PROGRAMAÇÃO I - CCT0827

Semana Aula: 10

Unidade 4. Tratamento de Dados Homogêneos

Tema

Tratamento de Dados Homogêneos

Palavras-chave

Vetor, Matrizes, Vetor de Objetos, ArrayList

Objetivos

O aluno deverá ser capaz de:

- Compreender o uso de vetor de tipos primitivos e de vetor de objetos.
- Escrever programas com vetor de tipos primitivos conceito de exceção.
- Realizar aplicações com ArrayList.

Estrutura de Conteúdo

Vetor

Semelhante ao C/C++, o Java também dá suporte a vetores e matrizes multidimensionais. No entanto, em Java, vetores são objetos.

Para declarar um vetor, usamos [ ] após o tipo desejado:

int [ ] v; ou também int v[ ];

float [] notas; ou também float notas[ ];

Outra diferença importante é que, ao declarar um vetor, **NÃO** definimos o seu tamanho.

Isso significa dizer que, ao declarar um vetor, o Java **NÃO** aloca espaço na memória para o vetor.

Como vetor é um objeto, precisamos criá-lo com o operador new. Nesse momento definimos o seu tamanho.

```
float[] nota; //poderia ser float nota[];
nota = new float[10];
```

O tamanho do vetor pode ser definido usando uma variável ou uma expressão.

Atenção: Se você acessar uma área fora do intervalo do vetor, ocorrerá uma exceção (*ArrayIndexOutOfBoundsException*) e a execução do programa será interrompida.

Assim como fazemos com variáveis comuns, também podemos inicializar vetores.

```
tipo[] nome = { lista de valores };
```

onde:

lista de valores: é uma lista cujos elementos são separados por vírgula

Exemplo :

```
float[] nota = { 5.5, 6.5, 7.0, 9.0, 8.0 };
```

Para criar vetores multidimensionais basta usar mais de uma dimensão na definição e na criação do vetor:

```
float[][] matriz;
matriz = new float[10][20];
```

Usamos **length** para recuperar o tamanho do vetor.

Existe uma outra sintaxe do comando for para percorrer vetores. Essa forma é usada apenas para ler os dados do vetor, mas não serve para alterar o vetor.

Exemplo: Aqui estamos usando o for-each. Não é preciso indexar.

```
public class Vetor {
public static void main(String[] args) {
        int[] v = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
        for (int n : v)
        System.out.println(n);
}
}
```

Observação: Quando trabalhamos com vetor de objetos é preciso criar cada objeto com o

operador new. Este estudo será feito, em detalhes, na aula seguinte.

Vetor de objetos

Quando usamos um vetor de objetos, ao criar o vetor não estamos criando os objetos em si, mas apenas referências para os objetos.

Além do vetor, cada objeto deve ser criado com o operador new.

```
public class EditorGrafico{
public static void main(String[] args) {
        Retangulo[] retangulos;
        retangulos = new Retangulo[3];
        retangulos[0] = new Retangulo(40, 20);
        retangulos[1] = new Retangulo(20, 20);
        retangulos[2] = new Retangulo(10, 30);
}
}
```

Generics

Quando recuperamos um objeto de uma coleção temos que usar o *cast*. Isso ocorre porque, na sua versão original, essas coleções trabalham com o tipo Object, e os métodos que retornam dados das coleções também retornam o tipo Object.

Trabalhar com o tipo Object, nesses caso, tem duas desvantagens:

1. Tem que sempre usar o *cast* para retornar ao tipo original que foi armazenado na coleção;
2. Possibilita a inserção de objetos "errados" nas coleções (por exemplo, o programador pode se enganar e inserir um objeto Turma em uma lista de Alunos).

O conceito de Generics veio para resolver esse problema. Podemos definir o tipo a ser armazenado nas coleções colocando o nome do tipo desejado entre **<** e **>** logo após o nome da coleção:

```
ArrayList<Aluno> alunos;

ArrayList<Float> notas;

ArrayList<String> palavras;

ArrayList<Veiculo> veiculos;//é possível armazenar carro ou caminhão nesta

//lista

ArrayList<Conta> contas;
```

Para criar a coleção colocamos os símbolos **<>** logo após o nome da coleção:

```
alunos = new ArrayList<>();

notas = new ArrayList<>();
```

```
palavras = new ArrayList<>();

veículos = new ArrayList<>();

contas = new ArrayList<>();
```

Importante:

Podemos retirar o <> que o código compilará normalmente, mas o compilador Java emitirá alguns avisos.

· Se tentarmos armazenar objetos de tipos diferentes daquele declarado, ocorrerá um erro de compilação.

· Ao recuperar um objeto da coleção não precisamos mais do *casting*, pois o compilador já sabe que tipo está armazenado na coleção.

Exemplo:

```
ArrayList<Aluno> alunos = new ArrayList<>();

alunos.add(new Turma(1, "POO")); //Erro de compilação: tipos incompatíveis

Aluno a = alunos.get(5); // Não precisamos do cast, pois o método get()

                         // retorna Aluno!
```

Estratégias de Aprendizagem

Realizar exercícios práticos que envolvam a criação e análise de programas que implementem vetores, vetores de objetos e ArrayList.

Indicação de Leitura Específica

Vetores

DEITEL, Paul. Java: como programar (Biblioteca Virtual). 10. ed. São Paulo: Pearson, 2017. [Páginas: 192 – 198]

Matrizes

DEITEL, Paul. Java: como programar (Biblioteca Virtual). 10. ed. São Paulo: Pearson, 2017. [Páginas: 213 – 216]

ArrayList

DEITEL, Paul. Java: como programar (Biblioteca Virtual). 10. ed. São Paulo: Pearson, 2017. [Páginas: 225 – 227]

Aplicação: articulação teoria e prática

Exercício:

1) Escreva um programa em Java para:

- ler o nome do curso.
- ler as notas de uma turma.
- imprimir a média da turma.
- imprimir as notas acima da média.

O programa deverá pedir, inicialmente, o total de notas que serão lidas.

2) Faça um programa em Java que crie uma lista de alunos, sabendo que qualquer aluno possui nome, matrícula e média. Depois de criada a lista, faça o que se pede:

- imprima todos os dados de todos os alunos.
- imprima os nomes dos alunos com média abaixo de 6.0.

Considerações Adicionais