ESTRUTURA DE DADOS

UNIDADE 1 – ESTRUTURA BÁSICA DE DADOS

Ana Paula dos Santos Braatz Vieira

Introdução

Você saberia explicar o que é uma abstração de dados? Como implementá-la em um projeto? Além disso, como manipular essa abstração e qual é a vantagem da sua utilização na prática?
Ao longo desta primeira unidade, vamos iniciar nosso aprendizado a respeito da estrutura de dados, ou seja, a forma mais adequada para organizar as informações para o computador processá-las com eficiente. Vamos, também, destacar os principais pontos sobre uma estrutura de dados, as vantagens e desvantagens, assim como suas implementações básicas, principalmente no que diz respeito à linguagem de programação Java, bastante utilizada atualmente.
Estudaremos que o algoritmo é uma sequência finita de ações para a obtenção de solução de determinado problema. Ele está relacionado diretamente com a estrutura de dados, pois não conseguimos analisá-la sem considerarmos os algoritmos.
Compreenderemos, ainda, os conceitos de orientação a objetos, como classe, objeto e instância. Por fim, aprenderemos a diferença entre variáveis primitivas e variáveis de referência, assim como as estruturas de dados homogêneas e as heterogêneas, cada uma com suas particularidades.
Assim, para adquirir o conhecimento deste conteúdo e obter as respostas para inúmeros questionamento relacionados à temática, leia o material com bastante atenção. Vamos começar? Acompanhe!

1.1 Conceitos iniciais da estrutura de dados

Nas linguagens de programação, podemos classificar as variáveis, as constantes, as funções e as expressões de acordo com suas características, indicando o tipo de dado. Isso porque os dados fazem parte de grupos de valores, como *int*, *float*, *char* e *boolean*. Nesse contexto, também temos o tipo abstrato de dados que, de acordo com Ziviani (2012), pode ser considerado um modelo matemático, tendo atributos e métodos para se fazer operações, como inserção, remoção e alteração de dados.
Para compreender uma estrutura de dados, ainda é necessário — além dos conceitos de abstração — aprendermos a diferenciar os conceitos relacionados às estruturas de dados estáticas e dinâmicas, bem como as estruturas de dados implícitos e explícitos. Compreender quando um projeto deve ser armazenado em memória interna ou externa é outro ponto muito importante.
Vamos conhecer tudo isso a partir de agora. Vejamos!

1.1.1 Tipos abstratos de dados (TAD)

As estruturas de dados são implementações concretas de tipos abstratos de dados ou, simplesmente, TAD. No entanto, antes de entendermos o que são os tipos abstratos, precisamos analisar o conceito dos tipos de dados.
Um tipo de dado pode ser definido como o conjunto de valores que uma variável pode assumir e determina as operações que podem ser aplicadas à variável, incluindo, também, como devem ser interpretadas. O tipo *int*, por exemplo, é o conjunto dos números inteiros, que possui valores negativos, zero e positivos.
Observe a estrutura a seguir para compreender melhor sobre o que estamos nos referindo. Veja que precisamos declarar uma variável.

```java
public class Imprimi_Soma_Variavel {
    public static void main (String[] args) {
        int x = 15;
        int y = 25;
        int soma = x + y;
        System.out.printf("Soma = " + soma);
    }
}
```

Figura 1 - Declaração de variável
Fonte: Elaborada pela autora, 2019.

Temos, então, que, a declaração de variável seria: <tipo_do_dado> <nome_variável> = <valor>.
Ziviani (2012, p. 2) destaca que um tipo de dado abstrato "[...] pode ser visto como um modelo matemático, acompanhado das operações definidas sobre o modelo", podendo ser representado por um par (*v*, *o*), em que *v* é o conjunto de valores e *o* é o conjunto de operações sobre esses valores. A título de exemplo, podemos citar a variável com o tipo inteiro:
v = **Z**
o = {+, –, , /, =, <, >, <=, >=}
Assim, os tipos de dados abstratos especificam os dados ou o conjunto de valores a serem armazenados e as operações a serem executadas. Isto é, a manipulação dos dados. Tem como objetivo reduzir a informação necessária para a criação de um programa, por meio de uma abstração das variáveis envolvidas. Dessa forma, podemos resumir os dados de uma pessoa com nome, RG e CPF, apenas pelo domínio "pessoa" e suas operações.
Temos, então, que empregar abstrações em um projeto é mais seguro, pois o usuário não tem acesso direto aos dados e consegue apenas manipular as operações. Além disso, o programador é capaz de alterar um tipo de dado abstrato sem modificar suas aplicações.

VOCÊ SABIA?

Um dos quatro pilares da Programação Orientada a Objetos (POO) é a abstração, um dos pontos mais importantes de qualquer linguagem orientada a objetos. Isso porque ela é a representação de um objeto real. No entanto, é válido lembrar que existem alguns pontos que devem ser levados em consideração na hora de se criar uma abstração, como dar uma identidade única dentro do sistema para não existirem conflitos. Depois, damos as características, que são os elementos que a definem, chegando ao último passo, que são as ações ou os métodos.

Agora que já sabemos o que são abstrações, precisamos estudar alguns pontos relevantes que todo programador de Java deve compreender para o efetivo estudo de uma estrutura de dados. Vamos em frente!

1.1.2 Estudo de estrutura de dados

A memória de um computador é composta por uma sequência de *bytes* endereçados. É nela que são alocados os programas a serem executados, os quais, por sua vez, são compostos por códigos (instruções) e dados (variáveis).
Existem memórias que armazenam o conteúdo temporariamente e memórias que o armazenam de forma definitiva. Para aprender mais sobre elas, clique nas abas abaixo.

- **Memórias voláteis**

  As que armazenam os dados de forma temporária são chamadas de **memórias voláteis**, em que o conteúdo armazenado é apagado quando encerramos o computador. Temos como exemplo a memória de acesso aleatório ou, como conhecemos, memória RAM (*Random Access Memory*).

- **Memória não volátil**

  Já a memória que armazena os dados de forma definitiva é chamada de **memória não volátil**, pois não perde os dados quando o computador é desligado, ou seja, não necessita de pulsos elétricos magnéticos para mantê-los.

Temos, então, que as memórias são dispositivos que permitem o armazenamento de informações e programas, podendo ser voláteis (no caso de programas) ou não. Atallah e Blanton (2010) descrevem a existência de dois tipos de memórias para a execução de programas: memória interna (RAM) e memória externa (disco rígido).
A **memória interna** é mais rápida que a externa, porém com capacidade de armazenamento menor. Assim, no caso de estruturas de dados projetadas para trabalhar com grandes quantidades de dados, é necessário utilizar o armazenamento na **memória externa**. Esta, por sua vez, é mais lenta, pois é composta por um meio magnético, ficando ligadas diretamente à CPU e podem ser desconectadas fisicamente do computador sem perder informações, além de possuir um espaço superior ao da memória interna.
Quando executamos um programa, ele é alocado na memória e executado; já quando finalizado, é retirado da memória. A administração dos recursos no uso da memória é realizada pelo Sistema Operacional, que controla as partes que estão disponíveis ou não. Devido a isso, em determinados momentos, existem locais que estão ocupados e outros que estão livres. Dentro desses espaços, podemos armazenar variáveis estáticas e variáveis dinâmicas.
De acordo com Atallah e Blanton (2010), alguns tópicos são relevantes para a compreensão das estruturas de dados, como a diferença de uma estrutura estática e de uma estrutura dinâmica, assim como também temos os dados implícitos e os explícitos.

1.1.3 Estruturas de dados estática, dinâmica, implícita e explícita

Uma **estrutura de dados estática** possui tamanho fixo, porém este é declarado no código, pelo programador. Quando uma variável estática é criada, seu tamanho não pode ser alterado até que o programa seja finalizado, mas ela suporta consultas por meio do uso de índices, como os *arrays*, que podem ser unidimensionais, bidimensionais ou multidimensionais.
Diferentemente da estrutura estática, a **estrutura de dados dinâmica** não tem tamanho fixo, pois seu comprimento evolui conforme a necessidade. No entanto, há um ponto em comum entre as duas estruturas, visto que a dinâmica também suporta consultas e atualizações. Nesse caso, são utilizados ponteiros para se fazer consultas, os quais "apontam" para os dados na memória.
Um exemplo de estrutura dinâmica é a lista, que faz alocação de memória para cada item inserido, sendo realizada em tempo de execução. A limitação da estrutura dinâmica é o limite físico da memória do computador.

VAMOS PRATICAR?

Crie uma aplicação que calcule a área de um triângulo. Elabore as variáveis necessárias e coloque o tipo de dado adequado para cada uma. No final, imprima a resposta. Lembre-se de que a base receberá o valor de 15 e a altura o valor de 7,5. Para calcular a área de um triângulo utilizamos a fórmula: área = $\frac{base \times altura}{2}$.

Existem, ainda, mecanismos de organização de dados. Munro e Suwanda (1980) descrevem que a informação estrutural está implícita na forma como os dados são armazenados, em vez de explícitas, com a utilização de ponteiros.

Um exemplo de **estrutura de dados implícita** são os vetores e a matriz, que requerem apenas a sobrecarga constante do comprimento. Por outro lado, uma **estrutura de dados explícita**, como a lista vinculada, possui um ponteiro que "aponta" para o próximo. É importante mencionar, nesse sentido, que a linguagem Java não possui variáveis de ponteiro, mas variáveis de referência. Todas as variáveis que se referem aos objetos são de referência.

1.2 Tipo de dados abstratos em Java

Como vimos, um tipo de dado abstrato especifica as informações a serem armazenadas e as operações a serem executadas para a manipulação desses dados. Geralmente, utilizamos a abstração para nomear uma classe, podendo ser representada por um substantivo, como pessoa, conta, automóvel... Algo que seja "abstrato".

De acordo com Wirth (1989), podemos dizer que a abstração é uma ferramenta mental importante para lidar com a complexidade. Assim, um problema mais complexo não pode ser visto imediatamente como instruções de computador, mas em termos de entidades naturais ao próprio problema, abstraído de maneira adequada da realidade.

Assim, ao longo deste tópico, vamos conhecer conceitos importantes envolvidos na linguagem Java e na estrutura de dados. Vamos começar!

1.2.1 Classes, objetos e instâncias

A **classe** é uma abstração de um conjunto de objetos com características comuns. Por exemplo, existe uma imensidade de tipos de transportes, como um Fusca, um Gol, um Camaro ou uma Lamborghini. Esses tipos podem ser agrupados em uma classe, denominada "automóvel", pois todos possuem particularidades em comum. Dessa forma, a classe "automóvel" representará qualquer tipo de transporte com as características em comum de desejamos armazenar.

VOCÊ O CONHECE?

James Gosling é um programador que ficou conhecido como o "pai" da linguagem de programação Java. James, na década de 1990, chefiava uma equipe de programadores que desenvolveu a linguagem Java, linguagem compilada para um *bytecode* que é interpretado por uma máquina virtual, o *Java Virtual Machine* (JVM). James também programou uma versão do editor de texto "Emacs" do sistema Unix, usado por programadores e usuários que necessitam desenvolver documentos técnicos. Antes de se juntar à Sun Microsystems, construiu uma versão do Unix para máquinas multiprocessadas, assim como compiladores e sistemas de mensagens eletrônicas (SCHOFIELD, 2007).

As características de uma classe podem ser dadas por meio de atributos, que são os dados que desejamos armazenar. Por exemplo, na classe "automóvel", podemos armazenar os atributos cor, marca, ano, velocidade, preço e modelo, conforme vemos no esquema retratado na figura a seguir.

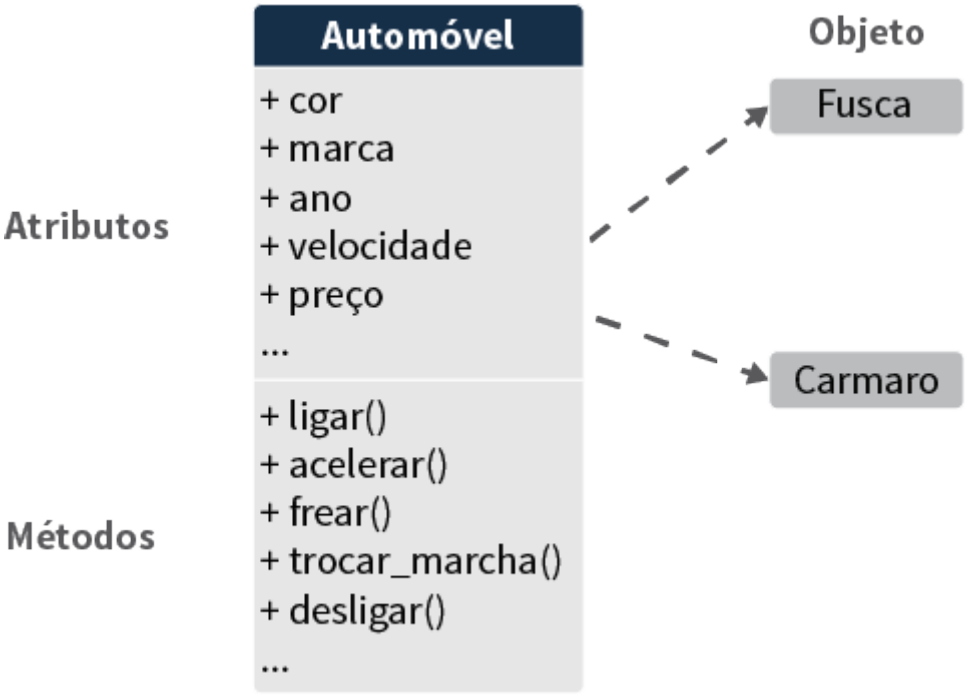

Figura 2 - Exemplificação da classe "automóvel" com seus atributos, métodos e objetos
Fonte: Elaborada pela autora, 2019.

Dessa forma, uma classe é um modelo para a criação de objetos, que terá as mesmas características da classe à qual pertence. É na classe que se define o tipo do objeto, especificando os atributos e métodos. Clique nas abas, para conhecer mais sobre o tema.

Os **métodos** são os comportamentos ou as funções que a classe pode ter. É, portanto, a maneira de fazer a operação com o objeto. Quando aceleramos um automóvel, sua

Métodos	velocidade aumenta, não é mesmo? O fato de acelerar é a função, ou seja, o método da classe.
Objeto	O **objeto**, por sua vez, é uma instância de uma classe que também possui variáveis e métodos. A classe "automóvel" é formada pelos objetos Fusca e Camaro, por exemplo. Assim, todos os objetos têm em comum o fato de serem automóveis, mas podem ter características diferentes entre si, como o preço, pois o valor de um Fusca é diferente do valor de um Camaro; assim como a velocidade que Fusca pode chegar, que também é diferente da velocidade limite de um Camaro.
Instanciar	Quando desejamos criar mais um objeto, é necessário **instanciar**. Assim, a instância de uma classe é um novo objeto, sendo que uma classe somente poderá ser utilizada após ser instanciada, ou seja, chamada.

VOCÊ QUER VER?

Existem variáveis de instância e variáveis de classes. Uma classe pode conter campos que são declarados fora dos métodos (variáveis de instância), usados pelos objetos para armazenarem seus dados. Temos, então, que uma variável de classe é o campo que pertence à classe em si e que é compartilhado por todos os objetos da classe,ou seja, independe da instância. Para saber mais sobre o assunto, assista ao vídeo "Curso de Java – Variáveis de classe e variáveis de instância – Orientação a Objetos", de Fábio dos Reis, disponível no *link*: https://youtu.be/H57gWJsfU3I?list=TLGGUklh0O25d-kxMjA3MjAxOQ&t=152.

Agora que já aprendemos o que é classe, objeto e instância, podemos prosseguir e aprender mais sobre as variáveis, que são os atributos que desejamos armazenar.

1.2.2 Variáveis primitivas vs variáveis de referência

De acordo com Deitel e Deitel (2010), o Java divide as variáveis em primitivas e de referência. Como primitivas, podemos destacar *int*, *float*, *double*, *char*, *short*, *long*, *boolean* e *byte*. Com base nisso, temos que todas as variáveis não primitivas são de referência, como as classes, que especificam o tipo do objeto. No entanto, é válido citar que tipos de dados não primitivos são aqueles construídos a partir de tipos primitivos. Os principais tipos não primitivos de dados são *array* (vetor e matriz), cadeia de caracteres, tabelas e listas (BIANCHI; FREITAS; JÚNIOR, 2014).

Em uma variável do tipo primitivo, só podemos armazenar um valor de seu tipo por vez. Caso queiramos atribuir outro valor, o que já existe será substituído pelo valor novo. Além disso, as variáveis de instância de tipo primitivo são iniciadas por padrão, ou seja, como 0, exceto o tipo primitivo *boolean*, que é inicializado como *false*.

No exemplo a seguir, temos uma declaração em Java dos tipos primitivos.

```java
public class Tipos_Primitivos {
    public static void main (String[] args) {
        byte tipoByte = 123;
        short tipoShort = 56325;
        char tipoChar = 'A';
        float tipoFloat = 10.6f;
        double tipoDouble = 5.63;
        int tipoInt = 25;
        long tipoLong = 1524689657423154558L;
        boolean tipoBooleano = false;
    }
}
```

Figura 3 - Exemplo de uma declaração em Java dos tipos primitivos
Fonte: Elaborada pela autora, 2019.

Uma variável de referência, por outro lado, armazena as localizações do objeto na memória do computador e é iniciada pelo valor *null*. Os objetos que são referenciados podem conter várias variáveis de instância e métodos. Assim, para conseguirmos acessar um objeto e seus métodos de instância, é preciso ter referência a algum objeto.

1.2.3 Declaração de variáveis

Como sabemos, as variáveis representam uma forma do programador identificar os dados por meio de um nome simbólico, a fim de recuperarem os dados que estão armazenados na memória. Dessa forma, uma variável deve estar associada a um tipo de dado primitivo ou de referência (classe), definidos pelo programador.
Forbellone e Eberspächer (2005) descrevem a memória como um armário, em que existem várias gavetas (variáveis) que precisam ser diferenciadas por meio de identificadores. Cada gaveta pode guardar apenas um item (dado) de cada vez, sendo que cada item possui um tipo primitivo.

CASO

Imagine que desejamos armazenar os dados dos alunos para um programa de uma escola. Para implementá-lo, devemos escolher um termo abstrato que represente todas as características e funções básicas de um aluno, ou seja, a classe. Vamos chamar nossa classe de "aluno", mas também poderíamos optar por "estudante" ou outro termo que encaixe no contexto. As características ou os atributos que podemos armazenar na classe são dados como o nome, o telefone para contato, o endereço, o número da matrícula, as disciplinas, entre outras informações.

Além dos atributos, a classe "aluno" também conta com funções e, no caso, as funções poderiam ser: cursar a disciplina, assistir às aulas, fazer uma prova, fazer um exercício, apresentar um seminário, entre outras. Assim, cada aluno que for cadastrado terá todas as características e funções que a classe possui.

Vamos considerar um projeto no qual precisamos armazenar informações sobre automóveis. "Automóvel" é o termo abstrato que utilizaremos para guardar os dados dos futuros veículos automotores que serão inseridos no programa. Para iniciar o projeto, devemos responder as seguintes perguntas: quais são as variáveis que devemos armazenar no projeto de automóveis? Quais são as funções básicas de um automóvel? Como conseguiremos acessar as informações desse projeto?

Os atributos relevantes para um automóvel são cor, marca, ano, modelo, velocidade, preço etc. As funções básicas ou operações, por sua vez, são o ligar, o acelerar, o frear, o trocar a marcha e o desligar. Assim, para acessar as informações de um projeto, devemos usar as variáveis de referência, que são todas aquelas que se referem aos objetos.

Vamos trabalhar, aqui, com a classe "automóvel". Com as informações dos atributos e métodos, podemos começar a criar um projeto para cadastro de automóveis. Para criar uma classe, devemos, primeiramente, declarar suas variáveis:

```
1  class Automovel {
2      String cor;
3      String marca;
4      Date ano;
5      … // atributos
6  }
```

Figura 4 - Declaração da variável do exemplo
Fonte: Elaborada pela autora, 2019.

Agora que já temos uma classe em Java, que especifica todas as características que um objeto deve conter, vamos criar um objeto. Para isso, teremos que utilizar o "Projeto.java" para instanciar o objeto, utilizando a palavra "*new* ".

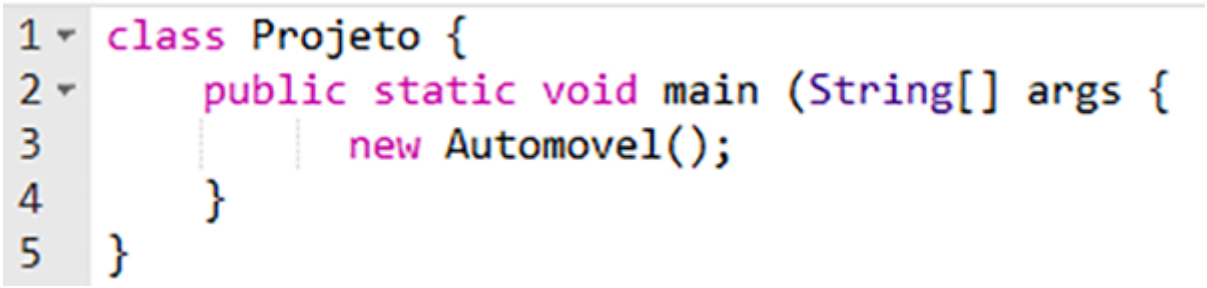

Figura 5 - Projeto.java para instanciar objeto
Fonte: Elaborada pela autora, 2019.

O código anterior cria um objeto de classe "automóvel", porém, para acessá-lo, é necessário referenciar o objeto, por isso, precisamos de uma variável.

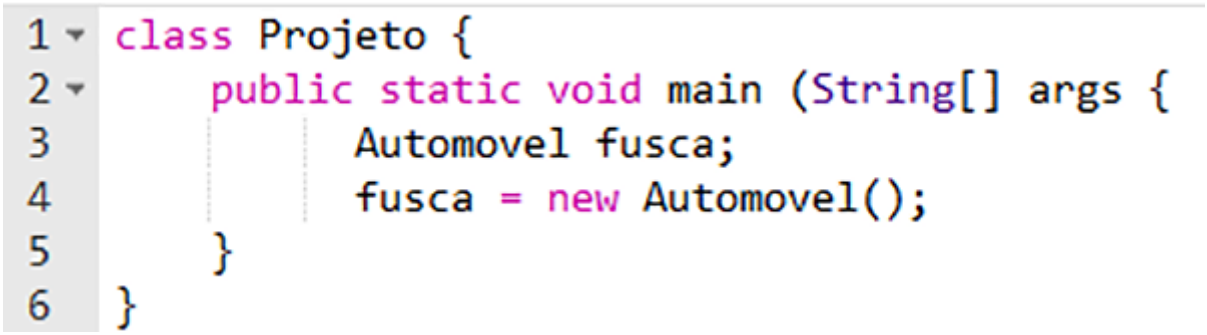

Figura 6 - Referenciando objeto por meio de uma variável
Fonte: Elaborada pela autora, 2019.

Por meio do objeto "fusca", podemos acessar e alterar todas as variáveis que foram declaradas na classe "automóvel". Lembre-se de que o objeto é uma instância de uma classe e possui variáveis e métodos da classe.

```
1  class Projeto {
2      public static void main (String[] args {
3          Automovel fusca;
4          fusca = new Automovel();
5
6          fusca.cor = “branco”;
7          fusca.modelo = “Volkswagen”;
8
9          System.out.println(“Cor do Automóvel: “ + fusca.cor);
10     }
11 }
```

Figura 7 - Alteração das variáveis
Fonte: Elaborada pela autora, 2019.

Na classe também declaramos os métodos, que são as funções ou os comportamentos da classe, como a função liga ().

```
1  class Automovel {
2      String cor;
3      String marca;
4      Date ano;
5
6      //demais atributos
7
8     void liga () {
9         System.out.println("O    automóvel    está    ligado");
10    }
11 }
```

Figura 8 - Declaração dos métodos na classe
Fonte: Elaborada pela autora, 2019.

Temos, com essas construções, os códigos para a criação de classe, seus atributos e métodos. Além disso, também instanciamos o objeto da classe.

VAMOS PRATICAR?

Crie, na linguagem Java, uma classe com o nome “Gato” e descreva pelo menos cinco atributos e quatro métodos que a classe pode possuir. Depois, crie os objetos “Persa”, “Siamês” e “Ragdool” para a classe em questão.

Na sequência, vamos aprender o que é a estrutura de dados e a diferença entre as estruturas homogêneas e as heterogêneas. Vejamos!

1.3 Estruturas de dados

As estruturas de dados envolvem algoritmos que trabalham com dados organizados na memória, de modo que possam ser utilizados de forma mais eficiente. Existem várias estruturas de dados, sendo que, com essas estruturas, podemos administrar uma grande quantidade de informações e utilizar em aplicações com banco de dados ou serviços de busca e indexação.

A maioria das estruturas de dados podem ser descritas como se fossem contêineres que armazenam uma coleção de objetos de determinado tipo, ou seja, os elementos dos contêineres. Assumimos que estes podem ser acessados por meio de variáveis chamadas "localizadores". Quando um objeto é inserido no contêiner, um localizador é retornado e pode ser utilizado para manipular os objetos. O localizador, geralmente, é implementado como um ponteiro ou o índice de um vetor ou matriz, ou seja, *array*.

Em uma estrutura de dados, precisamos saber como realizar algumas operações básicas, como inserir um novo item, excluir um item existente, localizar um dado específico e ordenar as informações. Além disso, quando vamos utilizar as estruturas de dados, temos que analisar critérios que podem incluir eficiência para buscas e padrões específicos de acesso, como manipular um grande volume de dados, análise de tempo, custo computacional, entre outras formas.

Assim, estruturas de dados eficientes são importantes para a elaboração de algoritmos, uma vez que muitas linguagens, inclusive o Java, possuem ênfase nas estruturas de dados, como evidenciado pela Programação Orientada a Objetos, que trabalha com abstrações.

Dessa forma, neste tópico, vamos entender a diferença entre uma estrutura de dados homogênea e uma heterogênea, bem como outros pontos importantes. Vamos aos estudos? Acompanhe!

1.3.1 Estrutura de dados homogêneas

Conforme nos explicam Forbellone e Eberspächer (2005, p. 69) que, "[...] quando determinada estrutura de dados é composta por variáveis com o mesmo tipo primitivo, temos um conjunto homogêneo de dados". Portanto, em uma analogia, podemos considerar que uma variável composta homogênea é como um cardume, em que os elementos (variáveis) são todos peixes (mesma espécie).

Dentro de uma estrutura de dados homogênea, existem variáveis compostas unidimensionais e multidimensionais.

As **estruturas unidimensionais homogêneas** são estruturas compostas por uma dimensão chamada "vetor", sendo este uma das estruturas de dados mais simples. De acordo com Tamassia e Goodrich (2006), trata-se de uma variável que armazena diversas variáveis do mesmo tipo, em posições consecutivas, reservando uma posição na memória (índice) em algum endereço da RAM (conforme o modelo de Von Neumann), auxiliando na manipulação dos dados.

VOCÊ QUER VER?

O modelo de Von Neumann, também chamada de Arquitetura de Von Neumann, é uma arquitetura de computador, ou seja, um modelo conceitual que mostra como funciona um computador, caracterizada pela possibilidade de uma máquina armazenar os programas no mesmo espaço de memória que os dados, podendo manipulá-los. Esse modelo é a base de praticamente todas as máquinas atuais. Ele é baseado na explicação do físico John Von Neumann. Para saber mais sobre esse modelo, indicados assistir ao vídeo intitulado "Arquitetura de Von Neumann de maneira descomplicada", disponível em: https://www.youtube.com/watch?v=V5qE-u6jGo4.

Os vetores são estruturas que permitem o acesso a uma grande quantidade de dados em memória, usando apenas um nome de variável do vetor. Um vetor também pode ser chamado de matriz de uma dimensão.
Para calcular a média de 100 valores, por exemplo, criamos um vetor com 100 posições (de 0 a 99). Para percorrer o vetor, criamos um laço de repetição, que faz a soma de todos os números e, depois, faz o cálculo da média. Esse laço de repetição que utilizamos para manipular o vetor implica em ganho de tempo e diminui muitas linhas de código, pois o programador não precisa criar 100 variáveis.
Veja um vetor retratado na figura a seguir.

Figura 9 - Representação de um vetor, iniciado na posição 0
Fonte: Elaborada pela autora, 2019.

A figura apresenta um vetor de *n* posições. Note que ele é iniciado na posição 0. O vetor em Java é iniciado em 0 e seu valor vai crescendo até a última posição do vetor. Tamassia e Goodrich (2006, p. 50) descrevem que "[...] essa forma de organização é extremamente útil, na medida em que permite computações interessantes", pois podemos percorrer se forma mais ágil um vetor por meio dos laços de repetição. Por ser uma estrutura de dados unidimensional, ela "cresce" em apenas uma direção.
Imagine que temos um vetor com 50 posições e só foram preenchidas as 10 primeiras. Dessa forma, precisamos imprimir os dados na ordem inversa. O que ocorre com os locais que reservamos no vetor que não foram preenchidos? Eles contêm valores indefinidos que já estavam na memória quando o programa começou a ser executado, por isso, esses valores são chamados de "lixo de memória".
As **estruturas multidimensionais homogêneas**, por outro lado, são estruturas compostas por duas ou mais dimensões chamadas de "matriz", que pode ser considerado um vetor de vetores. A matriz, assim como o vetor, também é uma estrutura de dados simples, sendo uma coleção de variáveis do mesmo tipo, em que é reservado uma posição no endereço de memória da memória principal.
A matriz é um *array* bidimensional ou até um *n*-dimensional, sendo que cada posição possui um índice, composto por linhas e colunas para ser acessado, conforme podemos ver na figura a seguir.

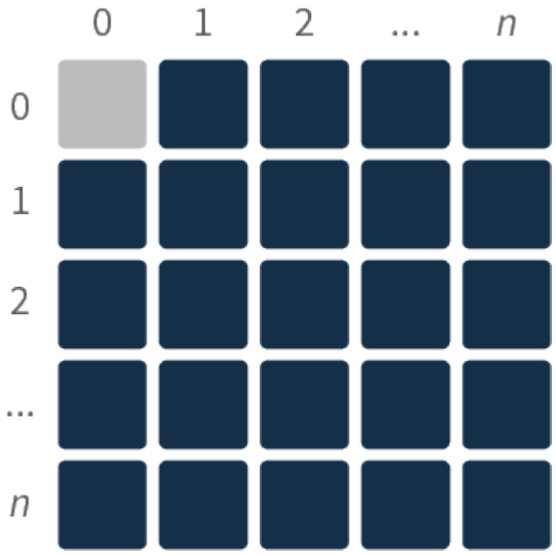

Figura 10 - Representação da matriz, composta por linhas e colunas
Fonte: Elaborada pela autora, 2019.

Na figura anterior, podemos perceber que a matriz possui linhas e colunas que variam de 0 até *n*. Quando queremos especificar um elemento da matriz, devemos especificar, também, em qual linha e qual coluna o elemento está situado. Por exemplo, o elemento destacado em preto na figura pode ser representado por [0,0]. Primeiramente, fazemos o cruzamento da posição da linha e, em seguida, da posição da coluna.

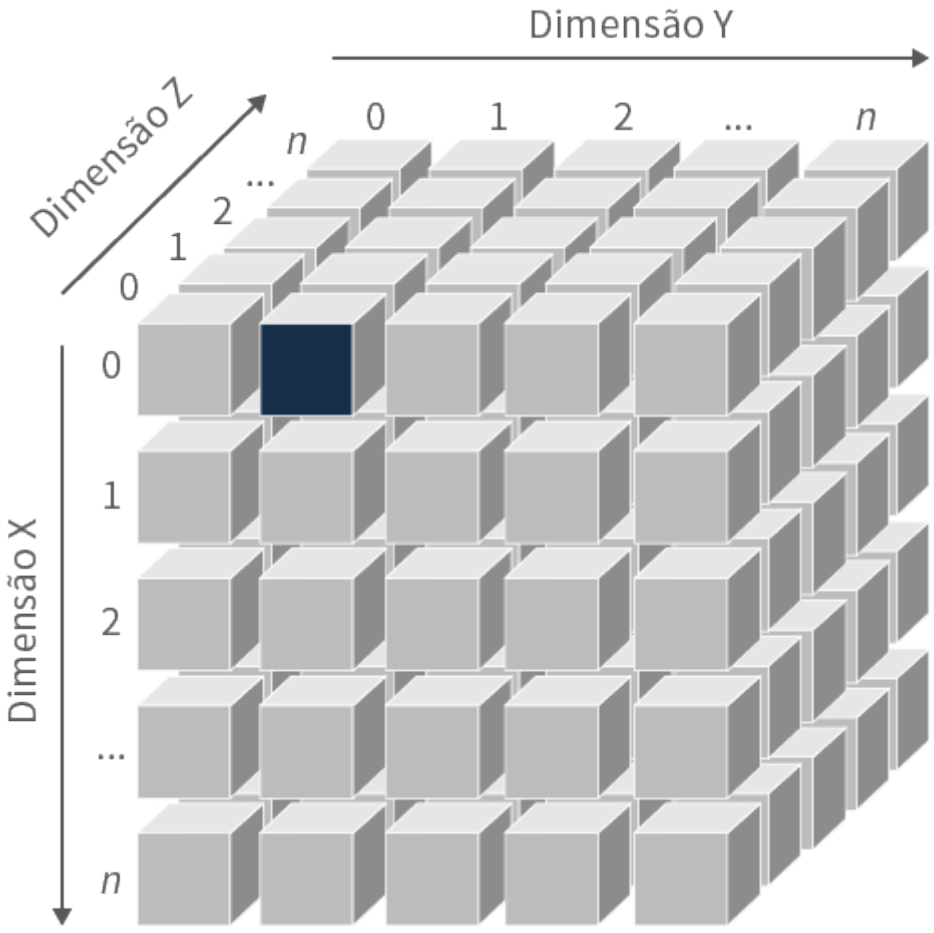

Figura 11 - Matriz tridimensional, com eixos x, y e z
Fonte: Elaborada pela autora, 2019.

Caso a matriz possua mais dimensões, como na figura anterior, deve-se fazer o cruzamento de todas as dimensões por meio dos índices. Por exemplo, no plano tridimensional da figura, temos as dimensões *x*, *y* e *z*. Quando desejamos especificar um elemento da matriz, devemos fazer os cruzamentos dos eixos [x, y, z]. O elemento destacado é [0,1,0], em que x = 0, y = 1 e z = 0.

1.3.2 Estruturas de dados heterogêneas

Uma estrutura de dados heterogênea é quando a estrutura é composta por variáveis com diferentes tipos primitivos. Podemos fazer uma analogia das variáveis compostas heterogêneas como um grupo de animais quadrúpedes, que pode ser formado por cães (matilha), bois (boiada ou manada), ovelhas (rebanho) etc.
Assim, quando possuímos uma estrutura de dados composta heterogênea, temos um registro ou *struct*, ou seja, é uma estrutura que oferece um formato especializado para armazenar informações em memória, podendo armazenar elementos de diferentes tipos, variando de acordo com a necessidade.
O registro é composto por campos que especificam cada uma das informações. Podemos considerar um cadastro de clientes como um registro, pois possui campos com informações como nome, idade, RG, CPF e endereço. Cada campo pode ter um tipo de dado diferente, como *char*, *float*, *double*, entre outros.
Considerando isso, você consegue ver alguma semelhança do conceito de registro com algum outro que já conhecemos? A classe é um conceito muito parecido, pois ela também pode armazenar variáveis de diferentes tipos, sendo que os objetos herdam suas variáveis e seus métodos.

VOCÊ QUER LER?

É importante ressaltar que, em Java, não existem exatamente registros, mas classes, que são recursos muito parecidos. A classe é um dos principais conceitos de Orientação a Objetos (OO), que, além de inserir dados de diferentes tipos, também insere métodos. Além disso, você sabia que o tipo *string* é uma classe? Trata-se de uma cadeia de caracteres. Para se aprofundar no assunto, sugerimos a leitura do artigo "Java *string*: manipulando métodos da classe *string*", disponível por meio do *link*: https://www.devmedia.com.br/java-string-manipulando-metodos-da-classe-string/29862.

Agora que compreendemos a diferente entre os tipos de estruturas de dados, na sequência, vamos aprender como declaramos as estruturas de dados com a linguagem de programação Java. Vejamos!

1.3.3 Declaração de estruturas de dados em Java

Como vimos, um vetor e uma matriz podem ser uma estrutura de dados. Eles são estruturas homogêneas, nos quais só podemos armazenar dados de mesmo tipo.
Para declararmos um **vetor**, devemos ter em mãos algumas informações importantes, como nome do vetor, quantidade de posições (seu tamanho) e, por último, o tipo de dados que será armazenado no vetor.
Veja a seguir a declaração de um vetor do tipo inteiro em Java. Note que a declaração de um vetor é similar à de variáveis, sendo que o que diferenciam as duas é o uso dos colchetes [ ].

int vetor[];

No entanto, apenas com a declaração anterior o vetor não está pronto para ser utilizado, pois precisamos reservar um espaço para alocar os elementos que serão armazenados na memória, ou seja, o tamanho do vetor. Dessa forma, temos que:

vetor = new int[5];

Podemos, ainda, combinar as declarações de criação de um vetor e a declaração de alocação de espaço em um vetor, formando uma mais compacta, conforme podemos observar na sequência:

int vetor[] = new int[5];

Para armazenar elementos no vetor, é necessário o uso de um índice que indique em qual posição desejamos armazenar o elemento. No exemplo a seguir, armazenamos o elemento 10 na posição 2 do vetor.

vetor[2] = 10;

Lembre-se de que a posição inicial do vetor é 0, por isso, temos um vetor com cinco posições, que vai de 0 até 4. Caso haja uma tentativa de armazenar um elemento em uma posição que não existe, ocorrerá um erro no programa que impedirá sua execução. Veja a figura a seguir.

Figura 12 - Índices de um vetor de cinco posições
Fonte: Elaborada pela autora, 2019.

Para percorrer um vetor, precisamos do auxílio do laço de repetição (*for*, por exemplo), para não ser necessário digitar elemento por elemento.

Observe a figura a seguir, em que temos a inserção de dados, a impressão dos dados e a soma dos elementos armazenados. A parte reta é o *output* do console.

```
1  public class Criando_Vetor {
2      public static void main(String[] args) {
3          int soma = 0;
4          int vetor[] = new int[5];
5
6          for (int i = 0; i < 5; i++) { //inserindo dados no vetor
7              vetor[i] = i * 5;
8          }
9
10         for (int i = 0; i < 5; i++) { //percorrendo o vetor e mostrando os valores
11             System.out.println(vetor[i]);
12         }
13
14         for (int i = 0; i < 5; i++) { //percorrendo para calcular a soma de todos o valores do vetor
15             soma = soma + vetor[i];
16         }
17        System.out.println("Soma dos numeros armazenados no vetor: "+ soma);
18     }
19 }
20
```

```
0
5
10
15
20
Soma dos numeros armazenados no vetor: 50
```

Figura 13 - Código em Java com criação de um vetor
Fonte: Elaborada pela autora, 2019.

Na figura anterior, temos a classe “criando vetor”, que está criando um vetor de tamanho 5. No primeiro laço de repetição *for*, o programa armazena números inteiros múltiplos de 5: [0,5,10,15,20]. No segundo laço, ele percorre todo o vetor e imprime o resultado. No último laço de repetição, é realizada a soma de todos os elementos do vetor, sendo armazenados na variável “soma”.

Quando alocamos um espaço na memória para o vetor, ele não pode ser alterado. Caso precise de mais de espaço, será necessário criar um novo *array*, ou seja, um novo vetor com um tamanho maior, em que podemos copiar os elementos do vetor anterior.

Uma **matriz**, por sua vez, pode ser considerada um vetor de vetores, que são referenciadas por um nome, sendo que seus elementos são identificados por índices. Contudo, a linguagem Java não suporta *arrays* multidimensionais de forma direta, mas permite que o programador especifique os *array* unidimensionais, cujos elementos também são *arrays* unidimensionais, ou seja, um vetor de vetores (*array* bidimensional).

Em Java os *array* são considerados um tipo por referência, sendo uma referência a um objeto *array* na memória.

Para declarar um *array* bidimensional, devem ser descritos os números de linhas e colunas, bem como o tipo de dados (a matriz é uma estrutura de dados homogênea). Sua declaração é muito parecida com o *array* unidimensional (vetor).

int m[][] = new int[2][3];

Podemos criar um *array* bidimensional e iniciá-lo com valores:

int m[][] = {{1, 2, 3}, {4, 5, 6} };

VAMOS PRATICAR?

Crie um vetor (vet_1) de inteiros com tamanho 100, insira os números de 1 até 100 em ordem crescente e, depois, crie outro vetor (vet_2) com o valor inverso do primeiro vetor. Depois de inserir, crie um terceiro vetor (vet_3) que armazena a média dos vetores vet.-1 e vet_2. No final, mostre os valores dos vetores 1, 2 e 3.

No código a seguir, estamos percorrendo um *array* bidimensional e imprimindo os valores por linha. A parte reta é o *output* do console.

```
1  public class Criando_Vetor_Bidimensional {
2      public static void main(String[] args) {
3          int m[][] = { {1, 2, 3}, {4, 5, 6}};
4
5          for (int i = 0; i < 2; i++){
6              System.out.printf("%da. linha: ", (i+1));
7
8              for (int j=0; j < 3; j++) {
9                  System.out.printf("%d ", m[i][j]);
10             }
11         System.out.printf("\n");
12         }
13     }
14 }
15
```

```
1a. linha: 1 2 3
2a. linha: 4 5 6
```

Figura 14 - Percorrendo e imprimindo dados em um array bidimensional

Fonte: Elaborada pela autora, 2019.

Assim, aprendemos como inserir e percorrer *arrays* unidimensionais e *arrays* bidimensionais. Esse conhecimento é fundamental para programadores.

Síntese

Chegamos ao fim da primeira unidade da disciplina de Estrutura de Dados. Aqui, pudemos conhecer conceitos sobre o que é uma abstração, como funciona a linguagem Java e a Orientação a Objetos. Também pudemos estudar a respeito das estruturas de dados e suas relações, bem como entendemos um pouco sobre o uso da memória.

Nesta unidade, você teve a oportunidade de:

- compreender os conceitos de classe, objeto e instância;
- entender as diferenças entre variáveis primitivas e variáveis de referência;
- compreender o que são e quais as diferenças entre estruturas de dados homogêneas e heterogêneas, assim como suas particularidades;
- aprender a declarar variáveis em Java;
- compreender como declarar estruturas de dados básicos em Java.

Bibliografia

ATALLAH, M. J.; BLANTON, M. *Algorithms and Theory of Computation Handbook Second Edition General Concepts and Techniques*. **CRC press**, 2010.

BIANCHI, F.; FREITAS, R.; JUNIOR, D. **Estrutura de dados e técnicas de programação**. Amsterdã: Elsevier, 2014.

BÓSON TREINAMENTOS. **Curso de Java – Variáveis de classe e variáveis de instância – Orientação a Objetos** . 12 jul. 2019. Disponível em: https://www.youtube.com/watch?v=H57gWJsfU3I&feature=youtu.be&list=TLGGUklh0O25d-kxMjA3MjAxOQ&t=152. Acesso em: 22 jul. 2019.

CANAL TI. **Arquitetura de Von Neumann de maneira descomplicada**. 14 dez. 2017. Disponível em: https://www.youtube.com/watch?v=V5qE-u6jGo4. Acesso em: 22 jul. 2019.

DEITEL, P.; DEITEL, H. **Java: como programar.** 8. ed. São Paulo: Pearson Education do Brasil, 2010.

DEVMEDIA. **Javas *string***: manipulando métodos da classe *string*. Rio de Janeiro, 2013. Disponível em: https://www.devmedia.com.br/java-string-manipulando-metodos-da-classe-string/29862. Acesso em: 22 jul. 2019.

FORBELLONE, A. L. V.; EBERSPACHER, H. F. **Lógica de programação**: a construção de algoritmos e estruturas de dados. 3. ed. São Paulo: Pearson, 2005.

MUNRO, J. I.; SUWANDA, H. *Implicit data structures for fast search and update*. **Journal of Computer and System Sciences**, v. 21, n. 2, p. 236-250, 1980.

SCHOFIELD, J. James Gosling, o pai do Java. **Gazeta do Povo**, 16 abr. 2007. Disponível em: https://www.gazetadopovo.com.br/economia/james-gosling-o-pai-do-java-afz7nojdkx57547vodfwa6zim/. Acesso em: 26 jul. 2019.

TAMASSIA, R.; GOODRICH, M. T. **Estrutura de dados e algoritmos em Java**. Porto Alegre: Bookman, 2006.

WIRTH, N. **Algoritmos e estruturas de dados**. Rio de Janeiro: Prentice-Hall, 1989.

ZIVIANI, N. **Projeto de algoritmos**: com implementações em JAVA e C++. São Paulo: Cengage Learning, 2012.