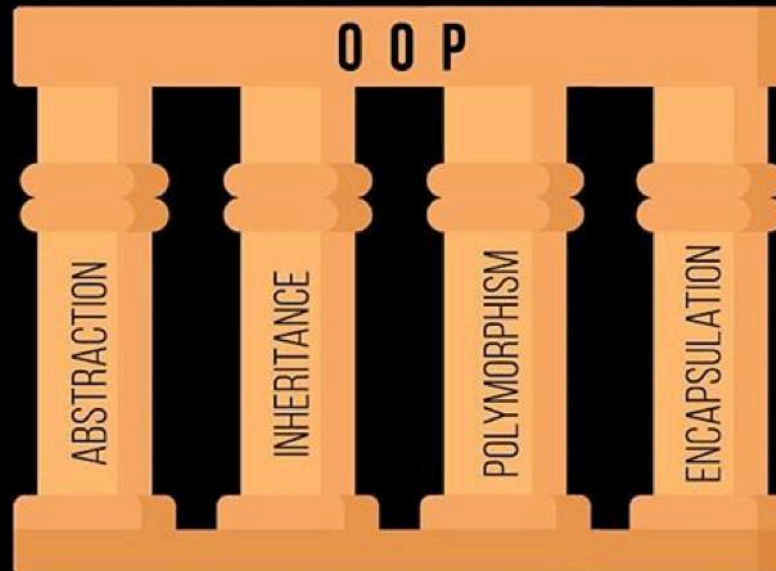
```
#include <stdio.h>
int fatorial(unsigned int i) {
    if(i <= 1) {
        return 1;
    }
    return i * fatorial(i - 1);
}

int main() {
    int i = 7;
    printf("fatorial de %d é %d\n", i, fatorial(i));
    return 0;
}
```

# Linguagem Java



# PILLARS OF OBJECT-ORIENTED PROGRAMMING



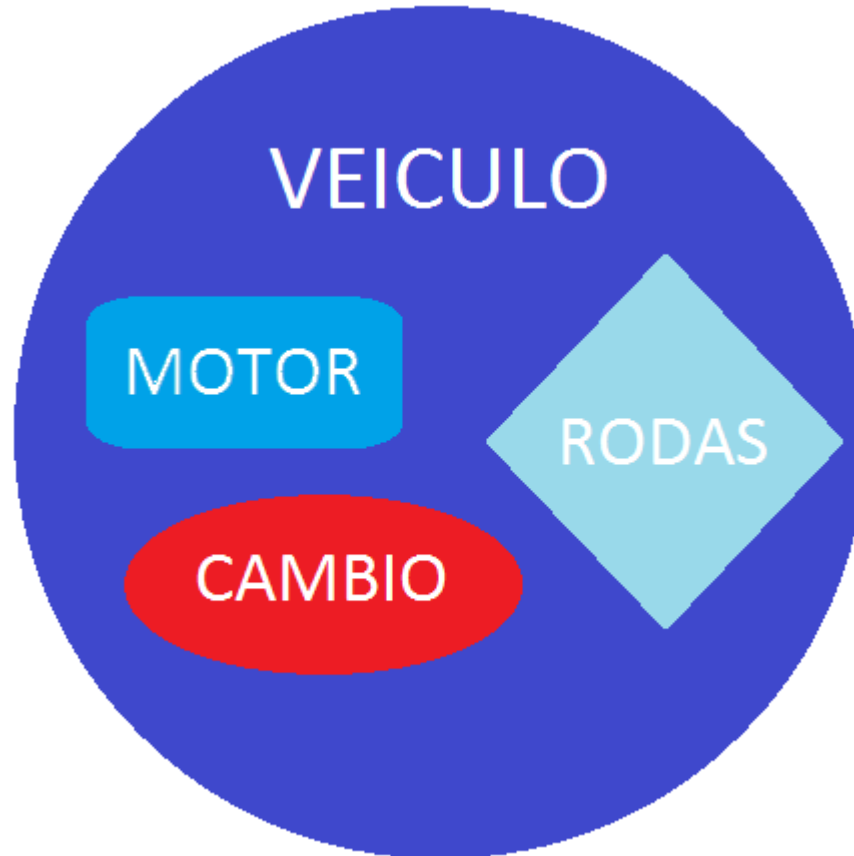
# Linguagem Java

Os 4 pilares

- Abstração / Interfaces
- Polimorfismo (parâmetros)
- Composição (todo/parte)
- Herança (múltipla?)

# Linguagem Java

Exemplo de Composição



# Linguagem Java

## Exemplo de uma interface

```
interface BicicletaInterface {  
    void mudarMarcha(int newValue);  
    void aumentarVelocidade(int increment);  
    void freiar ();  
}
```

```
class Bicicleta implements BicicletaInterface {  
    /* comentário Java:  
       implementar os 3 métodos da interface  
       BicicletaInterface  
    */  
}
```

# Linguagem Java

## Exemplo de uma herança

```
abstract class Pessoa {  
    private String nome;  
    private Integer idade;  
    private String sexo;  
    private Date dataNascimento;  
    public Pessoa(){}  
    /* .... Getters e setters .... */  
}
```

```
class Funcionario extends Pessoa{  
    private int salario ;  
    public Funcionario(){}  
    /* .... Getters e setters .... */  
}
```

```
public static void main (String a[]){  
    Funcionario f = new Funcionario();  
    f.setNome("Cassio");  
    f.setSalario(2000);  
    System.out.println("Nome: "+ f.getNome() + " Salario: "  
+f.getSalario());  
}
```



# Linguagem Java

## Classes Abstratas

- Servem como modelo para uma classe concreta
- Não podem ser instanciadas diretamente
- Podem conter ou não métodos abstratos
- Pode implementar ou não um método

Exemplo no netbeans

# Linguagem Java

## Classes Abstratas

- Servem como modelo para uma classe concreta
- Não podem ser instanciadas diretamente
- Podem conter ou não métodos abstratos
- Pode implementar ou não um método

Exemplo no netbeans

## Modelagem de sistema de software

- Complexidade cresce a medida que o software aumenta
  - Casa para o cachorro
  - Casa para família
  - Edifício
  - Software
- Características tratadas com diversos modelos, analogamente ao um avião que possui um diagrama elétrico, outro para aerodinâmica.
- Gerenciamento da complexidade
  - Comunicação entre as pessoas envolvidas
  - Redução dos custos no desenvolvimento
  - Previsão do comportamento futuro do sistema

## UML

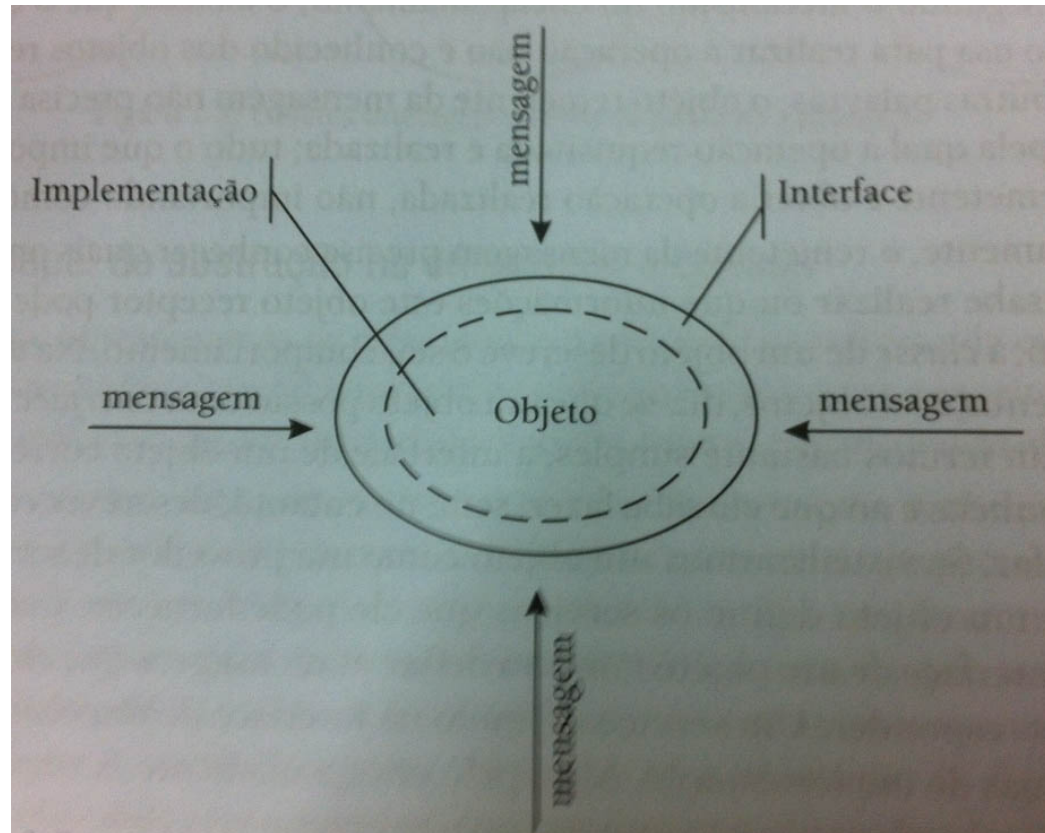
- Grandy Booch, James Rumbaugh e Ivar Jacobson. Os 3 amigos.
- Em 1997 a UML foi aprovada como padrão pelo OMG (Object Management Group. Consórcio Internacional que define e ratifica padrões na área de orientação a objetos.

## Modelagem de sistema de software

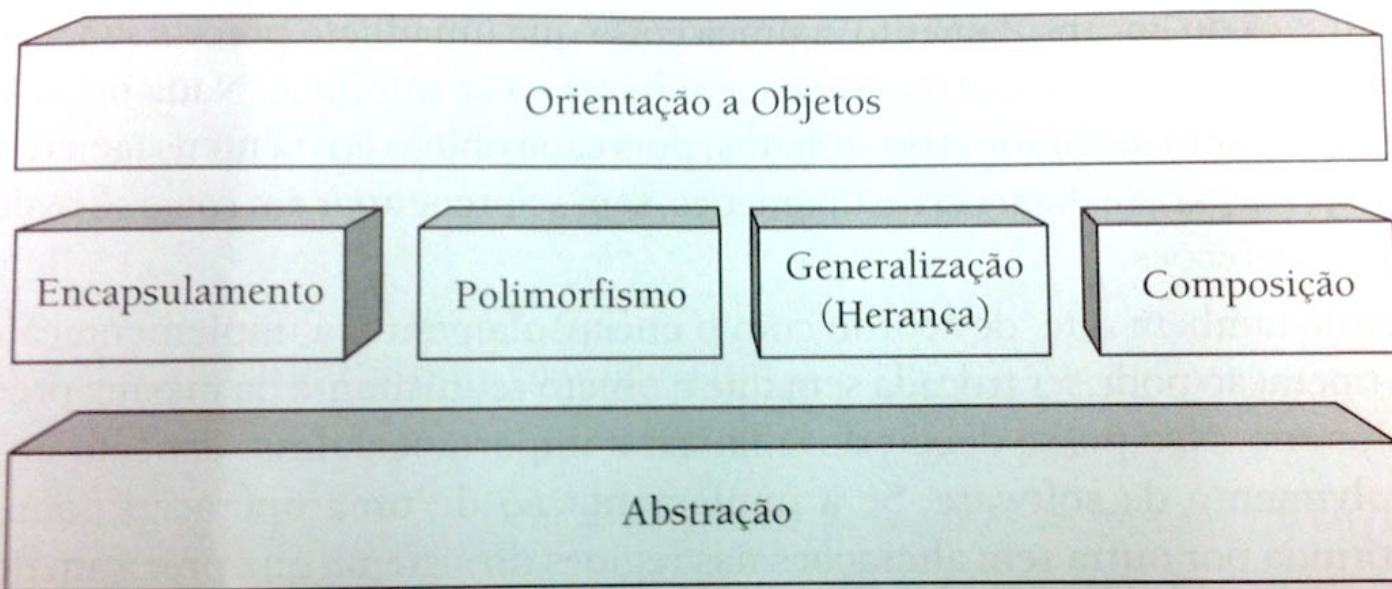
O paradigma da orientação a objetos (uma forma de abordar um problema)

1. Qualquer coisa é um objeto
2. Objetos realizam tarefas por meio da requisição de serviços a outros objetos.
3. Cada objeto pertence a uma determinada classe. Uma classe agrupa objetos similares.
4. A classe é um repositório para comportamento associados ao objeto.
5. Classes são organizadas em hierarquias.

## Modelagem de sistema de software



Fonte: Lorenzo Ridolfi, Sérgio Colcher (2007)



*Figura 1-2: Princípios da orientação a objetos podem ser vistos como aplicações de um princípio mais básico, o da abstração.*

Fonte: Lorenzo Ridolfi, Sérgio Colcher (2007)

## Encapsulamento

- Os objetos possuem comportamento (métodos)
- O encapsulamento é uma forma de restringir o acesso ao comportamento interno dos objetos.
- A relação entre objetos é feita através da troca de mensagens



## Herança (Generalização)

- Na generalização, classes semelhantes são agrupadas em uma hierarquia.
- Exemplo: um funcionário estende de uma classe de pessoa.

## Composição

- Objetos que compõe outros objetos
- Um livro é composto de paginas, capa, títulos, parágrafos.

## Polimorfismo

- O polimorfismo indica a capacidade de abstrair varias implementações diferentes em uma única interface. (Lorenzo Ridolfi, Sérgio Colcher)
- O polimorfismo permite programar no geral ao invés de programar no específico (deitel & deitel).

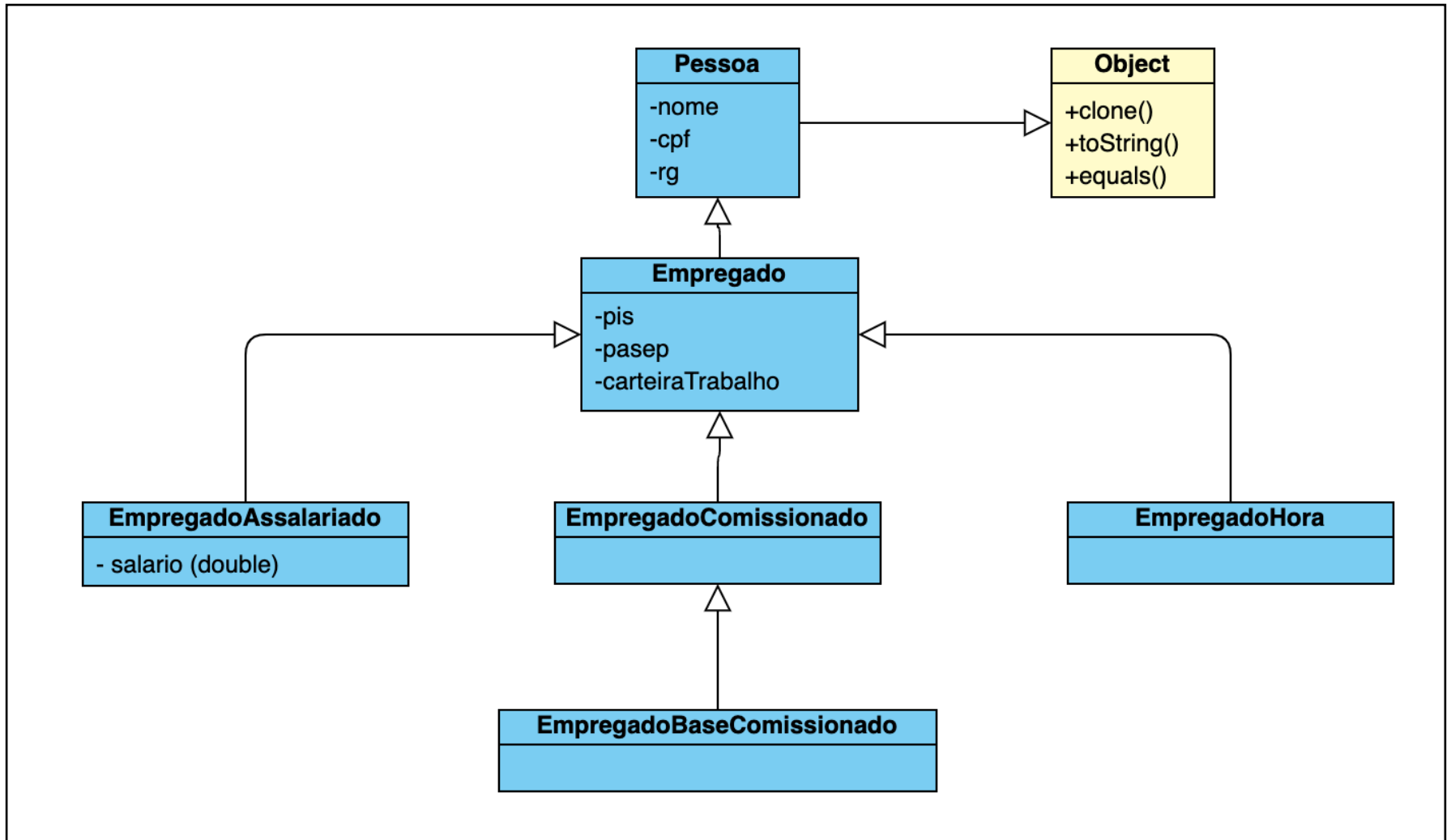
Exemplos de polimorfismo.

Sobrescrita e Sobrecarga de Métodos;

## Abstração

- Em java, a abstração é obtida por interfaces e classes abstratas. Podemos atingir 100% de abstração usando interfaces.
- O processo de abstração é usado para esconder certos detalhes e somente mostrar as características essenciais de um objeto, ou seja, apresentar uma visão externa de um objeto (interface)

## Diagrama de Classe



# Generics

Apartir Java 5

O que é um tipo Genérico

Para que serve?

Exemplo de Array[] simples

# Generics

```
Integer[] intArr = {1, 2, 3, 4, 5};  
Double[] douArr = {1.2, 2.1, 3.4, 4.5, 5.6};  
String[] strArr = {"Cassio", "Fred", "Juliana"};
```

Como imprimir sem Generics?

Como imprimir com Generics?

# Generics

- Tipos genéricos em Arrays funcionam somente substituindo tipos por referencias como Integer, Float, String. Tipos primitivos (int, char, float) não funcionam
- Os nomes dos parâmetros de tipo por toda declaracao do método devem corresponder aqueles declarados na assinatura do metodo
- O tipo Object pode ser usado invés do conceito generics quando o retorno do método é void.



# Generics

Convenção para determinar nomes para Genéricos

- E – Element (usado para todos elementos do Java Collections Framework, exemplo ArrayList, Set etc.)
- K – Key (usado na coleção Map)
- N – Number
- T – Type
- V – Value (usado na coleção Map)
- S,U,V etc. – T, U, V etc.. quando não sabemos o tipo
- U (undefined) = indefinido

# Escrita de arquivos

```
File f = new File("/Users/cassioseffrin/teste.txt");  
FileOutputStream fos = new FileOutputStream(f, true);  
fos.write("nome; cpf; rg; endereco; ".getBytes());  
fos.flush();  
fos.close();
```

# Leitura de arquivos

```
File f = new File("/Users/cassioseffrin/teste.txt");  
FileInputStream fis = new FileInputStream(f);  
Scanner scanner = new Scanner(fis);  
while (scanner.hasNextLine()) {  
    System.out.println(scanner.nextLine());  
}  
scanner.close();
```

## Bibliografia

Lorenzo Ridolfi, Sérgio Colcher. Princípios de Análise e Projeto de Sistemas com UML,

FREEMAN & FREEMAN, Eric & E. Padrões de Projetos: Seu cérebro em padrões de projetos. Rio de Janeiro: ALTABOOKS, 2007.

METSKER, S. J. Padrões de projeto em Java. Porto Alegre: Bookman, 2004.

ANSELMO, F. Aplicando Lógica OO [Orientada a Objetos] em JAVA. 2. ed. Florianópolis: Visual Books Ltda., 2005. 178 p. ISBN 85-7502-162-1.