POO I

Vinícius Machado

Vetores e Matrizes

Vetores

- Estruturas de dados homogêneas que permite guardar vários valores (dados) em um mesmo identificador.
- Utilizamos a posição para identificar qual dos dados queremos acessar.

vetor

vetor[0]	vetor[1]	vetor[2]	vetor[3]

Vetores

- Para declarar um vetor indicamos seu tipo e o tamanho.

```
int[ ] vetor = new int[4];
int vetor[ ] = new int[4];
int vetor[ ] = {1, 2, 3, 4};
```

vetor[0]	vetor[1]	vetor[2]	vetor[3]
0	0	0	0
1	2	3	4

Matrizes

- Matrizes são, assim como vetores, estruturas de dados homogêneas que armazenam valores num formato n dimensional.
- Normalmente utilizamos o formato bidimensional (linha, coluna). No entanto, algumas vezes precisamos guardar mais valores, podendo declarar matrizes de 2, 3, 4, …, n dimensões.

Matrizes

- O formato comumente adotado para declararmos uma matriz é da seguinte maneira:

```
int mat[ ][ ] = new int[4][5];

int[ ][ ] mat = new int[4][5];
```

mat[0][0]	mat[0][1]	mat[0][2]	mat[0][3]	mat[0][4]
mat[1][0]	mat[1][1]	mat[1][2]	mat[1][3]	mat[1][4]
mat[2][0]	mat[2][1]	mat[2][2]	mat[2][3]	mat[2][4]
mat[3][0]	mat[3][1]	mat[3][2]	mat[3][3]	mat[3][4]

Matrizes

- Ainda assim, é possível definirmos a quantidade de colunas após uma inicialização inicial das linhas. Por exemplo:

```
int values[ ][ ] = new int[5];
for (int i = 0; i < 5; i++) {
        values[ i ] = new int[ i * 3];
}
```

- Dessa forma podemos construirmos matrizes que não sejam necessariamente retangulares.

Vetores e Matrizes

- Tanto nas estruturas unidimensionais como n-dimensionais existe um atributo chamado length que permite acessar o tamanho dessa estrutura.
- Obs: tamanho != dados
- Dessa forma, podemos criar códigos que trabalhem com “qualquer” quantidade de dados sem precisarmos definir variáveis globais
    - Obs: variáveis globais são péssimas mas, raramente necessárias. Evitem!

Vetores - Exercício em aula

- Dado um vetor de números inteiros de tamanho definido pelo usuário, calcular:
  - Média
  - Mediana
  - Moda
  - Menor valor
  - Maior valor

- Que tal ordenarmos antes?

Matrizes - Exercícios em aula

- Crie um método capaz de printar uma matriz de qualquer tamanho.
- Faça um programa que chegue a estes padrões de desenho utilizando matrizes e utilize seu método para printá-las.

```
X O O O
X X O O
X X X O
X X X X
```

```
X X X X
X X X O
X X O O
X O O O
```

```
X X X X X
X O O O X
X O O O X
X X X X X
```

Classe Arrays

Como já falamos em aulas, JAVA possui muitas bibliotecas em sua API (veja a documentação). Estas bibliotecas aumentam nossa produtividade e diminuem a chance de erros em nossos programas.

A Classe Arrays possui vários métodos úteis para trabalharmos com vetores.

Arrays.sort(vetor[ ]);	Ordena um vetor em ordem crescente.
Arrays.fill (vetor[ ], value);	Preenche um vetor com o valor do parâmetro value.
Arrays.equal(vetor1[ ], vetor2[ ]);	Compara dois vetores retornando true/false
Arrays.binarySearch(vetor[ ], value);	Procura o valor value no vetor retornando sua posição. O vetor precisa estar ordenado. Caso o value não seja encontrado o resultado é menor que 0.

ArrayList

ArrayList

A API do Java fornece várias estruturas de dados pré definidas, chamadas coleções, utilizadas para armazenar grupos de objetos relacionados. Essas classes fornecem métodos eficientes que organizam, armazenam e recuperam seus dados sem que seja necessário conhecer como os dados são armazenados.

A classe de coleção ArrayList<T> do pacote java.util fornece uma solução conveniente em tempo de execução para acomodar elementos adicionais - Ela altera dinamicamente seu tamanho.

<T> é o tipo do ArrayList

ArrayList<Integer>
ArrayList<String>

ArrayList

- Para declarar um ArrayList

```
ArrayList<String> nomesTurma;
```

- Para inicializar

```
nomesTurma = new ArrayList<String>();
```

- Pode ser na mesma linha

```
ArrayList<String> nomesTurma = new ArrayList<String>();
```

ArrayList

Método	Descrição
`add`	Adiciona um elemento ao fim de `ArrayList`.
`clear`	Remove todos os elementos de `ArrayList`.
`contains`	Retorna `true` se `ArrayList` contiver o elemento especificado; do contrário, retorna `false`.
`get`	Retorna o elemento no índice especificado.
`indexOf`	Retorna o índice da primeira ocorrência do elemento especificado em `ArrayList`.
`remove`	Remove a primeira ocorrência do valor especificado.
`remove`	Remove o elemento no índice especificado.
`size`	Retorna o número de elementos armazenados no `ArrayList`.
`trimToSize`	Reduz a capacidade de `ArrayList` de acordo com o número de elementos atual.

Boas práticas de programação

- Um índice precisa ser um valor int ou um valor de um tipo que possa ser promovido a int - byte, short ou char. NÃO pode ser long, caso contrário ocorrerá um erro de compilação.
- ERRO COMUM:

  int [ ] a, b, c → 3 arrays
  int a[ ], b, c → 1 array e dois ints
- Vetores são sempre passados por referência para outros métodos

Arquivos

Arquivos - Objetivos

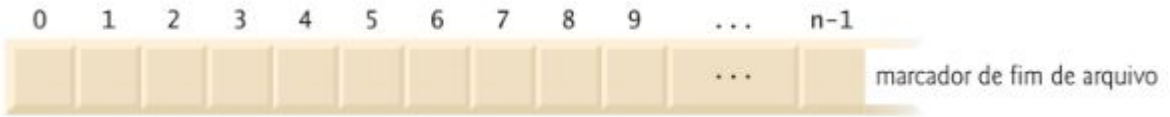
- Criar, ler, gravar e atualizar arquivos
- Recuperar informações sobre arquivos e diretórios usando os recursos das NIO.2 APIs
- Aprender as diferenças entre arquivos de texto e arquivos binários
- Utilização das classes
  - Scanner e Formatter para processar arquivos de texto
  - ObjectInputStream e ObjectOutputStream para ler e gravar objects em arquivos
  - Diálogo JFileChooser (Extra)

Arquivos

- Dados armazenados em variáveis e arrays são temporários — eles são perdidos quando uma variável local “sai do escopo” ou quando o programa termina.
- Para retenção de longo prazo dos dados, mesmo depois de os programas que os criaram serem fechados, os computadores usam arquivos.
- Os dados mantidos nos arquivos são dados persistentes.

Arquivos e Fluxos

- O Java vê cada arquivo como um fluxo de bytes sequencial.



  - Cada sistema operacional implementa sua forma de delimitar o fim do arquivo.
- Fluxos baseados em caracteres e em bytes
  - Fluxos baseados em bytes geram e inserem dados em um formato binário — um char tem dois bytes, um int tem quatro bytes, um double tem oito bytes etc.
  - Fluxos baseados em caracteres geram e inserem dados como uma sequência de caracteres na qual cada caractere tem dois bytes — o número de bytes para determinado valor depende do número de caracteres nesse valor.

Arquivos e Fluxos

Quando um programa Java começa a executar, ele cria três objetos de fluxo que estão relacionados com dispositivos — System.in, System.out e System.err.

- Entrada
  - O objeto System.in (fluxo de entrada padrão) normalmente permite que um programa insira bytes a partir do teclado.
- Saída
  - Já o objeto System.out (o objeto de fluxo de saída padrão), em geral, possibilita que um programa envie caracteres para a tela.
- Erro
  - o objeto System.err (o objeto de fluxo de erro padrão) na maioria das vezes autoriza que um programa gere mensagens de erro baseadas em caracteres e as envie para a tela.

Arquivos e Fluxos

- Os fluxos podem ser redirecionados em nosso código (para um arquivo de texto por exemplo).
- Para isso, a classe System fornece os métodos setIn, setOut, setErr para redirecionar os fluxos de entrada, saída e erro padrão, respectivamente.

** Java SE 8 adiciona outro tipo de fluxo (Capítulo 17 10ª Edição) -- Final do semestre veremos mais detalhes sobre as novidades do Java 8.

Arquivos - Obtendo informações

- As interfaces Path e DirectoryStream e as classes Paths e Files, todas do pacto java.nio.file, são úteis para recuperar informações sobre arquivos e diretórios no disco
- Interface Path
  - os objetos das classes que implementam essa interface representam o local de um arquivo ou diretórios. Objetos Path não abrem arquivos nem fornecem capacidades de processamento deles.
- Classe Paths
  - fornece métodos estáticos utilizados para obter um objeto Path representando um local de arquivo ou diretório
- Classe Files
  - fornece os métodos estáticos para manipulação de arquivos e diretórios comuns, como copiar arquivos, criar e excluir arquivos e diretórios, obter informações, ler o conteúdo dos arquivos, obter objetos que permitem manipular o conteúdo dos arquivos e mais...

Arquivos - Obtendo informações

- Classe DirectoryStream
  - Os objetos da classe que implementam essa interface possibilitam que um programa itere pelo conteúdo de um diretório.

- Caminhos absolutos x relativos
  - Absoluto: contém todos os diretórios, desde o diretório raiz.
  - Relativo: é relativo a outro diretório; por exemplo, um caminho relativo para o diretório em que o aplicativo começou a executar.

Arquivos - Exemplo

- Vamos utilizar as classes vistas anteriormente para produzir informações sobre o arquivo ou diretório digitado pelo usuário.

```
Path path = Path.get(text);

Files.exists(path);

path.getFileName();

Files.isDirectory(path);

path.isAbsolute();
```

```
Files.getLastModified(path);

Files.size(path);

path.toAbsolutePath();

DirectoryStream<Path> directoryStream;

Files.newDirectoryStream(path);
```

Arquivos - Exemplo

```
Digite o nome do arquivo ou diretório: C:\teste
teste existe
teste é um diretório
é um caminho absoluto
última modificação: 2013-11-08T19:50:00.838256Z
tamanho: 4096
Caminho: C:\teste
Caminho absoluto: C:\teste
Conteúdo do diretório:
C:\teste\pasta1
C:\teste\pasta2
C:\teste\pasta3
```

Arquivos

- Caractere separador
  - Um caractere separador é utilizado para separar diretórios e arquivos em um caminho. Em um computador Windows, o caractere separador é uma barra invertida (\). Em um sistema Linux ou Mac OS X, é uma barra (/). O Java processa esses dois caracteres de maneira idêntica em um nome de caminho. Por exemplo, o caminho abaixo é interpretado corretamente pelo Java

c:\Program Files\Java\jdk1.6.0_11\demo/jfc

**Ao criar Strings que representam informações de caminho, utilize File.separator para obter o caractere de separador adequado do computador local, em vez de utilizar explicitamente / ou \. Essa constante é uma String que consiste em um caractere — o separador apropriado para o sistema.

Boas práticas de programação

- Depois de confirmar que um Path existe, ainda é possível que os métodos lancem IOExceptions. Por exemplo, o arquivo ou diretório representado pelo Path pode ser excluído do sistema após a chamada para o método Files.exists e antes que as outras instruções sejam executadas. Programas de produção robustos com processamento de arquivos e diretórios exigem tratamento de exceção extenso para se recuperarem dessas possibilidades.

Boas práticas de programação

- Arquivos estão fortemente associados a exceções. Recomenda-se tratá-las de maneira correta e sempre que possível. Nesta altura das aulas ainda não vimos exceções mas queremos o poderio dos arquivos em nossas aplicações, por isso, em nosso método main poderá lançar tais exceções. Para isso, adicionaremos throws Exception no final de sua assinatura

```
public static void main(String[ ] args) throws Exception
```

Arquivos - Acesso sequencial

- Usaremos um Formatter para gerar Strings formatadas utilizando as mesmas capacidades de formatação que as do método System.out.printf. Um objeto Formatter pode gerar saída para vários locais, como para uma janela de comando ou um arquivo,
- O construtor do Formatter recebe como argumento uma String que contém o nome do arquivo.
    - Se o arquivo não existir ele é criado.
    - Se o caminho especificado for relativo, o arquivo será criado no mesmo diretório em que a aplicação começou a ser executada.

Arquivos - Acesso sequencial

- Criação do arquivo / abertura

```
Formatter saida = new Formatter(“data.txt”);
```

- Para escrever / adicionar dados no arquivo utilizaremos o método format do objeto Formatter

```
saida.format(“%s tem %d anos”, “Alexandre”, 14);
```

- Finalmente, ao final do uso, devemos fechar o arquivo para que ele possa ser aberto e editado em outros programas. Para isso:

```
saida.close();
```

Arquivos - Acesso Sequencial

- Além de ler dados dos arquivos, nossas aplicações devem ser capazes de lerem estes dados. Para ler sequencialmente de um arquivo podemos utilizar a classe Scanner (a mesma que lê dados do teclado).
- Para sua utilização, podemos passar para o construtor da classe Scanner o caminho do arquivo

```
Scanner inputFromFile = new Scanner(Paths.get(“data.txt”));
```

Arquivos - Acesso Sequencial

- Os dados em muitos arquivos sequenciais não podem ser modificados sem o risco de destruir outros.
- Por isso, os registros em um arquivo de acesso sequencial em geral não são atualizados no local; em vez disso, todo o arquivo é regravado.
- Como resolver este problema?
  - Serialização de objeto
  - Banco de dados

Arquivos - Serialização de Objetos

- Quando gravamos dados em arquivos, algumas informações são perdidas, como o tipo de dados de cada valor. Por exemplo, se guardarmos o valor 3 num arquivo de texto e mais tarde tentamos ler este dado, podemos guardar em um float, int, long, char… Não temos a informação correta de qual o tipo de valor.
- Outras vezes, queremos ler ou gravar um objeto em um arquivo ou em uma conexão de rede. O Java fornece a serialização de objetos para esse propósito.

Arquivos - Serialização de Objetos

- Um objeto serializado é representado como uma sequência de bytes que inclui os dados do objeto, bem como as informações sobre o tipo dele e a natureza dos dados armazenados nele. Depois que um objeto serializado foi gravado em um arquivo, ele pode ser lido a partir do arquivo e desserializado - em outras palavras, as informações dos tipos e bytes que representam o objeto e seus dados podem ser utilizadas para recriar o objeto na memória.

Arquivos - Serialização de Objetos

- As classes ObjectInputStream e ObjectOutputStream permitem que objetos inteiros sejam lidos ou gravados em um fluxo (possivelmente um arquivo).
- Para usar a serialização com arquivos, arquivos, inicializamos os objetos ObjectInputStream e ObjectOutputStream com objetos de fluxo que leem e gravam em arquivos. Esse tipo de inicialização de objetos de fluxo com outros objetos de fluxo é chamado, às vezes, de empacotamento — o novo objeto de fluxo em processo de criação empacota o objeto de fluxo especificado como um argumento do construtor.
- Utilizamos os métodos writeObject que recebe um Object como um argumento e grava suas informações em um OutputStream. O método readObject lê e retorna uma referência para o tipo real do objeto.

Arquivos - Serialização de Objetos

Escrevendo Objetos em Arquivos

```
ObjectOutputStream out;

out = new ObjectOutputStream(Files.newOutputStream(Paths.get("data.vfm")));

out.writeObject(myObject);

out.close();
```

Arquivos - Serialização de Objetos

Lendo objetos de arquivos serializados

```
ObjectInputStream input;

input = new ObjectInputStream(Files.newInputStream(Paths.get("data.vfm")));

MyObject obj = (MyObject) input.readObject();

input.close();
```

Arquivos - Serialização de Objetos

Lendo um número desconhecido de registros

```
try {
	while (true) {
		//leitura
	}
} catch (EOFException endOfFile) {
	//arquivo chegou ao final sem erros
}
```

Bônus - JFileChooser

JFileChooser

- A classe JFileChooser exibe uma caixa de diálogo que permite ao usuário selecionar facilmente arquivos ou diretórios.
- Utilizamos a constante JFileChooser static FILES_AND_DIRECTORIES a fim de indicar que arquivos e diretórios podem ser selecionados.
  - Outras constantes static incluem FILES_ONLY (o padrão) e DIRECTORIES_ONLY
- O método showOpenDialog serve para exibir o diálogo de JFileChooser
  - O argumento this especifica a janela pai do diálogo de JFileChooser, que estabelece a posição do diálogo na tela. Se null for passado, o diálogo será exibido no centro da tela — caso contrário, ele é centralizado na janela do aplicativo (determinado pelo argumento this)
- O método showOpenDialog retorna um inteiro especificando em qual botão (Open ou Cancel) o usuário clicou para fechar o diálogo

JFileChooser

- O método JFileChooser getSelectedFile recupera um objeto File (pacote java. io) que representa o arquivo ou diretório que o usuário selecionou, e
- O método File toPath retorna um objeto Path que contém o caminho selecionado pelo usuário.

```
import javax.swing.JFileChooser;
JFileChooser fileChooser = new JFileChooser();
fileChooser.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
int result = fileChooser.showOpenDialog(this);
fileChooser.getSelectedFile( ).toPath( );
```

Exercícios - Moodle

Leitura Recomendada

- Java: Como Programar. Capítulos 7, 17**.
  - Exercícios Capítulo 7, 17**.
- https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html
- https://docs.oracle.com/javase/8/docs/api/java/nio/file/Path.html
- https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html

** Obs: Na 10ª Ed o capítulo de arquivos é o 15º. Utilizaremos como referência pois foram introduzidas novas classes mais otimizadas para tratamento de arquivos. No entanto, as versões do Cap 17 da 8ª Ed continuam sendo muito utilizadas.

Referências

Livro

Java: Como programar.

Use a Cabeça! Java

Online

https://howtodoinjava.com

https://docs.oracle.com/javase/8/docs/

Contato

vinicius.machado@osorio.ifrs.edu.br