

# **Estrutura de Dados I**

## **Ciência da Computação**

Prof. André Kishimoto  
2024

# Programação

O que significa programar?

# Programação

O que significa programar?

“Ato de criar programas de computador.”

# Programação

O que é um programa de computador?

# Programação

O que é um programa de computador?

“Um conjunto de instruções que define o que o computador deve fazer.”

# Programação

Logo...

Programar significa definir as tarefas que um computador deve realizar.

# Programação

Por que o computador precisa realizar um determinado conjunto de tarefas?

ou...

Por que usamos computadores?

# Programação

- Calculadora
- Editor de textos
- E-mail, IM
- Redes sociais
- E-commerce
- GPS
- Internet banking
- Jogos



# Programação

O que significa programar?

“Programar (computadores) é definir as tarefas que um computador deve realizar para resolver um determinado problema.”

# Estrutura de Dados

Onde entra a nossa disciplina de Estrutura de Dados I?

- Criaremos novos tipos de dados a partir de tipos primitivos.
- Dados estruturados.

Porém...

- Apenas definir novos tipos de dados não serve para muita coisa.
- Precisamos manipular esses dados de alguma maneira.

Assim...

- Estudaremos alguns Tipos Abstratos de Dados (TAD).
- Resumidamente: TAD = dados + operações.

# Estrutura de Dados

Como?

- Prática e implementação com a linguagem Java.
  - Fundamentos da linguagem, evitando funcionalidades avançadas/que fogem do escopo da disciplina.

Referências Java? Sugestões:

- <https://docs.oracle.com/en/java/>
- <https://dev.java/>



# Tipos de dados

Trabalhamos com três principais tipos de dados:

- Tipos primitivos (básicos, fundamentais)
  - Tipos estruturados (agregados, derivados, registros)
  - Tipos abstratos
- 
- No entanto, os tipos estruturados e abstratos acabam usando os tipos primitivos da linguagem.
    - É possível usar tipos estruturados e abstratos aninhados.

# Tipos primitivos

Tipos primitivos são aqueles oferecidos pela linguagem de programação.

Exemplos em Java:

`char`, `boolean`, `int`, `float`, `double`

E suas variantes, dependendo da linguagem. Por exemplo:  
`short`, `long`

# Tipos estruturados

Tipos estruturados são criados a partir da composição de tipos primitivos (e podem conter outros tipos estruturados).

Isto é, ao invés de declararmos N variáveis de tipos primitivos, criamos uma estrutura (um novo tipo de dado) que agrupa os dados e que faça sentido ao problema que vamos resolver.

Exemplo:

Quero um programa que me permita cadastrar as informações básicas e notas das minhas turmas.

# Tipos estruturados

Exemplo:

Quero um programa que me permita cadastrar as informações básicas e notas das minhas turmas.

Como resolver?

- Nesse momento, não vamos nos preocupar com as operações necessárias / possíveis do programa.
- Foco na representação dos dados.

# Tipos estruturados

Primeiro passo:

Entender o problema (“cadastrar as informações básicas e notas das turmas”).

- O que são “informações básicas” e “notas”?



# Tipos estruturados

Primeiro passo:

Entender o problema (“cadastrar as informações básicas e notas das turmas”).

- O que são “informações básicas” e “notas”? Possível solução:
  - Nome completo
  - Matrícula / TIA / RA
  - N1, N2, Sub, PF → N1 e N2 são compostas por outras notas.
  - Disciplina (nome, código, período, professor, ...) → Provavelmente uma outra estrutura que possa ser referenciada pelo código da disciplina.
  - Curso (nome, código, unidade, ...) → Outra estrutura também.

# Tipos estruturados

Para simplificar o exemplo, vamos considerar apenas os seguintes dados para esse problema:

- Nome completo
- Matrícula / TIA / RA
- N1, N2, Sub, PF
  - N1 e N2, nesse caso, não serão compostas por outras notas.
- Quais são os tipos de dados para o exemplo acima?

# Tipos estruturados

Para simplificar o exemplo, vamos considerar apenas os seguintes dados para esse problema:

- Nome completo → string
- Matrícula / TIA / RA → string (int?)
- N1, N2, Sub, PF → float
  - N1 e N2, nesse caso, não serão compostas por outras notas.

# Tipos estruturados

- Nome completo → string
- Matrícula / TIA / RA → string
- N1, N2, Sub, PF → float

Lembrando que String não é um tipo primitivo de Java (é uma classe).

Porém, vamos considerar como sendo um tipo primitivo para esse exemplo.

```
String matricula = "123456789-0";  
String nome = "John Doe";  
float n1 = 0.0f, n2 = 0.0f, sub = 0.0f, pf = 0.0f;
```

```
System.out.println("Matrícula: " + matricula);  
System.out.println("Nome: " + nome);  
System.out.println("N1: " + n1);  
System.out.println("N2: " + n2);  
System.out.println("Sub: " + sub);  
System.out.println("Final: " + pf);
```

(versão usando tipos primitivos e não estruturados)

# Tipos estruturados

Em Java, criamos um novo tipo estruturado com classes:

```
public class Estudante {  
    private String matricula;  
    private String nome;  
    private float n1;  
    private float n2;  
    private float sub;  
    private float pf;  
}
```

Com isso, podemos declarar uma variável do tipo Estudante e criar um objeto (instância da classe) com new:

```
Estudante estudante = new Estudante();  
  
Estudante st;  
st = new Estudante();
```

# Tipos estruturados

- Como atribuir valores e ler os valores das variáveis (atributos) da classe Estudante?
- Duas situações: Código de leitura/escrita dos atributos está...
  - Dentro da classe Estudante → tratamos como variáveis normais.
  - Fora da classe Estudante → precisamos de funções (métodos) públicas na classe, conhecidas como “*getters* e *setters*” (detalhes serão estudados nas próximas aulas; nesse momento, vamos aceitar o conteúdo do próximo slide...).

# Tipos estruturados

```
public class Estudante {  
    private String matricula;  
  
    public String getMatricula() {  
        return matricula;  
    }  
  
    public void setMatricula(String matricula) {  
        this.matricula = matricula;  
    }  
  
    // Outros atributos privados e seus respectivos  
    // métodos getters/setters públicos...  
}
```

# Tipos estruturados

```
public class Estudante {
```

```
    private String
```

```
    public String  
        return matricula  
}
```

```
    public void  
        this.matricula  
}
```

```
// Outros atributos  
// métodos gerais
```

```
}
```

## Observação:

Embora os nomes dos identificadores neste material estejam em português, é recomendável acostumar a usar nomes em inglês.

Embora não seja a escolha natural da maioria das pessoas cuja língua nativa não seja inglês, considere que você trabalhará com pessoas do mundo inteiro e que, ainda hoje, (feliz ou infelizmente) o inglês é a “língua padrão” em Computação.

E se você pretende escrever software de código aberto ou criar uma API (e quer que as pessoas usem sua API), é importante manter código e documentação em inglês.

Além disso, é muito estranho ler códigos do tipo `getMatricula()`, `getDados()`, `setEndereco()` e assim por diante...



# Tipos estruturados

- Com os *getters* e *setters* públicos adicionados na classe, a leitura de um atributo da classe Estudante é feita da seguinte maneira:

```
// Assumindo o objeto st dos slides anteriores...  
String matricula = st.getMatricula();  
System.out.println(matricula);
```

- E alterar o mesmo atributo é feito com o código:

```
st.setMatricula("987654321-0");
```

# Tipos estruturados

Pausa para analisar...

- Como podemos descrever uma estrutura de dados para disciplinas e cursos?

# Tipos abstratos

## Abstração

- Algo que existe como ideia ou conceito, mas talvez não concretizado.
- “Usamos um objeto sem saber o seu mecanismo interno.” (ex. carro, para a população em geral).
- Refinar um sistema complicado para suas partes fundamentais, descrevendo essas partes em uma linguagem simples e precisa.
  - Dar nomes e
  - Explicar suas funcionalidades.

# Tipos abstratos

Ao aplicarmos o conceito de abstração em um tipo de dado, começamos a definir tipos abstratos.

Tipo Abstrato de Dado (TAD) ou Abstract Data Type (ADT).

- Baseado em um tipo de dado,
  - Especifica **o que** cada operação realiza.
  - *Não* especifica *como* a operação realiza sua tarefa.

Em outras palavras... Quais são as funções necessárias para manipular os dados do problema e o que cada função deve fazer?

# Tipos abstratos

Um TAD pode ser visto como um contrato entre três pessoas:

- Projetista define os dados e as operações (TAD).
- Programador(a) codifica as funções que realizam as operações.
- Qualquer pessoa usuária da versão concreta (implementada) do TAD sabe quais funções estão disponíveis para uso e o que fazem.

# Tipos abstratos

Declarações de pré-condição e pós-condição definem o antes e o depois de uma operação.

- Pré-condição:
  - Indica o que deve ser verdadeiro antes de uma operação ser executada.
  - Garantir que a pré-condição seja válida é responsabilidade da pessoa usuária do TAD, mas quem implementa o TAD pode usar programação defensiva e verificar se os valores são válidos.
- Pós-condição:
  - Indica o que será verdadeiro após uma operação ser executada.
  - Quem implementa o TAD deve garantir que a pós-condição seja válida.

# Tipos abstratos

## Principais benefícios dos TADs

- Encapsulamento
  - O código das operações fica “escondido” da pessoa usuária do TAD.
  - Acesso via interface do TAD.
- Abstração
  - Pessoa usuária do TAD não precisa saber como TAD foi implementado.
- Reuso
  - TADs podem ser reusados em diferentes projetos e contextos.

# Tipos abstratos

## Principais benefícios dos TADs

- Manutenção do código
  - Manutenção/melhorias no código interno do TAD, sem afetar usuários.
- Padronização
  - TAD bem projetado segue algum padrão para as operações e a interface.
  - Uso do TAD fica consistente, podendo reduzir erros de programação.
- Segurança
  - Restringir acesso e manipulação direta dos dados.
  - Melhora segurança do código e reduz problemas com os dados.



# Tipos abstratos

Continuando com o nosso exemplo do tipo Estudante, quais operações podem existir para manipular esse novo tipo de dado? Quais são as pré e pós-condições de cada operação?

(Agora sim vamos nos preocupar com as operações necessárias / possíveis do programa.)

# Tipos abstratos

Continuando com o nosso exemplo do tipo *Estudante*, quais operações podem existir para manipular esse novo tipo de dado? Quais são as pré e pós-condições de cada operação?

OPERAÇÃO	COMPORTAMENTO
CriarEstudante(matricula, nome)	<p>Cria um <i>Estudante</i> com matrícula <i>matricula</i> e nome <i>nome</i>.</p> <p>Pré-condição: Não existe um <i>Estudante</i> com <i>matricula</i> e <i>nome</i> informados. Pós-condição: Retorna novo <i>Estudante</i> com <i>matricula</i> e <i>nome</i>.</p>
AtualizarN1(estudante, nota)	<p>Atualiza a nota <i>n1</i> de <i>estudante</i> com <i>nota</i>, que não pode ser negativa.</p> <p>Pré-condição: <i>estudante</i> é válido. Pós-condição: Nota <i>n1</i> de <i>estudante</i> deve estar atualizada com o novo valor.</p>

# Tipos abstratos

Continuando com o nosso exemplo do tipo *Estudante*, quais operações podem existir para manipular esse novo tipo de dado? Quais são as pré e pós-condições de cada operação?

OPERAÇÃO	COMPORTAMENTO
AtualizarN2( <i>estudante</i> , <i>nota</i> )	Atualiza a nota <i>n2</i> de <i>estudante</i> com <i>nota</i> , que não pode ser negativa.  Pré-condição: <i>estudante</i> é válido. Pós-condição: Nota <i>n2</i> de <i>estudante</i> deve estar atualizada com o novo valor.
AtualizarSub( <i>estudante</i> , <i>nota</i> )	Atualiza a nota <i>sub</i> de <i>estudante</i> com <i>nota</i> , que não pode ser negativa.  Pré-condição: <i>estudante</i> é válido. Pós-condição: Nota <i>sub</i> de <i>estudante</i> deve estar atualizada com o novo valor.

# Tipos abstratos

Continuando com o nosso exemplo do tipo *Estudante*, quais operações podem existir para manipular esse novo tipo de dado? Quais são as pré e pós-condições de cada operação?

OPERAÇÃO	COMPORTAMENTO
AtualizarPF( <i>estudante</i> , <i>nota</i> )	Atualiza a nota <i>pf</i> de <i>estudante</i> com <i>nota</i> , que não pode ser negativa.  Pré-condição: <i>estudante</i> é válido. Pós-condição: Nota <i>pf</i> de <i>estudante</i> deve estar atualizada com o novo valor.
Exibir( <i>estudante</i> )	Exibe as informações de <i>estudante</i> na saída padrão do sistema.  Pré-condição: <i>estudante</i> é válido. Pós-condição: Informações de <i>estudante</i> são exibidas na saída padrão do sistema.

# Tipos abstratos

Implementação em Java:

- Vamos aplicar o conceito de modularidade / modularização: os dados e operações relacionados à parte específica do tipo Estudante será colocado em um arquivo `Estudante.java` específico para o tipo.
- O exemplo a seguir usa alguns detalhes da linguagem Java que resultam em uma implementação que não mapeia o TAD descrito nos slides anteriores de forma 1:1.
  - Alguns conceitos serão estudados em aulas futuras...
  - Parte do código está omitido e/ou reduzido.

# Tipos abstratos

```
// Estudante.java
public class Estudante {

    private String matricula;
    // Outros atributos aqui.

    Estudante() { this("?", ""); }

    Estudante(String matricula, String nome) {
        this.matricula = matricula;
        this.nome = nome;
        n1 = 0.0f;
        n2 = 0.0f;
        sub = 0.0f;
        pf = 0.0f;
    }

    void AtualizarN1(float nota) { n1 = nota; }
    void AtualizarN2(float nota) { n2 = nota; }
```

```
void AtualizarSub(float nota) { sub = nota; }
void AtualizarPF(float nota) { pf = nota; }

public String getMatricula() {
    return matricula;
}
public void setMatricula(String matricula) {
    this.matricula = matricula;
}

// Outros getters/setters aqui.

@Override
public String toString() {
    return matricula + "; " + nome + "; " + n1 +
        "; " + n2 + "; " + sub + "; " + pf;
}
}
```

# Tipos abstratos

```
// Program.java
public class Program {

    public static void main(String[] args) {
        Estudante estudante = new Estudante("123456789-0", "John Doe");
        estudante.AtualizarN1(5.0f);
        estudante.AtualizarN2(7.5f);
        estudante.AtualizarSub(2.5f);
        estudante.AtualizarPF(7.8f);
        System.out.println(estudante);
    }
}
```

# TAD e Estrutura de Dados

Uma estrutura de dados é uma maneira de definir como os dados são organizados, armazenados e manipulados por um programa de computador.

TAD = abstração.

Estrutura de dados = implementação concreta de um TAD.

TAD (“o que”) vs. Estrutura de dados (“como”)



# Tipos abstratos

Pausa para analisar...

- Como podemos definir os TADS para disciplinas e cursos?
- Como implementar esses novos TADS em Java?



# CC BY-SA 4.0 DEED

Atribuição-Compartilha Igual 4.0 Internacional

Canonical URL: <https://creativecommons.org/licenses/by-sa/4.0/>

[See the legal code](#)


## Você tem o direito de:


**Compartilhar** — copiar e redistribuir o material em qualquer suporte ou formato para qualquer fim, mesmo que comercial.

**Adaptar** — remixar, transformar, e criar a partir do material para qualquer fim, mesmo que comercial.

O licenciante não pode revogar estes direitos desde que você respeite os termos da licença.

## De acordo com os termos seguintes:

 **Atribuição** — Você deve dar o [crédito apropriado](#), prover um link para a licença e [indicar se mudanças foram feitas](#). Você deve fazê-lo em qualquer circunstância razoável, mas de nenhuma maneira que sugira que o licenciante apoia você ou o seu uso.

 **Compartilha Igual** — Se você remixar, transformar, ou criar a partir do material, tem de distribuir as suas contribuições sob a [mesma licença](#) que o original.

**Sem restrições adicionais** — Você não pode aplicar termos jurídicos ou [medidas de caráter tecnológico](#) que restrinjam legalmente outros de fazerem algo que a licença permita.



# CC BY-SA 4.0 DEED

Attribution-ShareAlike 4.0 International

Canonical URL: <https://creativecommons.org/licenses/by-sa/4.0/>

[See the legal code](#)


## You are free to:


**Share** — copy and redistribute the material in any medium or format for any purpose, even commercially.

**Adapt** — remix, transform, and build upon the material for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

## Under the following terms:

 **Attribution** — You must give [appropriate credit](#), provide a link to the license, and [indicate if changes were made](#). You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

 **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the [same license](#) as the original.

**No additional restrictions** — You may not apply legal terms or [technological measures](#) that legally restrict others from doing anything the license permits.

## Notices: