# 5 Estruturas de Dados Compostas

*Onde é mostrado como podem ser definidos novos tipos de dados para representar valores compostos.*

| Pré-requisito(s): | Objetivos (ao final você deverá ser capaz de): |
|---|---|
| • Conhecer os tipos de dados primitivos<br>• Saber declarar variáveis com esses tipos<br>• Saber criar algoritmos com estruturas de controle distintas<br>• Saber como utilizar as estruturas de controle de Java | • Identificar que tipo de dado é mais apropriado para representar os dados manipulados em um algoritmo<br>• Identificar quando um problema necessita de um tipo de dado composto<br>• Declarar e manipular vetores e matrizes<br>• Declarar e manipular registros<br>• Declarar variáveis compostas por outras variáveis compostas<br>• Utilizar estruturas de dados compostas em Java |

No capítulo 3, vimos o conceito de variável e como ser possível atribuir um nome a um endereço de memória. Utilizamos esse conceito de variável para construir vários algoritmos que fazem uso de leitura e escrita de dados, atribuição de valores e, no capítulo 4, diferentes controles de fluxo de execução. No momento, sabemos reconhecer que para o armazenamento na memória do nome de uma pessoa, precisamos declarar uma variável do tipo literal, enquanto que para armazenar sua idade, precisamos declarar uma variável do tipo inteiro, por exemplo. O que não sabemos ainda é como manter armazenado na memória dados sobre uma lista de 1000 (mil) funcionários de uma empresa, cada um composto por um conjunto de 10 (dez) tipos de dados pessoais diferentes, sem ter que utilizar 10000 (dez mil) identificadores diferentes. Este capítulo mostra como podemos representar e armazenar essa quantidade e variedade de dados com o uso de estruturas de dados

apropriadas para este fim. O poder de representatividade das estruturas compostas que são apresentadas no capítulo permitem que manipulemos os dados envolvidos no problema dos funcionários ilustrado acima, fazendo uso de um único identificador.

# 5.1 Estruturas Homogêneas

Para certos problemas computacionais a associação de uma variável a um único endereço de memória não é adequada ou suficiente.

Considere, por exemplo, um algoritmo para calcular a média geral ponderada final dos formandos de um determinado curso universitário. A média geral ponderada de um aluno é calculada da seguinte forma:

1. Multiplica-se a média de cada uma das disciplinas cursadas pelo número de créditos respectivo daquela disciplina;
2. Realiza-se o somatório do resultado dessa multiplicação para todas as disciplinas
3. Divide-se este somatório pelo total de créditos cursados.

Se considerarmos um curso composto por 50 disciplinas no total, para cada formando teríamos que ler e armazenar os valores de 50 médias e 50 totais de crédito. Ou seja, nosso algoritmo seria como ilustrado na figura 5.1.

```
1.  Algoritmo MediaGeralPonderada;
2.  Variáveis
3.     m1, m2,...,m50: real; //médias das disciplinas
4.     c1, c2,...,c50: inteiro; //total de créditos
5.     nformandos: inteiro //número de formandos
6.     i: inteiro;
7.     nome: literal; //nome do formando
8.     mgp: real;
9.  Início
10.    leia (nformandos);
11.    para i <- 1 até nformandos faça
12.       leia (nome);
13.       leia (m1, c1);
14.       leia (m2, c2);
15.       leia (m3, c3);
16.       ...
17.       leia (m49, c49);
18.       leia (m50, c50);
19.       mgp <- (m1*c1 + m2*c2 + ... + m50*c50)/(c1+c2+...c50);
20.       escreva("MGP de ",nome," = ",mgp);
21.    fimpara;
22. Fim.
```

**Figura 5.1** – Cálculo da média geral ponderada utilizando variáveis simples.

Como pode ser observado, a quantidade de variáveis que teriam que ser definidas e a quantidade de comandos de leitura, atribuição, e de expressões para cálculo da `mgp` é enorme.

Grandes sistemas que lidam com grandes bancos de dados, normalmente necessitam de um número muito maior de variáveis, na ordem de centenas ou milhares. Sem dúvida, o trabalho de programar seria extremamente exaustivo e, na verdade, impraticável.

Para lidar com esses casos, foi criada uma alternativa para alocação de memória e, conseqüentemente, uma nova forma de definir variáveis. Essa nova é denominada de **variável indexada**.

Uma variável indexada corresponde a uma seqüência de endereços de memória, a qual se atribui um único nome, mas que podem ser acessadas individualmente através de um **índice**. O índice normalmente é representado por um valor inteiro. Cada um dos endereços de memória de uma variável indexada pode receber valores no decorrer da execução de um algoritmo como se se tratasse de uma variável comum. Entretanto, todos esses valores devem possuir o mesmo tipo. Por isso, uma variável indexada é considerada uma estrutura de dados **homogênea**.

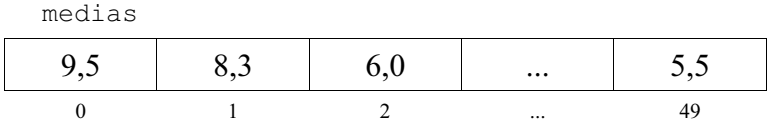
## 5.1.1 Estrutura composta unidimensional

Uma variável indexada por apenas uma dimensão de índices é conhecida pelo nome de **vetor**.

Uma variável vetor é declarada da seguinte forma:

*<nome>: vetor [N] de <tipo>;*

onde, <nome> é o identificador da variável vetor, N determina o número máximo de valores que a variável poderá armazenar e <tipo> representa o tipo desses valores. A figura 5.2 traz uma representação visual de um vetor de 50 posições (N = 50) que armazena valores reais (<tipo> = real).




Note que os índices variam de 0 a 49 e cada índice representa um endereço de memória que armazena um valor. Dessa forma, podemos utilizar um índice para resgatar um determinado valor armazenado ou para alterar o valor armazenado.

Por exemplo, considerando o vetor de nome `medias` da figura, se quiséssemos escrever o valor da primeira média utilizaríamos o comando `escreva` na forma

abaixo:

```
escreva(medias[0]);
```

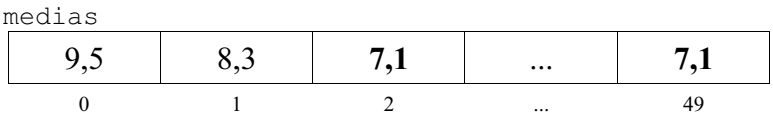
O número entre colchetes `[...]` representa o índice da variável `medias` cujo valor queremos resgatar.

Para proceder a uma alteração do valor da terceira média e da última média, utilizaríamos os comandos de atribuição abaixo, respectivamente:

```
medias[2] <- 7,1;
```

```
medias[49] <- 7,2;
```

O resultado dessas operações iria resultar no vetor medias ilustrado na figura 5.3, com os novos valores em destaque.




O pseudocódigo da figura 5.4 mostra como o uso de vetores ajudaria na solução do problema do cálculo da média geral ponderada, ilustrado anteriormente.

```
1.  Algoritmo MediaGeralPonderadaVetores;
2.  Variáveis
3.     m: vetor[50] de real; //médias das disciplinas
4.     c: vetor[50] de inteiro; //créditos
5.     somapond, somacred, mgp: real;
6.     nformandos: inteiro; //número de formandos
7.     i, j: inteiro; //variáveis de controle de laço
8.     nome: literal; //nome do formando
9.  Início
10.    leia (nformandos);
11.    para i <- 1 até nformandos faça
12.       leia (nome);
13.       somapond <- 0;
14.       somacred <- 0;
15.       para j <- 0 até 49 faça
16.          leia (m[j], c[j]);
17.          somapond <- somapond + m[j]*c[j];
18.          somacred <- somacred + c[j];
19.       fimpara;
20.       mgp <- somapond/somacred;
21.       escreva("MGP de ",nome," = ",mgp);
22.    fimpara;
23. Fim.
```

**Figura 5.4** – Uso de vetores para cálculo da média geral ponderada.

Nas linhas 3 e 4, são declarados dois vetores, `m` e `c`. O vetor `m` será responsável por armazenar todas as médias de um formando, enquanto que o vetor c armazena o total de créditos daquela disciplina. Note que as disciplinas são

conseqüentemente representadas pelos índices desses vetores. Ou seja, a primeira disciplina do curso é representada pelo índice 0 (zero), e portanto, sua média está guardada em `m[0]` e seu número de créditos está guardado em `c[0]`; e assim sucessivamente para todas as demais 49 disciplinas.

O algoritmo é organizado por um laço mais externo (linha 11) , que controla o número de formandos. Observe que para cada formando (cujo nome é lido na linha 12), a leitura das médias e dos totais de créditos (linha16) e a soma ponderada desses valores (linha 17) são realizados 50 vezes. Essa repetição é garantida pelo laço mais interno definido na linha 14, onde a variável de controle `j` funciona como índice para os vetores. Ao final da execução dessa repetição mais interna, a `mgp` do formando é calculada e escrita (linhas 20 e 21, respectivamente). Nesse momento o algoritmo passa para a leitura dos dados do próximo formando e todas as variáveis de soma (`somapond` e `somacred`) são apropriadamente zeradas (linhas 13 e 14). O processo se repete até se esgotar o número de formandos.

## 5.1.2 Estrutura composta multidimensional

Vamos supor agora que quiséssemos armazenar os dados das médias de cada disciplina para cada um dos formandos e também a MGP final, com o objetivo de imprimir algum tipo relatório ao final do processamento. Nesse caso, precisaríamos de uma estrutura capaz de armazenar uma espécie de tabela como a ilustrada na figura 5.5.

| | | 0 | 1 | 2 | 3 | ... | 49 | 50 |
|---|---|---|---|---|---|---|---|---|
| | | Cálculo I | Física I | Cálculo II | ICC | ... | Monografia | MGP |
| 0 | Fulano | 9,5 | 8,3 | 7,1 | 7,1 | | 9,5 | |
| 1 | Beltrana | 7,5 | 5,3 | 4,1 | 6,1 | | 8,5 | |
| 2 | Cicrano | 6,5 | 7,0 | 7,8 | 7,8 | | 9,0 | |
| ... | ... | | | | | | | |
| ... | ... | | | | | | | |
| 5 | Deltrana | 10,0 | 9,2 | 7,0 | 6,1 | | 9,5 | |

**Figura 5.5** – Tabela de média de alunos por disciplina.

Observe que, claramente, uma estrutura em formato de tabela se caracteriza por uma estrutura de duas dimensões, onde uma dimensão poderia ser representada pelos índices de linha e a outra dimensão pelos índices de coluna.

Uma variável indexada por duas dimensões de índices é conhecida pelo nome de **matriz** e é declarada da seguinte forma:

*<nome>: vetor [N, M] de <tipo>;*

Observe que o termo *vetor* continua a ser utilizado, no entanto, agora são considerados dois índices (N e M) ao invés de apenas um.

Considerando, por exemplo, a variável de nome `tabela` para representar a estrutura da figura 5.5, para escrever o valor da média de `Beltrana` em `Cálculo II`, podemos utilizar o comando `escreva` na forma abaixo:

```
escreva(tabela[1,2]);
```

ou seja, escreva o valor armazenado na variável tabela, indexado pela linha 0 e coluna 0, respectivamente. Esse valor seria 4,1.

Para alterarmos o valor da média de `ICC` e da monografia de `Deltrana`, utilizaríamos os comandos de atribuição abaixo, respectivamente:

```
tabela[5,3] <- 9,5;
tabela[5,49] <- 10,0;
```

O pseudocódigo da figura 5.6 ilustra o uso de matrizes no problema do cálculo da média geral ponderada para uma turma de 40 formandos em `Física`.

```
1.  Algoritmo MediaGeralPonderadaMatrizes;
2.  Variáveis
3.     m: vetor[40,51] de real; //médias
4.     c: vetor[50] de inteiro; //créditos
5.     nome: vetor[40] de literal; //nomes dos formandos
6.     somapond, somacred, mgp: real;
7.     nformandos: inteiro; //número de formandos
8.     i, j: inteiro; //variáveis de controle de laço
9.  Início
10.    para i <- 0 até 39 faça
11.       leia (nome[i]);
12.       somapond <- 0;
13.       somacred <- 0;
14.       para j <- 0 até 49 faça
15.          leia (m[i,j], c[j]);
16.          somapond <- somapond + m[i,j]*c[j];
17.          somacred <- somacred + c[j];
18.       fimpara;
19.       m[i,j+1] <- somapond/somacred;
20.       escreva("MGP de ",nome[i]," = ",m[i,j+1]);
21.    fimpara;
22. Fim.
```

**Figura 5.6** – Uso de matrizes para cálculo da média geral ponderada de todos os alunos de uma turma.

Ao final da execução do programa, teríamos uma variável vetor (`nome`) com os nomes dos formandos armazenados e uma variável matriz (`m`) contendo as médias de cada disciplina para cada um dos formandos.

Note, por exemplo, que o índice 0 da variável `nome` (`nome[0]`) armazena o nome do primeiro formando lido e todas as suas médias estão armazenadas na variável `m` indexada pela linha 0 e por uma seqüência de colunas que varia de 0 a

49. A coluna 50, acessada pelo índice `j+1` (linha 19), é reservada para guardar as MGPs dos formandos. Observe a figura 5.7 que ilustra a representação visual dessas variáveis.

Variável `m`

| [i] \ [j] | 0 | 1 | 2 | 3 | ... | 49 | 50 |
|---|---|---|---|---|---|---|---|
| 0 | 9,5 | 8,3 | 7,1 | 7,1 | | 9,5 | |
| 1 | 7,5 | 5,3 | 4,1 | 6,1 | | 8,5 | |
| 2 | 6,5 | 7,0 | 7,8 | 7,8 | | 9,0 | |
| 3 | | | | | | | |
| ... | | | | | | | |
| 39 | 10,0 | 9,2 | 7,0 | 6,1 | | 9,5 | |

Variável `nome`

| [i] | |
|---|---|
| 0 | Fulano |
| 1 | Beltrana |
| 2 | Cicrano |
| 3 | |
| ... | |
| 39 | Deltrana |

**Figura 5.7** – Variável bidimensional `m` e variável unidimensional `nome`.

Um bom exemplo do uso de estruturas multidimensionais é o problema da multiplicação de duas matrizes.

O pseudocódigo da figura 5.8 traz o algoritmo para solução do problema de multiplicação de duas matrizes 3x3. O resultado da multiplicação é armazenado em uma terceira matriz.

```
1.  Algoritmo MultiplicaçãoMatrizes;
2.  Tipos
3.     matriz = vetor[3,3] de inteiros;
4.  Variáveis
5.     a, b, r: matriz;
6.     i, j, k: inteiro;
7.  Início
8.     //laço para ler os valores da matriz a
9.     para i <- 0 até 2 faça
10.       para j <- 0 até 2 faça
11.          leia (a[i,j]);
12.       fimpara;
13.    fimpara;
14.    //laço para ler os valores da matriz b
15.    para i <- 0 até 2 faça
16.       para j <- 0 até 2 faça
17.          leia (b[i,j]);
18.       fimpara;
19.    fimpara;
20.    //laço para calcular a multiplicação de a por b
```

```
21.     para i <- 0 até 2 faça
22.        para j <- 0 até 2 faça
23.           r[i,j] <- 0;
24.           para k <- 0 até 2 faça
25.              r[i,j] <- r[i,j] + a[i,k]*b[k,j];
26.           fimpara;
27.        fimpara;
28.     fimpara;
29. Fim.
```

**Figura 5.8** – Algoritmo para multiplicação de duas matrizes 3x3.

Observe que ao se calcular qualquer elemento `r[i,j]`, o índice de linha `i` se repete na matriz `a` e o índice da coluna `j` se repete na matriz `b`. Já a coluna de `a` é igual a linha de `b`, e repete-se 3 vezes, de 0 a 2. O índice `k` efetua essa repetição e dessa forma um elemento `r[i,j] = a[i,k]*b[k,j]`, somados em três momentos conforme a variação de `k`.

Note nesse algoritmo o uso de uma nova seção, chamada **`Tipos`**. Assim como existem os tipos predefinidos de uma linguagem, `inteiro`, `real`, `literal`, `lógico`, etc, é possível definir novos tipos de dados. No exemplo, foi definido um novo tipo chamado `matriz`.

## 5.2 Estruturas Heterogêneas

Vetores e matrizes não permitem o armazenamento de valores de tipos distintos. Ou seja, não podemos, por exemplo, criar um vetor com dez índices e armazenarmos três valores inteiros, dois valores reais, quatro valores literais e um valor lógico.

Existem problemas, entretanto, que o uso de uma estrutura heterogênea dessa forma é bastante apropriado.

Considere como exemplo dados dos funcionários de uma instituição. Cada empregado está representado por um conjunto de dados que incluem um número de matrícula, um nome, data de admissão, salário, etc. A figura 5.9 ilustra o exemplo de uma ficha de um funcionário dessa instituição.

| Matricula | Nome | Sexo |
|---|---|---|
| 356609 | Maria Auxiliadora | F |
| **Admissão** | **Salário (R$)** | **Função** |
| 31/10/2008 | 4890,00 | Gerente |
| **Endereço** | **Cidade** | **Estado** |
| Av. Adélia Franco, 30 | Aracaju | SE |

**Figura 5.9** – Ficha de um funcionário contendo nove campos de dados.

Os dados existentes nessa ficha possuem valores de tipos diferentes. Por exemplo, o Nome, o Sexo, a Admissão, a Função, o Endereço, a Cidade e o Estado são tipos literais. A Matricula é um valor inteiro. E o Salário é claramente um valor numérico do tipo real.

Para facilitar a manipulação de uma estrutura desse tipo, podemos fazer uso de uma variável chamada de **registro**.

Uma variável registro é declarada da seguinte forma:

*<nome>: registro*

*<campo1>: <tipo>;*

*<campo2>: <tipo>;*

*...*

*<campoN>: <tipo>;*

*fimregistro*

onde, <nome> é o identificador da variável registro, <campo> são identificadores para cada um dos **campos** do registro, N determina o número de campos e <tipo> representa o tipo de cada um dos campos.

A figura 5.10 mostra a definição de um registro capaz de representar o exemplo da ficha de funcionários. Note que, primeiramente, um novo tipo chamado `ficha` é definido (linha 3) e então uma variável chamada `funcionario` é declarada (como pertencendo a esse novo tipo (linha 15).

```
1.  ...
2.  Tipos
3.     ficha = registro
4.        mat: inteiro;
5.       nome: literal;
6.       sexo: literal;
7.        adm: literal;
8.        sal: real;
9.        fun: literal;
10.       end: literal;
11.       cid: literal;
12.       est: literal;
```

```
13.   fimregistro;
14. Variáveis
15.    funcionario: ficha;
16. Início
17.   ...
18. Fim.
```

**Figura 5.10** – Definição do tipo de dado registro em pseudocódigo.

Para se fazer referência aos campos de um registro, utilizamos o identificador da variável do tipo registro seguido de um ponto e o identificador do campo desejado. Por exemplo, considerando a ficha da funcionária Maria Auxiliadora da figura 5.9, poderíamos utilizar o seguinte comando para aumentar seu salário em 20%.:

```
funcionario.sal <- funcionario.sal*1,2;
```

Para fazermos a leitura de seu novo endereço, utilizamos:

```
leia(funcionario.end);
```

O pseudocódigo da figura 5.11 faz uso de registro para representar os alunos de uma turma. Como resultado do processamento, o algoritmo escreve o total de aprovados e reprovados daquela turma.

```
1.  Algoritmo RegistroAluno;
2.  Tipos
3.    TipoAluno = registro
4.                  Nome: literal;
5.                  Matricula: literal;
6.                  Media: real;
7.                  Frequencia: inteiro;
8.                fimregistro;
9.  Variaveis
10.    Aluno: TipoAluno;
11.    Aprovados, Reprovados : inteiro;
12. Inicio
13.    Aprovados <- 0;
14.    Reprovados <- 0;
15.    escreva("Nome: ");
16.    leia(Aluno.Nome);
17.    enquanto Aluno.Nome <> "fim" faça
18.       escreva("Matricula: ");
19.       leia (Aluno.Matricula);
20.       escreva("Media: ");
21.       leia(Aluno.Media);
22.       escreva("Frequencia: ");
23.       leia(Aluno.Frequencia);
24.       se (Aluno.Media >= 7) e (Aluno.Frequencia >= 70) então
25.          Aprovados <- Aprovados + 1
26.       senão
27.          Reprovados <- Reprovados + 1;
28.       escreva("Nome: ");
29.       leia(Aluno.Nome);
30.    fimenquanto;
31.    escreva("Total de Aprovados: ", Aprovados);
```

```
32.    escreva("Total de Reprovados: ", Reprovados);
33. Fim.
```

**Figura 5.11** – Uso de registros para representar alunos de uma turma.

Um tipo registro para representar os alunos de uma turma foi definido na linha 3 (`TipoAluno`). Um aluno está representado pelo seu nome, número de matrícula, sua média e sua freqüência na disciplina; esses dados constituem os campos do registro.

A variável `Aluno`, definida na linha 10, é do tipo `TipoAluno`. Para realizar a leitura dos dados dos alunos, utiliza-se a variável `Aluno` seguida de um ponto e o identificador do campo que se deseja acessar. Observe, por exemplo, a leitura do nome de um aluno na linha 19: `leia(Aluno.Nome)`.

Note que o algoritmo realiza a leitura de todos os campos para cada aluno até que o usuário forneça como nome de um aluno a palavra `"fim"` (laço `enquanto` da linha 17).

## 5.2.1 Conjunto de registros

No exemplo visto anteriormente, realizamos a leitura dos dados de um aluno e, após a verificação de sua situação incrementamos o total de aprovados ou de reprovados da turma. Procedemos então para leitura dos dados de um segundo aluno, novamente verificamos sua situação e realizamos o incremento devido. Continuamos o processo até que o aluno de nome "fim" seja lido.

Se por acaso quiséssemos elaborar um relatório final contendo informações sobre todos os alunos lidos durante o processo, isso não seria possível. Por que?

Para que isso fosse possível seria necessário armazenar os dados desses alunos à medida que fossem lidos. O problema é que quando o laço é repetido sobrepomos os dados do aluno anterior. Isso acontece porque só é utilizada uma única estrutura do tipo registro.

Para solucionar o problema, podemos definir um conjunto de registros. Ou seja, podemos criar uma variável do tipo vetor, onde o valor armazenado em cada posição do vetor é do tipo registro.

Considere um problema semelhante, onde o registro de uma mercadoria de uma loja contém os seguintes dados: código, nome, preço e estoque. O problema consiste em dado um determinado código, exibir o nome, preço e estoque da mercadoria em questão (figura 5.12).

```
1.  Algoritmo Mercadorias;
2.  Tipos
```

```
3.     Tmercadoria = registro
4.                     COD : string[6];
5.                    NOME : string[15];
6.                    PRECO: real;
7.                     EST : integer;
8.                   fimregistro;
9.     ConjuntoMercadoria = vetor [50] de registro;
10. Variáveis
11.    TAB : ConjuntoMercadoria;
12.      I : integer;
13.      K : literal;
14.   RESP : literal;
15. Begin
16.   //Leitura dos dados das 50 mercadorias
17.   para I <- 0 até 49 faça
18.      leia(TAB[I].COD);
19.      leia(TAB[I].NOME);
20.      leia(TAB[I].PRECO);
21.      leia(TAB[I].EST);
22.   fimpara;
23.   repita
24.     //leitura da chave de pesquisa
25.     escreva("Entre com o código desejado: ");
26.     leia(K);
27.     //testa em cada registro se o código é igual a chave
28. pesquisada
29.     para I <- 0 até 49 faça
30.        se (K = TAB[I].COD) então
31.          escreva(TAB[I].NOME, TAB[I].PRECO, TAB[I].EST);
32.        //verifica se o usuário deseja pesquisar outro código}
33.        escreva("Outra mercadoria(S/N)?");
34.        leia(RESP);
35.     fimpara;
36.   até (RESP = "N");
37. Fim.
```

**Figura 5.12** – Uso de um vetor de registros para representar alunos de uma turma.

## 5.2.2 Registro de conjuntos

Da mesma forma que podemos criar um vetor que armazene valores do tipo registro, é possível criar registros com campos do tipo vetor.

Vamos retomar o problema do compto do número de aprovados e reprovados de uma turma, solucionado pelo pseudocódigo da figura 5.11. Caso o registro de um aluno na universidade não possuísse um campo para representar a média, mas sim, um conjunto de três notas, como poderíamos modificar o algoritmo em questão para refletir essa mudança?

Observe o pseudocódigo da figura 5.13. Note que ao invés do campo `Media` do tipo `real`, o registro `TipoAluno` contém um campo `Notas` do tipo composto `vetor`.

```
1.  Algoritmo RegistroAluno;
2.  Tipos
3.    TNotas = vetor[3] de real;
4.    TipoAluno = registro
5.                   Nome: literal;
6.                   Matricula: literal;
7.                   Notas: TNotas;
8.                   Frequencia: inteiro;
9.                fimregistro;
10. Variaveis
11.    Aluno: TipoAluno;
12.    Aprovados, Reprovados: inteiro;
13.    media: real;
14. Inicio
15.    Aprovados <- 0;
16.    Reprovados <- 0;
17.    escreva("Nome: ");
18.    leia(Aluno.Nome);
19.    enquanto Aluno.Nome <> "fim" faça
20.       escreva("Matricula: ");
21.       leia (Aluno.Matricula);
22.       escreva("Notas: ");
23.       leia (Aluno.Notas[0],Aluno.Notas[1],Aluno.Notas[2]);
24.       media <- (Aluno.Notas[0]+
25.                 Aluno.Notas[1]+
26.                 Aluno.Notas[2]) / 3;
27.       escreva("Frequencia: ");
28.       leia(Aluno.Frequencia);
29.       se (media >= 7) e (Aluno.Frequencia >= 70) então
30.          Aprovados <- Aprovados + 1
31.       senão
32.          Reprovados <- Reprovados + 1;
33.       escreva("Nome: ");
34.       leia(Aluno.Nome);
35.    fimenquanto;
36.    escreva("Total de Aprovados: ", Aprovados);
37.    escreva("Total de Reprovados: ", Reprovados);
38. Fim.
```

**Figura 5.13** – Uso de registros com campos compostos para representar alunos de uma turma e suas três notas.

# 5.3 Estruturas de Dados Compostas em Java

## 5.3.1 Vetores e Matrizes

Em Java, vetores, matrizes e estruturas compostas homogêneas de mais dimensões são declaradas utilizando a seguinte sintaxe:

*<tipo>[][]...[] x*,

onde <tipo> representa qualquer tipo primitivo Java ou definido pelo usuário,

e o número de colchetes representa o número de dimensões do vetor. Por exemplo, a declaração a seguir representar um vetor de valores inteiros com identificador x:

*int[] x;*

Para reservarmos uma quantidade fixa de posições de memória para o vetor x, utilizamos a sintaxe utilizada pelo exemplo abaixo:

```
x = new int[50];
```

o que significa dizer que a variável x representa 50 posições de memória e, portanto, tem capacidade para armazenar 50 valores inteiros.

Para atribuirmos um valor a uma determinada posição no vetor, indicamos a posição que queremos entre colchetes e utilizamos o comando de atribuição de Java, como ilustrado abaixo:

```
x[2] = 135;
```

o que significa dizer que na terceira posição do vetor x iremos armazenar o valor inteiro 135.

A figura 5.14 traz um programa Java que escreve os nomes de todos os meses do ano, a estação do ano a que pertencem e o n.º de dias de cada mês, usando variáveis do tipo vetor.

```
1.  public class Meses {
2.     static String meses[] = {"Janeiro", "Fevereiro", "Março",
3.                              "Abril", "Maio", "Junho",
4.                              "Julho", "Agosto", "Setembro",
5.                              "Outubro","Novembro","Dezembro"};
6.     static int diasmes[] = { 31, 28, 31, 30, 31, 30, 31, 31,
7.                              30, 31, 30, 31};
8.     public static void main(String args[]) {
9.            String estacao = "";
10.           for (int mes = 0; mes < 12; mes++) {
11.               switch (mes + 1){
12.                     case 1:
13.                     case 2:
14.                     case 3: estacao = "verão"; break;
15.                     case 4:
16.                     case 5:
17.                     case 6: estacao = "outono"; break;
18.                     case 7:
19.                     case 8:
20.                     case 9: estacao = "inverno"; break;
21.                     case 10:
22.                     case 11:
23.                     case 12: estacao = "primavera";
24.               }
25.                System.out.println(meses[mes] + " é um mês
26.                   do(a)" + estacao + " com " + diasmes[mes]
27.                   + " dias.");
28.           }
29.    }
```

```
30. }
```

**Figura 5.14** – Exemplo do uso de vetores em Java para imprimir nome, estação do ano e número de dias para cada um dos meses do ano.

Observe no código que são definidos duas variáveis do tipo vetor: um vetor de String para armazenar o nome de todos os meses do ano (linha 2) e um vetor de inteiro para armazenar o número de dias de cada um dos 12 meses do ano, na ordem de janeiro a dezembro (linha 6). Note que os valores a serem armazenados por cada posição do vetor já estão definidos no código através do uso de chaves `{ }`. O programa principal, simplesmente realiza uma repetição com variável de controle (`mes`) que varia de 0 a 11, correspondendo aos índices dos dois vetores (linha 10). A estrutura de múltipla escolha `switch` da linha 11 testa o valor dessa variável e atribui à variável que armazena o valor da estação do ano (estacao), o nome da estação correspondente ao mês em questão. O último comando do laço (linha 25) imprime a mensagem objetivo para cada mês do ano.

O programa ilustrado na figura 5.15 mostra um outro exemplo do uso de vetores unidimensionais em Java. O programa lê um conjunto de alunos de uma turma, cada um com o nome e a média obtida na disciplina. Em seguida exibe o nome dos alunos que possuem a nota maior do que a média alcançada pela turma.

```
1.  public class MaioresMedias {
2.      public static void main (String[] args){
3.          String alunos[] = new String[10];
4.          float notas[] = new float[10];
5.          float soma = 0, media = 0;
6.          for(int i = 0; i <= 9; i++){
7.              System.out.println("Entre com o nome do "+
8.                                 (i+1)+"° aluno:");
9.              alunos[i] = System_in.readString();
10.             System.out.println("Entre com a nota do "+
11.                                (i+1)+"° aluno:");
12.             notas[i] = System_in.readFloat();
13.             soma = notas[i] + soma;
14.         }
15.         media = soma / 10;
16.         System.out.println("Média da turma: "+media);
17.         System.out.println("Alunos com nota maior que a média
18.                            da turma:");
19.         for(int j = 0; j <= 9; j++){
20.             if(notas[j] > media){
21.                 System.out.println(alunos[j]);
22.             }
23.         }
24.     }
25. }
```

**Figura 5.15** – Exemplo do uso de vetores em Java para imprimir o nome dos alunos com

médias maiores que a da turma.

Foi definido um vetor para representar o nome de 10 (dez) alunos (linha 3) e um vetor para representar as notas (médias) desses alunos (linha 4). Em seguida, um laço `for` é utilizado para solicitar ao usuário o nome e nota de cada um desses 10 alunos (linha 6). Cada nome lido é armazenado no vetor `alunos` na posição indicada pela variável de controle do laço `i` (linha 9). Processo semelhante é feito para as notas lidas (linha 12). Ao final do laço de leitura, a média da turma é calculada usando o somatório das notas lidas e dividindo este valor por 10 (dez) (linha 15). Finalmente, um novo laço (linha 19) é utilizado para acessar o valor armazenado em cada índice da variável `notas` e compará-lo com o valor da média da turma, armazenado na variável `media`. Os índices (valor de `j`) cuja nota for maior que a média (linha 20) serão utilizados para imprimir o nome armazenado no vetor `alunos` correspondendo àquele mesmo valor de índice (`alunos[j]`, linha 21).

O uso de matrizes em Java é exemplificado no programa da figura 5.16. O programa realiza a leitura dos elementos de uma matriz e aponta a posição (linha X coluna) do elemento de menor valor da matriz.

```
1.  public class MenorValorMatriz {
2.      public static void main(String[] args){
3.            int[][] a = new int[10][10];
4.            int mlin = 0, mcol = 0, me;
5.            System.out.println("Digite o número de linhas e o de
6.                                colunas:");
7.            int m = System_in.readInt();
8.            int n = System_in.readInt();
9.            for (int i = 0; i < m; i++){
10.               System.out.println("Digite os valores da
11.                                   linha "+ (i+1));
12.               for (int j = 0; j < n; j++)
13.                   a[i][j]= System_in.readInt();
14.           }
15.           System.out.println("\nMatriz lida:");
16.           for (int i = 0; i < m; i++){
17.                for (int j = 0; j < n; j++){
18.                    System.out.print("\t" +a[i][j]+ " ");
19.                }
20.                System.out.println();
21.           }
22.           me = a[0][0];
23.           for (int i = 0; i < m; i++){
24.                for (int j = 0; j < n; j++)
25.                    if (a[i][j] < me){
26.                         me = a[i][j];
27.                         mlin = i;
28.                         mcol = j;
29.                    }
```

```
30.             }
31.             System.out.println("\nPosição do menor valor:\n" +
32.                                "Linha = " + (mlin+1)+
33.                                "\tColuna = " + (mcol+1));
34.         }
35.  }
```

**Figura 5.16** – Exemplo do uso de matrizes em Java para imprimir os índices (linha x coluna) referentes ao menor elemento armazenado.

Observe a declaração da matriz `a` de inteiros com capacidade para armazenar até 10 x 10 valores (linha 3). O programa pede então que o usuário forneça a dimensão da matriz (linhas 5 à 8). Os valores fornecidos são armazenados em duas variáveis `m` e `n`, que representarão os limites da linha e da coluna da matriz, respectivamente. Em seguida dois laços `for` aninhados (linhas 9 e 12) são utilizados para armazenar os valores na matriz. Esses valores são lidos pelos usuários (linha 13). Uma vez criada, a matriz é impressa (linhas 16 à 21) e a identificação do menor elemento é feita varrendo a matriz e comparando cada um dos valores armazenados com o menor até então (linhas 23 à 30).

Vetores e matrizes são estruturas poderosas de representação e manipulação de dados. Diversos problemas com solução computacional seriam extremamente difíceis de serem resolvidos sem o uso de uma estrutura composta desse tipo.

Um desses problemas é a implementação de um programa para imprimir o conhecido **Triângulo de Pascal**. Cada elemento do Triângulo de Pascal é igual à soma dos elementos que lhe ficam imediatamente acima, caso o elemento não seja nem o primeiro nem o último da sua linha, e é 1 no caso contrário. Um exemplo do Triângulo de Pascal de tamanho 6 é ilustrado na figura 5.17.

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 1 | | | | | |
| 1 | 1 | 1 | | | | |
| 2 | 1 | 2 | 1 | | | |
| 3 | 1 | 3 | 3 | 1 | | |
| 4 | 1 | 4 | 6 | 4 | 1 | |
| 5 | 1 | 5 | 10 | 10 | 5 | 1 |

$$M_{i,j} = \begin{cases} 1 \leftarrow j = 0 \vee j = i \\ M_{i-1,j-1} + M_{i-1,j} \leftarrow outros\,casos \end{cases}$$

**Figura 5.17** – Triângulo de Pascal de tamanho 6.

Na solução Java proposta na figura 5.18, a dimensão do Triângulo é especificada pelo usuário.

```
1.  class TPascal {
2.     public static void main(String Args[]) {
3.            System.out.print("Introduza a dimensão do Triângulo:
4.                             " );
5.            int dimensao = System_in.readInt();
6.            int [][] matriz = new int [dimensao][dimensao];
7.            for (int i = 0;i < dimensao; i++)
8.               for (int j = 0; j <= i;j++)
9.                     if ((j==0) || (j == i))
10.                          matriz[i][j] = 1;
11.                    else
12.                          matriz[i][j] = matriz[i-1][j-1] +
13.                                         matriz[i-1][j];
14.           for (int i = 0;i < dimensao; i++) {
15.               for (int j = 0; j <= i;j++)
16.                     System.out.print(matriz[i][j] + " " );
17.               System.out.println();
18.           }
19.    }
20. }
```

**Figura 5.18** – Implementação do Triângulo de Pascal em Java.

## 5.3.2 Registros

A linguagem Java não possui uma sintaxe específica para criação do tipo registro. Mas existe uma estrutura em Java que pode simular um registro: a **classe**.

Assim como registros, classes Java são utilizadas para agrupar diferentes dados que se referem a uma mesma entidade do problema. Por exemplo, podemos definir uma classe de alunos que agrupa dados como nome, número de matrícula, sexo, endereço, curso, disciplinas cursadas e respectivas médias, e média geral ponderada. A sintaxe para definição de classes em Java está ilustrada abaixo:

*class <nome> {*
  *<tipo> ident1;*
  *<tipo> ident2;*
  *...*
*}*

onde, <nome> representa o nome do tipo classe a ser utilizado na declaração de variáveis desse tipo, e <tipo> representa o tipo dos elementos que pertencem à classe.

Então, para simular em Java o registro definido no algoritmo da figura 5.11, por exemplo, escreveríamos algo como ilustrado abaixo:

```
class TipoAluno {
      String nome;
      String matricula;
      float media;
      int frequencia;
},
```

e definiríamos uma variável para identificar a classe no programa da seguinte forma:

```
...
TipoAluno aluno;
aluno = new TipoAluno();
...,
```

onde a segunda linha reserva de fato um espaço na memória suficiente para armazenar dois valores String (`nome` e `matricula`), um valor do tipo float (`media`) e um valor inteiro (`frequencia`).

A atribuição de um valor a um campo da classe se faz através do identificador da classe seguido de um ponto e o identificador do campo desejado:

```
aluno.nome = "Fulano";
```

Considere como exemplo do uso de registros como classes em Java um programa que leia e armazene os valores relativos aos lucros obtidos em cada mês do ano por uma empresa, num conjunto de 5 lojas distintas. Ao final, o programa apresenta o valor total de lucros da empresa, bem como o valor máximo obtido e em que loja este valor foi obtido (figura 5.19).

```
1.  class Lucros {
2.     public static void main(String Args[]) {
3.           class Loja {
4.                 int numero;
5.                 float[] lucro = new float[12];
6.           }
7.           class Empresa {
8.                 String nome;
9.                 String setor;
10.                Loja[] loja = new Loja[5];
11.          }
12.          Empresa empresa = new Empresa();
13.          empresa.nome = "Fantasia";
14.          empresa.setor = "Construção";
15.          float maxlucro = 0;
16.          Loja melhorLoja = null;
17.          int num, mes;
18.          float total = (float)0.0;
19.          for (num = 0; num < 5; num++)
```

```
20.                 for (mes = 0; mes < 12; mes++)
21.                       empresa.loja[num] = new Loja();
22.             for (num = 0; num < 5; num++)
23.                 for (mes = 0; mes < 12; mes++) {
24.                       empresa.loja[num].numero = num+1;
25.                       System.out.print("Introduza o valor do
26.                              lucro em 2001 na loja " + (num+1)
27.                              + " no mes " + (mes+1) + ": ");
28.                       empresa.loja[num].lucro[mes] =
29.                                              System_in.readFloat();
30.                 }
31.                       // Procura a loja com o melhor lucro
32.             for (num = 0; num < 5; num++)
33.                 for (mes = 0; mes < 12; mes++) {
34.                   if (maxlucro < empresa.loja[num].lucro[mes]) {
35.                       maxlucro = empresa.loja[num].lucro[mes];
36.                       melhorLoja = empresa.loja[num];
37.                   }
38.                   total += empresa.loja[num].lucro[mes];
39.                 }
40.             System.out.println("Total de Lucros: " + total);
41.             System.out.println("Máximo de lucros obtido na
42.                 loja:"  + (melhorLoja.numero) + " no valor de: "
43.                 + maxlucro);
44.     }
45. }
```

**Figura 5.19** – Uso de registros como classes em Java.

A solução define duas classes a serem usadas como registros: a classe `Empresa` que representa uma empresa contendo um nome, um setor de atuação e um conjunto de lojas (linhas 7 à 11), e a classe `Loja` que representa uma loja em particular, contendo seu número identificador e um vetor de lucros que armazena o lucro da loja obtido em cada mês do ano (linhas 3 a 6).

Na linha 12, o espaço em memória é reservado (alocado) para armazenar todos os dados de uma empresa. O trecho de código das linhas 19 à 21 é responsável por reservar espaço em memória para aramazenar os dados de cada uma das lojas de uma empresa. Da linha 22 à linha 30, o programa solicita ao usuário que forneça os lucros obtidos em todos os meses por cada uma das lojas da empresa, e esse valor é armazenado em sua posição específica dentro do vetor que representa as lojas (linha 28). Uma vez devidamente armazenados, todos os valores são acessados através dos laços aninhados das linhas 32 e 33, e cada valor é comparardo com uma variável que armazena o maior lucro obtido até então (`maxLucro`, linha 35) e uma outra que armazena qual o número da loja que obteve esse maior lucro (`melhorLoja`, linha 36). O lucro total obtido pela empresa é vai sendo acumulado na linha 38. Ao final, linhas 40 e 41, as informações desejadas são impressas.

# Resumo

- Muitos problemas necessitam de tipos de dados mais complexos para que os valores manipulados sejam mais adequadamente representados.
- Linguagens de programação permitem que o programador defina seu próprio tipo de dado de acordo com a necessidade dele.
- Dois tipos de dados compostos são bastante utilizados em programação: estruturas compostas homogêneas e estruturas compostas heterogêneas.
- Um vetor é uma estrutura de dados composta homogênea que permite o uso de apenas um identificador para representar várias células contíguas de memória que armazenam dados de um mesmo tipo.
- Em um vetor, cada valor armazenado é acessado através de um índice.
- Um vetor pode ter uma ou mais dimensões; uma matriz é um vetor bidimensional e pode ser utilizada para representar estruturas tipo tabela.
- Um registro é uma estrutura composta heterogênea que permite o uso de apenas um identificador para representar células de memória que armazenam dados de tipos diferentes.
- A ficha de um funcionário contendo nome, sexo, idade, matrícula, cargo, salário, etc, é um exemplo de registro.
- Java permite a declaração de variáveis compostas homogêneas e heterogêneas.
- Vetores em Java podem ter n dimensões.
- Registros em Java são simulados através da definição de classes extras.
- Uma classe Java possui um nome e a possibilidade de conter várias variáveis agrupadas de tipos diferentes, incluindo vetores e outras classes.

# Na próxima aula...

… veremos como um problema pode ser dividido em subproblemas e como escrever nosso programa de forma a refletir essa característica.

# Referências e Sugestões de Leitura

Estruturas compostas de Dados são apresentadas no capítulo 5 e 6 de

**LEITE, M., Técnicas de Programação: uma Abordagem Moderna, BRASPORT, 2006.**

e no capítulo 4 de

**FORBELLONE, A., EBERSPÄCHER, H. Lógica de Programação. Prentice Hall, 3ed, 2005.**

Para um aprofundamento teórico sobre os tipos compostos de dados recomendo leitura do capítulo 6 de

**SEBESTA, R. Conceitos de Linguagens de Programação, Bookman, 5ed, 2005.**

e capítulo 3 de

**VAREJÃO, F. Linguagens de Programação: Conceitos e Técnicas, Campus, 2004.**

Para utilizar estruturas de dados compostas em Java, recomendo capítulo 7 (sobre vetores e matrizes) e capítulo 8 (sobre classes como registros) de

**DEITEL, H., DEITEL, P. Java como Programar, Prentice-Hall, 2005.**

# Exercícios Propostos

(PSEUDOCÓDIGO) Para cada um dos problemas a seguir crie um algoritmo para resolvê-lo utilizando a sintaxe de pseudocódigo.

*(Algoritmos com registros)*

5.1 – Definir um registro para conter informações sobre uma disciplina. Deve guardar o código da disciplina (6 dígitos), o nome da disciplina (até 40 caracteres) e o período. Escreva um programa que leia os dados de 3 disciplinas usando variáveis do tipo do registro e imprima todos os dados na tela.

5.2 – Definir um registro para conter a avaliação de um aluno em uma turma. O registro deve conter a matrícula do aluno, o nome do aluno, a freqüência e as três notas. Escreva um programa que leia os dados de 3 alunos usando variáveis do tipo do registro e imprima os dados dos alunos com média maior ou igual a 7 na tela.

5.3 – Definir um registro professor contendo a matrícula (7 dígitos), o nome (até 30 caracteres).

5.4 – Defina um registro para guardar informações sobre uma turma. Esse registro deve usar os registros definidos nas questões 1, 2 e 3. Cada turma pode ter no máximo 100 alunos, mas deve-se guardar o número de alunos realmente matriculados na disciplina, além das informações a eles associadas. Devem ser mantidas as informações sobre o professor e a disciplina. Escreva um programa que preencha os dados da turma. O usuário deve preencher os dados da disciplina e

professor, além de informar quantos alunos estão na turma, fornecendo os dados de cada um deles. No final, o programa deve apresentar na tela todos os dados lidos, além da situação do aluno na disciplina (AP, RM ou RF, para freqüência mínima de 70% e média >= 5).

*(Algoritmos com vetores e matrizes)*

5.5 – Armazenar dez números na memória do computador. Exibir os valores na ordem inversa à da digitação.

5.6 – Armazenar dez valores na memória do computador. Após a digitação dos valores, criar uma rotina para ler os valores e achar e exibir o maior deles.

5.7 – Armazenar vinte valores em um vetor. Após a digitação, entrar com uma constante multiplicativa que deverá multiplicar cada um dos valores do vetor e armazenar o resultado no próprio vetor, na respectiva posição.

5.8 – Armazenar vinte valores na memória. Após a digitação, entrar com uma constante multiplicativa que deverá multiplicar cada um dos valores do vetor e armazenar o resultado em outro vetor, porém em posições equivalentes. Exibir os vetores na tela.

5.9 – Armazenar o nome, sexo e idade de cem pessoas. Consistir as entradas no sentido de aceitar apenas “F” ou “M” para o sexo e valores positivos para a idade. Após a digitação dos dados, exibir os dados (nome, sexo e idade) de todas as pessoas do sexo feminino.

5.10 – Armazenar o nome, sexo e idade de cem pessoas. Consistir as entradas no sentido de aceitar apenas “F” ou “M” para o sexo e valores positivos para a idade. Após a digitação dos dados, exibir os dados de todas as pessoas com idade superior a vinte anos. No final da listagem, exibir a quantidade de pessoas que foram listadas e a porcentagem que este valor representa em relação ao total de pessoas digitadas.

5.11 – Alterar o programa anterior, no sentido de controlar o layout final de tela, para que o usuário pressione uma tecla para visualizar os dados das pessoas passo a passo, por exemplo, de dez em dez pessoas.

5.12 – Entrar via teclado com doze valores e armazená-los em uma matriz de ordem 3x4. Após a digitação dos valores solicitar uma constante multiplicativa, que deverá multiplicar cada valor matriz e armazenar o resultado em outra matriz de mesma ordem, nas posições correspondentes. Exibir as matrizes na tela, sob a forma matricial, ou seja, linhas por colunas.

5.13 – Entrar com uma matriz de ordem MxN, onde a ordem também será escolhida pelo usuário, sendo que no máximo 10x10. A matriz não precisa ser quadrática. Após a digitação dos elementos, calcular e exibir a matriz transposta.

5.14 – Entrar com uma matriz de ordem MxM, onde a ordem também será escolhida pelo usuário, sendo que no máximo será de ordem 10 e quadrática. Após a digitação dos elementos, calcular e exibir a matriz inversa. Exibir as matrizes na tela, sob a forma matricial (linhas x colunas).

5.15 – Entrar com uma matriz de ordem MxM, onde a ordem também será escolhida pelo usuário, sendo que no máximo será de ordem 10 e quadrática. Após a digitação dos elementos, calcular e exibir determinante da matriz.

(JAVA) Para cada um dos problemas a seguir crie um programa Java para resolvê-lo utilizando vetores, matrizes e registros (classes) sempre que achar adequado.

5.16 – Leia um conjunto de notas, cuja quantidade seja determinada pelo usuário. Calcule a média de todas elas. Exiba o conjunto das notas maiores do que a média calculada. Após a exibição anterior, exiba o outro conjunto de notas que são menores do que a média. Exiba também a quantidade de notas que é exatamente igual a média.

5.17 – Leia um conjunto de salários, sendo que para terminar a entrada será fornecido o valor -1. Após toda a entrada ter sido realizada, leia o valor de um reajuste. Em seguida exiba todos os salários já reajustados.

5.18 – Leia um conjunto de números. Exiba este conjunto acompanhado do seu elemento simétrico em relação a sua posição no conjunto. Veja o exemplo:

Entrada: 5, 7, 9, 2

Saída:

5 - 2

7 - 9

9 - 7

2 - 5

5.19 – Leia dois conjuntos de números com a mesma quantidade. Exiba a intersecção dos conjuntos, ou seja, os números que são repetidos nos dois conjuntos.

5.20 – Faça um software para calcular a distribuição de freqüência de 15 valores sorteados de 0 a 9. Suponha os seguintes números sorteados: 4, 5, 7, 8, 9, 1, 8, 2, 4, 3, 2, 5, 6, 7, 0

Saída do programa:

Número 0: 1 vez

Número 1: 1 vez

Numero 2: 2 vezes

Número 3: 1 vez

Número 4: 2 vezes

Número 5: 2 vezes

Número 6: 1 vez

Número 7: 2 vezes

Numero 8: 2 vezes

Número 9: 1 vez

5.21 – Faça um simulador do famoso jogo "leilão do menor valor único". Dica: peça ao usuário para fornecer o valor máximo do leilão, em seguida crie um vetor para armazenar a quantidade de lances para cada valor possível. Para isto dimensione o tamanho do vetor para 100 vezes o valor máximo do lance (assim ele poderá representar os centavos também).

5.22 – Leia dois conjuntos de números (podem ter o tamanho diferente) já ordenados de forma crescente. Crie um outro vetor para armazenar os dois conjuntos unidos, sendo que os números devem permanecer ordenados. Finalmente, exiba este vetor resultante.

5.23 – Criar um programa que represente um "Jogo da Velha", onde o programa solicita os nomes dos jogadores e passa a administrar a jogada de cada um. Solicitar de cada jogador, a posição do "tabuleiro" que deseja jogar e consistir esta posição. Evidentemente que não poderão ser feitas jogadas em posições já ocupadas. Exibir mensagem divulgando o nome do vencedor ou então "empate", se for o caso. Perguntar se o usuário deseja ou não uma nova partida.

5.24 – Vamos fazer uma excursão para Arauá. Para isto, vamos de ônibus, e precisaremos controlar as reservas de lugares do ônibus. Sabe-se que o ônibus possui quatro fileiras de dez cadeiras cada. Fazer um programa para solicitar o nome do usuário e o lugar que pretende reservar (fileira e cadeira), e se este lugar estiver disponível o programa deverá concretizar a reserva, caso contrário, enviar mensagem comunicando que esta poltrona já está ocupada. Perguntar se existe mais alguém que queira viajar para a cidade, e em caso negativo exibir um "mapa" mostrando os nomes e lugares de cada passageiro que efetuou a reserva, assim como os lugares que permanecem livres. Lembre-se que o ônibus possui uma capacidade limitada de poltronas e que o programa deverá encerrar estas reservas, caso esta capacidade tenha sido alcançada.

5.25 – Criar um programa para controlar as reservas de poltronas de uma peça teatral, "A UFS e seus *Nerds*", sabendo que a peça será apresentada em três sessões, manhã, tarde e noite e que o teatro possui "X" fileiras de "Y" cadeiras cada. Os valores de "X" e "Y" serão digitados. O usuário digita seu nome, lança a

sessão, o número da fileira e da cadeira que pretende reservar, e se estiver livre a reserva será efetuada, caso contrário, o programa deverá enviar mensagem comunicando que a poltrona está ocupada e solicitar outro lugar. Perguntar ao usuário se mais alguém pretende fazer reservas. As reservas poderão ser efetuadas até completar um máximo de quatro quintos da capacidade total do teatro. No final do programa de reservas, exibir um "mapa" mostrando as poltronas do teatro, por sessão, com os nomes de cada ocupante, ou ainda com a informação "Poltrona livre".

5.26 – Num parque de estacionamento foi registado, ao longo de um dia, o movimento de entrada e saída de veículos. O parque está aberto diariamente das 8:00 às 24:00. Na abertura está sempre vazio e a sua capacidade é de 100 veículos. Sejam entradas e saídas vectores, de dimensão 16, que representam o número de entradas e saídas ocorridas em cada hora. Escreva métodos numa classe Java para determinar:

a) uma das horas do dia em que houve maior diferença entre as entradas e as saídas (essa hora deve ser o resultado do método);

b) o número de horas do dia em que a ocupação do parque foi máxima, escrevendo este valor e uma mensagem a indicar se o parque está completo ou não;

c) os intervalos em que o número de veículos no parque aumentou estritamente. Para cada intervalo, devem ser escritas em três colunas a hora inicial, a hora final e qual o aumento total nesse intervalo.

5.27 – Dada uma tabela bidimensional a[N][M] (M e N constantes) de número inteiros, defina uma classe Java e métodos para implementar cada uma das seguintes operações sobre a tabela:

a) trocar o menor com o maior número inteiro de cada linha;

b) deslocar a primeira coluna para a segunda, a segunda para a terceira, etc., a (N−1)-ésima para a primeira;

c) deslocar a linha 1 para a linha 0, a linha 2 para a linha 1, etc., a linha 0 para a linha M−1;

d) substituir todas as colunas pela coluna cuja soma dos elementos é mínima;

e) ordenar a matriz considerando o seguinte critério: a linha i é maior do que a linha j, se na coluna de menor índice em que os elementos diferem, o elemento da linha i é maior do que o elemento da linha j.

5.28 – Dada uma tabela bidimensional com 1000 linhas e 7 colunas representando o resultado de 1000 sorteios do Totoloto (para valores inteiros não repetidos entre 1 e 49). Defina uma classe Java e escreva métodos para implementar as seguintes operações:

a) para cada sorteio o número de pares de números consecutivos;

b) os 3 números que ocorrem mais vezes;

c) a maior seqüência de números consecutivos num sorteio.

5.29 – Simulação do jogo do galo. O jogo do galo joga-se num tabuleiro de 3×3 e dois jogadores. Cada jogador coloca alternadamente um 1 ou um 2 numa das quadrículas e ganha o jogador que colocar 3 peças em linha (horizontal, vertical ou diagonal principal). O tabuleiro é representado por uma tabela a[2][2] que contém as jogadas realizadas, supondo-se que se o jogador i jogou num certa posição então o conteúdo dessa posição é i, para i = 1,2. Se uma posição ainda não foi jogada, o conteúdo de a nessa posição é 0. Nenhum jogador pode jogar numa posição já preenchida e o jogo termina empatado se todas as posições estão ocupadas e nenhum jogador ganhou. Em cada jogada deve ser imprimido o tabuleiro, i.e. a. No início o programa deve pedir o nome de cada um dos jogadores e imprimir o nome do jogador que ganhar.