

# Paradigma orientado a objetos

André Luís Mendes Fakhoury

Eduardo Dias Pennone

Gustavo Vinícius Vieira Silva Soares

Matheus Steigenberg Populim

Thiago Preischadt Pinheiro

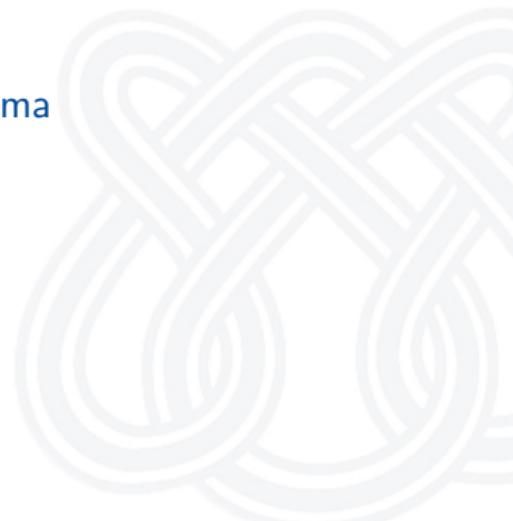
SCC0217 - LINGUAGENS DE PROGRAMAÇÃO E  
COMPILADORES

Prof. Diego Raphael Amancio

ICMC - USP

1 de julho de 2021

# Sumário



Introdução

Conceitos importantes presentes no paradigma

Herança, encapsulamento e polimorfismo

Tipagem dinâmica e estática

Exemplos de linguagens

Smalltalk

C++ e Java

Questões de design e implementação

Referências

# Sumário

## Introdução

Conceitos importantes presentes no paradigma

Herança, encapsulamento e polimorfismo

Tipagem dinâmica e estática

Exemplos de linguagens

Smalltalk

C++ e Java

Questões de design e implementação

Referências



# Introdução

## Histórico

- O conceito de programação orientada a objetos teve suas raízes no SIMULA 67, mas não foi totalmente desenvolvido até que a evolução do Smalltalk resultou no Smalltalk 80.
- Muitos consideram o Smalltalk como sendo o modelo básico para uma linguagem de programação puramente orientada a objetos.

# Introdução

## O paradigma da orientação a objetos

- Uma linguagem orientada a objetos deve fornecer suporte para três recursos-chave da linguagem: tipos de dados abstratos, herança e vinculação dinâmica de chamadas de método a métodos.
- Com base nesses recursos temos os 4 pilares da orientação a objetos: Abstração, Encapsulamento, Herança e Polimorfismo.

# Sumário

Introdução

Conceitos importantes presentes no paradigma

Herança, encapsulamento e polimorfismo

Tipagem dinâmica e estática

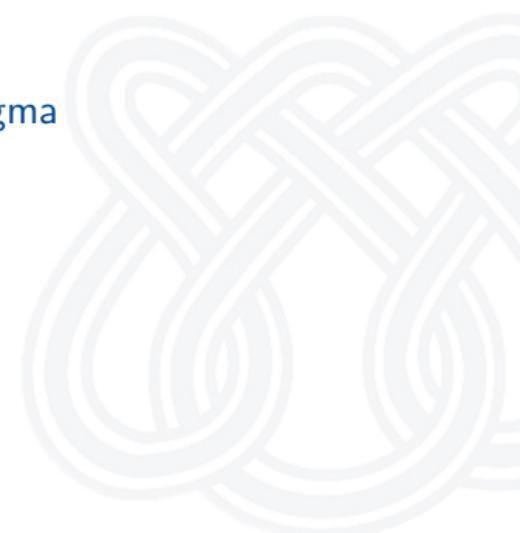
Exemplos de linguagens

Smalltalk

C++ e Java

Questões de design e implementação

Referências



# Conceitos importantes presentes no paradigma

## Classe

- Uma abstração de um conjunto de objetos que possuem características em comum.
- A descrição de propriedades, estados, comportamentos e ações de um conjunto de objetos.

# Conceitos importantes presentes no paradigma

## Objeto

- A instância de uma classe.
- Representação de um objeto do mundo real pertencente a uma classe especificada.
- Caracterizado por atributos e métodos.

# Conceitos importantes presentes no paradigma

## Atributos

- As características, propriedades ou estados de um objeto.
- Exemplos: tamanho, cor, peso.

# Conceitos importantes presentes no paradigma

## Métodos

- Os comportamentos, ações ou funcionalidades que um objeto pode desempenhar.
- Exemplos: comer, voar, latir.

# Conceitos importantes presentes no paradigma

## Abstração

- Uma abstração é uma visão ou representação de uma entidade que inclui apenas os atributos mais significativos. Em um sentido geral, a abstração permite coletar instâncias de entidades em grupos nos quais seus atributos comuns não precisam ser considerados.

# Conceitos importantes presentes no paradigma

## Herança

- A herança possibilita que as classes compartilhem seus métodos e atributos entre si seguindo uma hierarquia baseada nos níveis de abstração.
- Facilitam a reutilização do código, uma vez que atributos e métodos das classes superiores podem ser aproveitados.

# Conceitos importantes presentes no paradigma

## Encapsulamento

- Controla o acesso à atributos e métodos de uma classe.
- Evitar o acesso direto a propriedade do objeto adiciona segurança à aplicação.
- O código pode evoluir a partir de uma interface em comum.

# Conceitos importantes presentes no paradigma

## Polimorfismo

- Consiste na alteração do funcionamento interno de um método herdado de um objeto pai.
- Subclasses diferentes que possuem uma mesma classe pai, possuem comportamentos diferentes em relação ao mesmo método herdado.

# Sumário

Introdução

Conceitos importantes presentes no paradigma

**Herança, encapsulamento e polimorfismo**

Tipagem dinâmica e estática

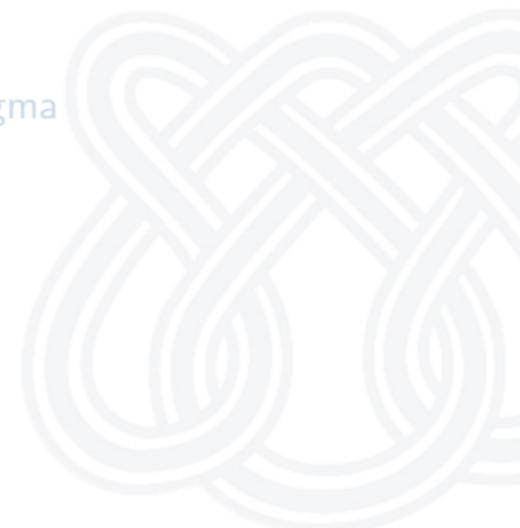
Exemplos de linguagens

Smalltalk

C++ e Java

Questões de design e implementação

Referências



# Herança

- Criar uma classe derivada a partir de uma classe base
- Classe derivada herda os atributos e métodos da base

# Herança

- Normalmente os construtores e destrutores não são herdados
- Métodos herdados podem ser sobreescritos
- Novos métodos e atributos podem ser adicionados

# Herança

## Tipos

- Simples
- Múltipla
- Multinível



# Herança

## Simples

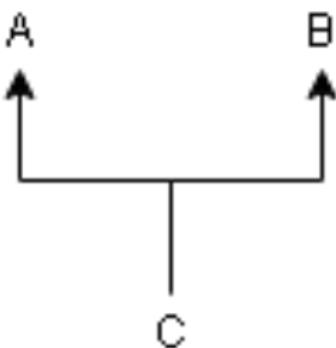
- Cada classe tem no máximo uma superclasse
- A hierarquia pode ser representada por uma árvore



# Herança

## Múltipla

- Cada classe pode ter diversas superclasses
- Pode ser necessário tratar ambiguidades na herança
- A hierarquia pode ser representada por um grafo direcionado



# Herança

## Multinível

- Uma classe derivada pode ser base de outra classe



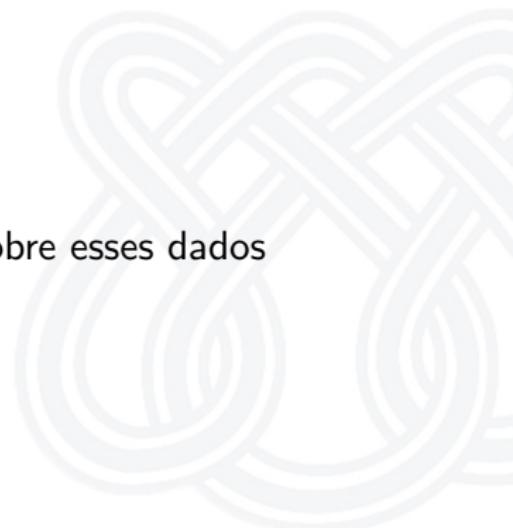
# Herança

## Usos

- Reuso de código
- Implementar uma mesma interface de diferentes formas

# Encapsulamento

- Agrupar dados e métodos que agem sobre esses dados
- Restringir o acesso direto aos dados



# Encapsulamento

## Usos

- Esconder informação
- Esconder detalhes de implementação
- Evitar que o objeto entre em um estado inválido



# Polimorfismo

- Utilizar o mesmo símbolo para representar vários tipos
- Tratar objetos de uma classe derivada com objetos da classe base

# Polimorfismo

## Tipos

- Ad hoc
- Paramétrico
- Por subtipos



# Polimorfismo

Ad hoc

- Sobrecarga de operador/função
- Definir diversas funções com mesmo nome e quantidade de argumentos, mas com diferentes tipos

```
soma(a: int, b: int) : int
soma(a: float, b: float) : float
```

# Polimorfismo

## Paramétrico

- Definir funções que aceita um tipo genérico
- O comportamento da função é uniforme em relação ao tipo da entrada

# Polimorfismo

Por subtipos

- Definir uma função que aceita um tipo A
- A função funciona corretamente para um subtipo de A

# Sumário

Introdução

Conceitos importantes presentes no paradigma

Herança, encapsulamento e polimorfismo

Tipagem dinâmica e estática

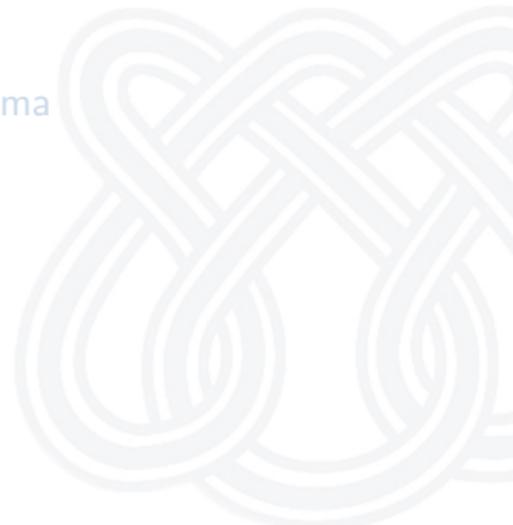
Exemplos de linguagens

Smalltalk

C++ e Java

Questões de design e implementação

Referências



# Tipagem dinâmica e estática

## Escolha de design

- No paradigma orientado a objetos, as linguagens possuem uma escolha com relação à tipagem: dinâmica ou estática
- Equilíbrio entre o custo de execução e de compilação
- Algumas linguagens de orientação a objetos realizam vínculo estático ou dinâmico a depender de algumas condições específicas

# Tipagem dinâmica

- Muitas vezes, chamadas internas a métodos implementados por subclasses possuem uma informação incompleta
- Em Java, por exemplo, há diversas implementações do método add em classes diferentes.

```
this.add(elemento);
```

# Tipagem dinâmica

- Muito comum para situações de polimorfismo
- Uma referência pode apontar para objetos de classes distintas
- O sistema precisa determinar, em tempo de execução, a qual classe um método se refere, visto que há ambiguidade
- Provê facilidade no desenvolvimento e manutenção dos objetos da aplicação

# Tipagem estática

- Muitas vezes um custo de execução pode ser convertido em compilação
- A informação completa ou restrição do contexto permite essa conversão
- Em Java, por exemplo, quando não há dúvidas da origem de uma chamada de um método, o compilador armazena as informações dos métodos e classes utilizadas

```
colecao.addAll(outraColecao);
```

- A classe precisa implementar `AbstractCollection`, e portanto o método `addAll` deve ser definido como `static` ou `final`

# Sumário

Introdução

Conceitos importantes presentes no paradigma

Herança, encapsulamento e polimorfismo

Tipagem dinâmica e estática

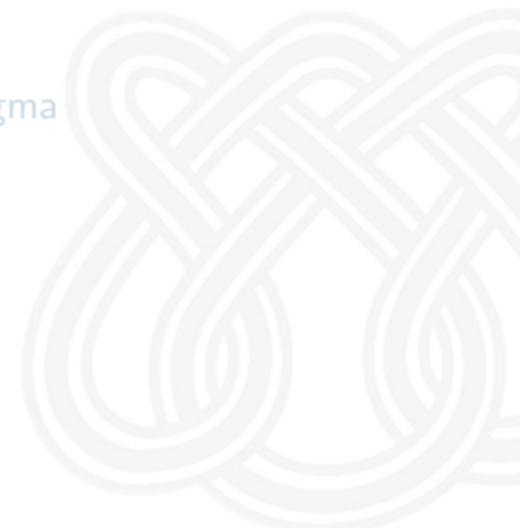
**Exemplos de linguagens**

Smalltalk

C++ e Java

Questões de design e implementação

Referências



# Smalltalk

## Histórico

- Teve inicio com um projeto de pesquisa na Xerox Corporation's Palo Alto Research Center em 1970
- Foi continuado por Adele Goldberg e Daniel Ingalls
- Influência de Simula e Lisp
- Foi inovadora para a divisória criada para interfaces gráficas.
- Ambiente de desenvolvimento completo, com sistemas de janelas, menus e cursor

# Smalltalk

## Design

- Puramente orientada a objetos (até constantes)
- Mensagens e Objetos de dados
- Objetos possuem características e comportamentos

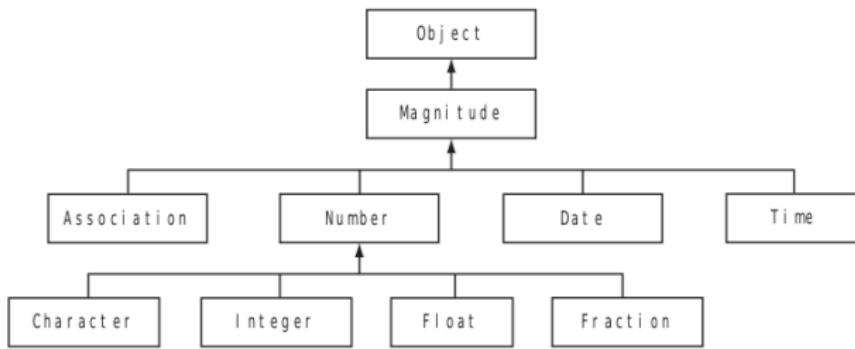
# Smalltalk

## Design

- Características são atributos de uma classe ou instância
- Comportamentos são ações a serem reproduzidas por métodos
- Métodos podem enviar outras mensagens e podem devolver objetos
- Mensagens podem ser unárias ou até mesmo incluir parâmetros

```
Set new includes: 'Hello'  
(Array new: 10) at: 1 put: 66
```

# Smalltalk



**Figure 5.1** Smalltalk's magnitude classes

**Figura: Hierarquia de classes de magnitude [2]**

# C++ e Java

## C++ e Java

- Linguagens muito relevantes atualmente muito importantes ao longo da história e no contexto atual de programação
- Orientação a objetos como um dos principais fatores de popularidade.

# C++ e Java

## Popularidade

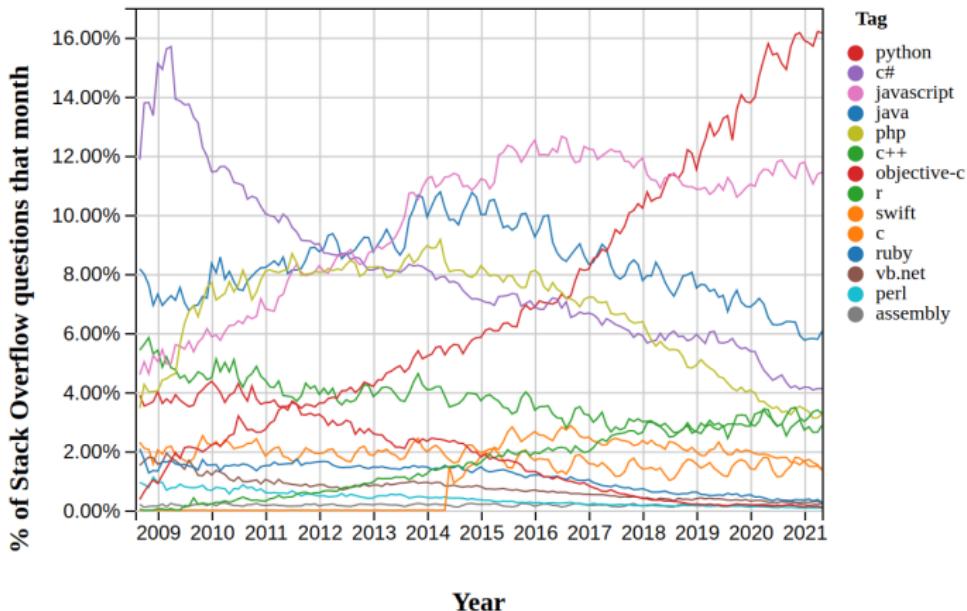


Figura: Popularidade das linguagens de programação.[4]

# C++ e Java

## C++

- Lançada por Bjarne Stroustrup em 1985.
- Planejada para ser um "C com classes".
- Adotou recursos de diversos paradigmas.
- Muito utilizada para contextos com recursos limitados ou que exijam alto desempenho, como software de sistema, embarcados, computação científica, gráficos, jogos, etc.

# C++ e Java

## Java

- Lançada pela Sun Microsystems em 1996.
- Fortemente orientada a objetos.
- Write Once, Run Anywhere (JVM).
- Uma das linguagens mais utilizadas atualmente no contexto de finanças, projetos corporativos, desenvolvimento de apps, etc.

# C++ e Java

## Diferenças

- Ao contrário do C++, Java não suporta sobrecarga de operador ou herança múltipla para classes, embora herança múltipla seja compatível com interfaces.
- Gerenciamento de memória: Manual vs Garbage Colector.[1]
- Passagem de argumentos por ponteiros vs por valor.

# Sumário

Introdução

Conceitos importantes presentes no paradigma

Herança, encapsulamento e polimorfismo

Tipagem dinâmica e estática

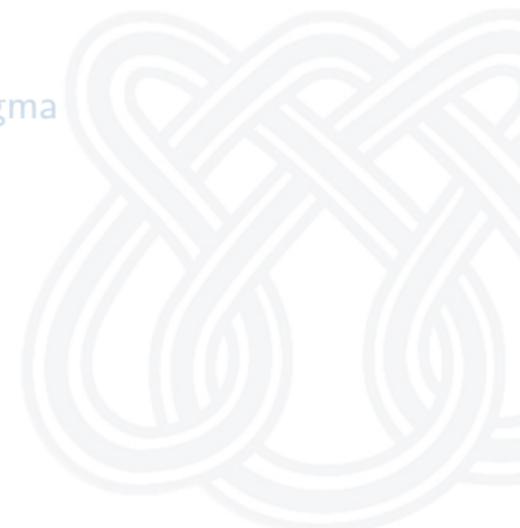
Exemplos de linguagens

Smalltalk

C++ e Java

Questões de design e implementação

Referências



# Questões de projeto

## Design e implementação

- Algumas questões surgem com o paradigma orientado a objetos
- Escolha entre **flexibilidade** e **eficiência de execução**
- Algumas linguagens focam mais em eficiência que outras

# Questões de projeto

## Exclusividade de Objetos

- Introdução do conceito de classes além dos tipos existentes
- **Vantagens:** uniformidade e elegância
- **Desvantagens:** menor eficiência, pois operações simples são feitas por processos de passagem de mensagens

# Questões de projeto

## Exclusividade de Objetos

Escolha entre tipos e classes:

- ① Adição de classes e modelo de objetos à linguagem já existente (Objective-C)
- ② Classes são construtores de tipos, ou seja, declarações de classes fazem parte do sistema de tipagem (C++)
- ③ Exclusão dos outros tipos estruturados, e todos os tipos são classes (Smalltalk)

# Questões de projeto

## Exclusividade de Objetos

Exemplos das linguagens:

- Smalltalk, Ruby: apenas objetos
- C++, Objective-C, Java, C#: tipos primitivos + objetos

# Questões de projeto

## Diferenças entre classes e subclasses

Existem muitas possibilidades de diferenças entre subclasses e classes pai ou base

- Divergir no número de métodos, parâmetros de algum método, corpo de algum método, tipos de retornos dos métodos diferentes...

Normalmente são restringidos:

- A subclasse restringe-se a um subtipo da classe pai - relacionamentos **é-um(a)**
- Métodos da subclasse que sobrescrevem métodos da classe pai devem ser compatíveis com os sobrescritos

# Questões de projeto

## Verificação de tipos e polimorfismo

Variáveis **polimórficas** podem referenciar objetos da classe base ou descendentes

- Por isso, nem sempre podem ser determinadas estaticamente;
- Questão é: sendo dinamicamente, quando realizar as verificações de tipos das vinculações.

# Questões de projeto

## Herança simples e múltipla

Nas heranças múltiplas, uma nova classe herda duas ou mais classes

- Isso pode impactar a linguagem tanto na **complexidade** de programação quanto na **eficiência**.

# Questões de projeto

## Herança simples e múltipla

Exemplo de questão que surge: **heranças compartilhadas**.

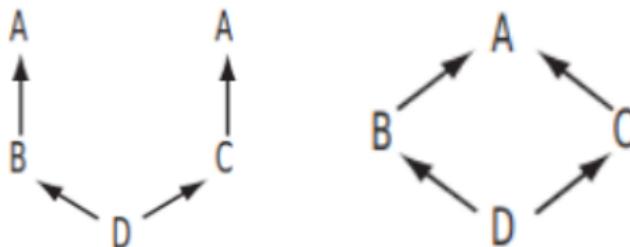


Figura: A mesma classe pai A é referenciada por B e por C. [2]

- Na esquerda: duas cópias da classe A (herança repetida).
- Na direita: a mesma cópia de A usada por B e C (herança compartilhada).

# Questões de projeto

## Herança simples e múltipla

Tipos de herança presentes em algumas linguagens:

- Smalltalk: apenas simples
- Objective-C: apenas simples, alguns efeitos com protocolos
- Java, C#: apenas simples, alguns efeitos com interfaces
- Ruby: apenas simples, alguns efeitos com módulos
- C++: ambas

# Questões de projeto

## Vinculação estática e dinâmica

- Vinculações estáticas ou dinâmicas;
- Estáticas são mais rápidas, mas as dinâmicas também possuem vantagens;
- **Exemplos em linguagens:**
  - Smalltalk, Ruby: todas as vinculações são dinâmicas
  - C++, Objective-C, Java, C#: podem ser estáticas ou dinâmicas

# Questões de projeto

## Alocação e liberação de objetos

- Memória stack ou heap;
  - Em atribuições de objetos de classes iguais ou descendentes, escolher entre cópia de ponteiros ou valores;
  - Alocação e desalocação implícita ou explícita.
- Exemplos:**
- C++: os objetos podem ser estáticos, dinâmicos na stack ou dinâmicos na heap; alocações e desalocações são feitas explicitamente.
  - Java: objetos são dinâmicos na heap; alocação é explícita e desalocação é implícita.

# Questões de projeto

## Classes aninhadas

Caso uma classe seja utilizada apenas por uma outra, pode-se escolher **ocultá-la** das outras. Exemplo:

```
class Arvore {  
    class No {  
        ...  
    };  
  
    std::vector<No> nos;  
    ...  
};
```

- Existem em C++, Java, C#, Ruby, mas não em Smalltalk e Objective-C.

# Questões de projeto

## Inicialização de objetos

- Pode-se optar por inicializar objetos manualmente ou por algum mecanismo implícito;
- Em objetos de subclasse, pode-se associar a inicialização à classe pai ou não
- **Exemplos:**
  - C++, Java, C#, Ruby: construtores são chamados implicitamente;
  - Smalltalk e Objective-C: construtores são chamados explicitamente.

# Sumário

Introdução

Conceitos importantes presentes no paradigma

Herança, encapsulamento e polimorfismo

Tipagem dinâmica e estática

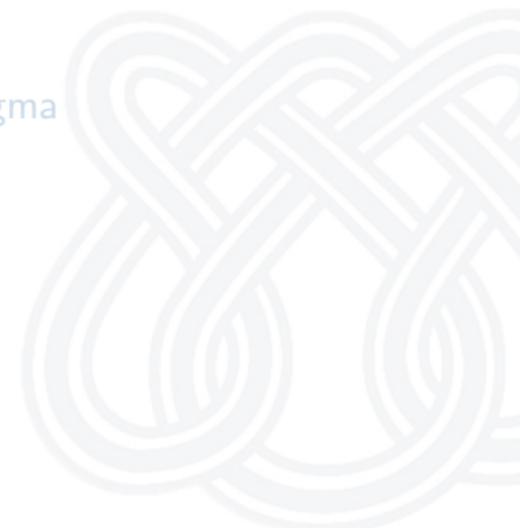
Exemplos de linguagens

Smalltalk

C++ e Java

Questões de design e implementação

Referências



# Referência Bibliográfica I

-  DANIEL GRAZIOTIN, *Object Oriented Memory Management*.  
[https://ineed.coffee/post/  
object-oriented-memory-management](https://ineed.coffee/post/object-oriented-memory-management).  
Online; accessed 30 June 2021.
-  K. C. LOUDEN AND K. A. LAMBERT, *Programming Languages - Principles and Practice*, Course Technology, 3rd ed., 2012.
-  R. W. SEBESTA, *Concepts of Programming Languages*, Pearson, 11th ed., 2016.
-  STACK OVERFLOW, *Stack Overflow Trends*.  
<https://insights.stackoverflow.com/trends>.  
Online; accessed 30 June 2021.