BandTec

DIGITAL SCHOOL

BandTec
DIGITAL SCHOOL

# ED

# Estrutura de Dados e Armazenamento

Arquivos

# Arquivos

- Dados armazenados em variáveis, arrays, ArrayLists ou Lists são temporários:
  - São perdidos quando uma variável local sai do escopo ou quando o programa termina.
- Para retenção de longo prazo dos dados, os computadores utilizam **arquivos**.
- Os computadores armazenam os arquivos em **dispositivos de armazenamento secundários**, como SSD, discos rígidos, unidades flash e fitas magnéticas.
- Os dados mantidos nos arquivos são **dados persistentes** porque existem além da duração da execução do programa.

# Registros e Campos

- Arquivo
  - Grupo de **registros** relacionados
  - Em geral, vários campos compõem um **registro** (implementado como uma classe em Java).
- Registro
  - Grupo de campos relacionados.
- Campo
  - Informação

# Registros e Campos

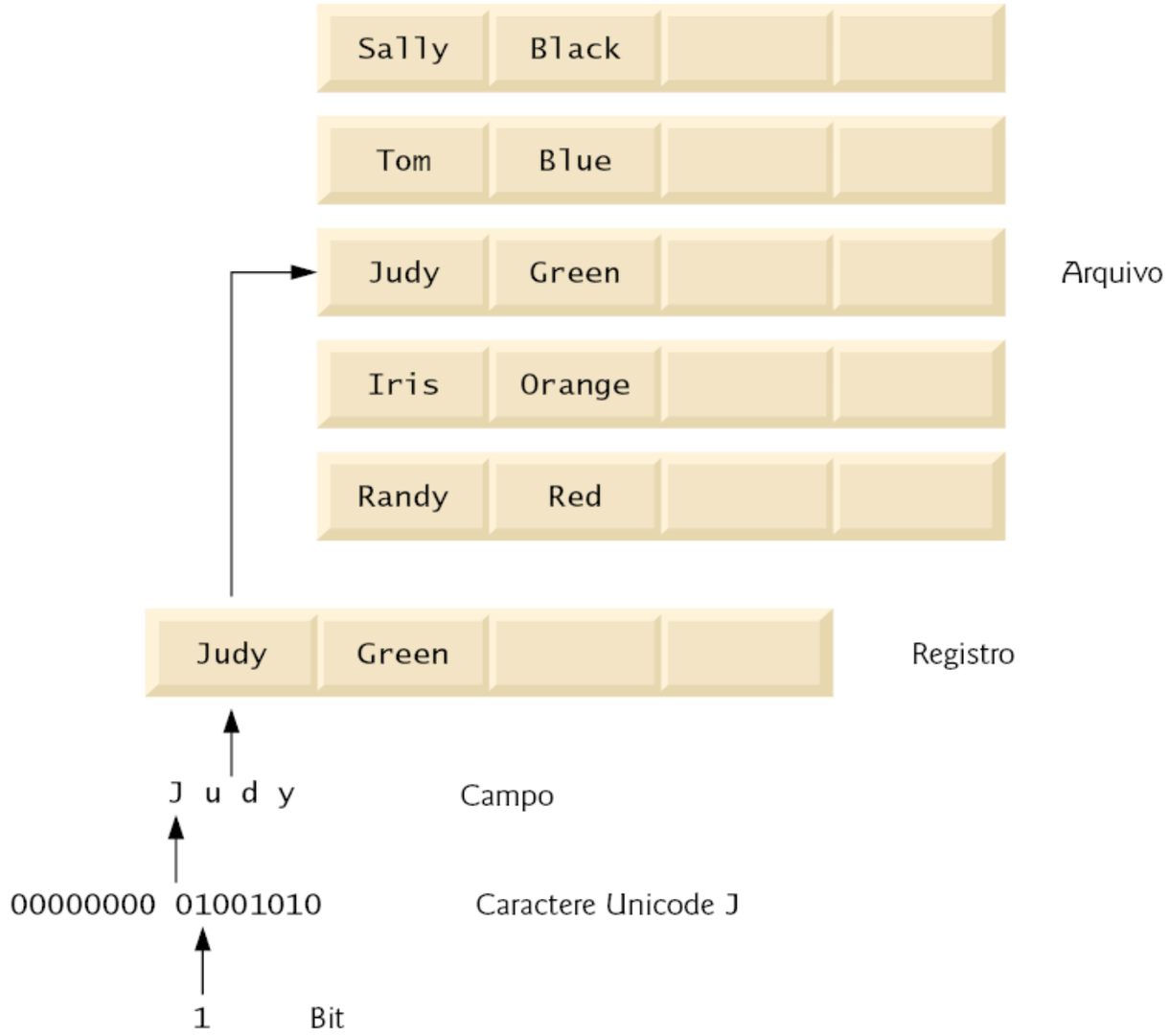


**Figura 17.1** | Hierarquia de dados.

# Chave de registro

- Para facilitar a recuperação de registros específicos de um arquivo, pelo menos um campo em cada registro é escolhido como uma chave de registro.
- Uma chave de registro identifica um registro de forma única (é semelhante ao conceito de chave primária em banco de dados)
- Costuma ser utilizada para pesquisar e classificar registros.
- Há muitas maneiras de organizar registros em um arquivo. A mais comum é chamada arquivo sequencial, em que os registros são armazenados em ordem pelo campo da chave de registro.

# Arquivos e Fluxos

- O Java visualiza cada arquivo como um fluxo sequencial de bytes
- Cada sistema operacional fornece um mecanismo para determinar o final de um arquivo, como um marcador de fim de arquivo ou uma contagem do total de bytes do arquivo que estão gravados na estrutura de dados administrativa mantida pelo sistema.
- Um programa Java simplesmente recebe uma indicação do sistema operacional quando ele alcança o final do fluxo.

# Marcador do fim de arquivo

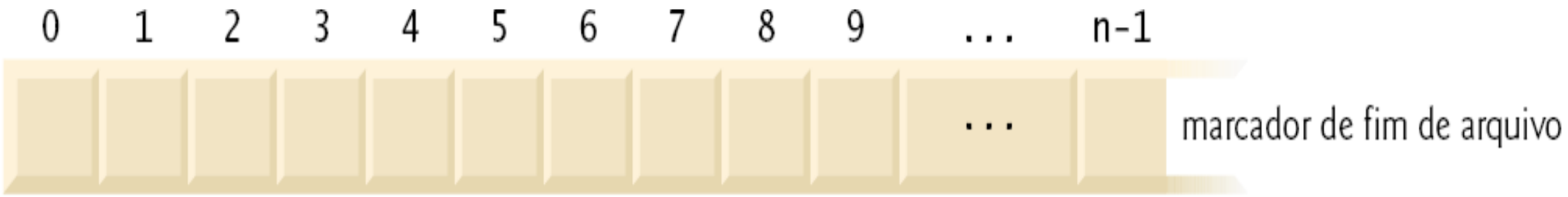


**Figura 17.2** | Visualização do Java de um arquivo de *n* bytes.

# Arquivos Binários e Arquivos Texto

- Fluxos de arquivos podem ser utilizados para entrada e saída de dados, como bytes ou caracteres:
  - Fluxos baseados em bytes – dados no formato binário
  - Fluxos baseados em caracteres – dados como sequência de caracteres.
- Arquivos binários:
  - Criados utilizando-se fluxos baseados em bytes
  - Lidos por programas que entendem o conteúdo específico do arquivo e a ordem desse conteúdo
- Arquivos texto
  - Criados com fluxos baseados em caracteres.
  - Podem ser lidos por editores de textos.

# Criando/Abrindo Arquivos Texto para Escrita

- Criar / abrir arquivo texto para escrita
  - Há várias formas de se criar ou abrir um arquivo texto para escrita
  - A forma que utilizaremos é criando um objeto da classe Formatter, fornecendo o nome do arquivo como uma String
  - Se o arquivo não existir, ele criará o arquivo com o nome fornecido. Se o arquivo já existir, ele abrirá o arquivo.
  - Mas, devido ao uso do try/catch, normalmente declaramos o objeto da classe Formatter fora do bloco try e criamos o objeto dentro do bloco try:

```
Formatter output= null; //declara a variável output fora do bloco try
try {
   output = new Formatter("arquivo.txt");//cria o objeto associado ao arq
}
catch ....
```

  - Erros tratados na criação/abertura do arquivo:
    - SecurityException – qdo não há permissão de escrita no arquivo
    - FileNotFoundException – erro ao abrir ou criar o arquivo

# Gravando Dados em Arquivos Texto

- Gravar dados em um arquivo texto
  - Para gravar dados em um arquivo texto, utilizaremos o método format da classe Formatter
  - A maneira de se utilizar o format é semelhante à maneira como se utiliza o método format da classe String (dentro do toString(), por exemplo)
  - Devemos passar uma String contendo o formato de gravação, seguido pelos dados a serem gravados, separados por vírgulas.
  - Por exemplo: supondo que output seja o objeto da classe Formatter criado associado ao arquivo texto, a instrução abaixo grava nesse arquivo o conteúdo das variáveis idDisc, nome, curso, semestre e nroAlunos:

```
output.format("%d %s %s %d %d\n",idDisc,nome,curso,semestre,nroAlunos);
```

  - Essa instrução deve estar dentro de um bloco try, para que os possíveis erros de gravação possam ser tratados.
  - Erros tratados:
    - FormatterClosedException – erro ao gravar no arquivo

# Fechando Arquivos Texto

- Fechar arquivo texto
  - Todo arquivo, após criado ou aberto, deve ser fechado quando não se deseja mais utilizá-lo.
  - Para se fechar um arquivo, utiliza-se o método close.
  - Por exemplo: supondo que output seja o objeto da classe Formatter criado associado ao arquivo texto:

```
output.close( );
```

- Verifique os códigos exemplos:
  - Disciplina.java
  - CriaArquivoTexto.java
  - LeArquivoTexto.java

# Abrindo Arquivos Texto para Leitura

- Abrir arquivo texto para leitura:
  - Há várias formas de se abrir um arquivo texto para leitura.
  - A forma que utilizaremos é criando um objeto da classe Scanner (a mesma classe que se utiliza para ler do teclado), fornecendo ao seu construtor um objeto da classe FileReader criado com o nome do arquivo a ser aberto
  - Ex:

```
Scanner input= null; //declara a variável input fora
                     //do bloco try
try {
   input = new Scanner(new FileReader("arquivo.txt"));
}
catch ....
```

  - Erro tratado:
    - FileNotFoundException – erro ao abrir arquivo

# Lendo dados de Arquivo Texto

- Ler dados de um arquivo texto
  - Para ler dados de um arquivo texto, utilizaremos os métodos next( ), nextInt( ), nextFloat( ), nextDouble( ) da classe Scanner, da mesma forma que utilizamos para ler do teclado
  - Por exemplo: supondo que input seja o objeto da classe Scanner criado associado ao arquivo texto, as instruções abaixo leem dados desse arquivo:

```
nome=input.next();
curso=input.next();
semestre=input.nextInt();
nroAlunos=input.nextInt();
```

  - Essas instruções devem estar dentro de um bloco try, para que os possíveis erros de leituras possam ser tratados.
  - Erros tratados:
    - NoSuchElementException – arquivo com problemas
    - IllegalStateException – erro na leitura do arquivo
  - Para saber quando se chegou ao final do arquivo, basta chamar o método hasNext() da classe Scanner. Quando o retorno for false, significa que chegou ao final do arquivo

# String.format ou System.out.printf

- O método format da classe String é utilizado para formatar uma String.
- O método printf foi implementado no Java para manter a compatibilidade com as linguagens C e C++
- Ambos os métodos recebem como argumento uma String que é a mensagem formatada. No lugar onde aparecerão os dados, dentro dessa String, colocamos os formatos que deverão ser aplicados aos dados. Após a String com a mensagem formatada, seguem as variáveis separadas por vírgulas.
- Por exemplo:

```
System.out.printf ("O nome é %s", nome);
System.out.printf ("A idade é %d e o salário é R$ %.2f",
                   idade, salario);
```

# Formatos

| | |
|---|---|
| %c | caracter |
| %d | número inteiro |
| %f | número real (com casas decimais) |
| %s | String |
| | |
| %.2d | máximo de 2 dígitos (número inteiro) |
| %6d | alinhado à direita, se o número tiver menos do que 6 dígitos, será preenchido com brancos à esquerda |
| %-6d | alinhado à esquerda e será preenchido com brancos à direita se tiver menos do que 6 dígitos |
| %06d | alinhado à direita e será preenchido com zeros à esquerda se tiver menos do que 6 dígitos |

# Lendo dados de Arquivo Texto tipo csv

- Os dados em um arquivo csv são separados por ';'
- Neste caso, também podemos usar o objeto da classe Scanner.
- Ao criar o objeto, porém :

```
Scanner input; // fora do try/catch
// dentro do try/catch:
 input = new Scanner(new FileReader("arquivo.txt"))
                          .useDelimiter(";|\\r\\n");
```

- Ler dados de um arquivo texto
  - Depois, basta usar os métodos next(), nextInt(), nextDouble(), etc

```
nome=input.next();
curso=input.next();
semestre=input.nextInt();
nroAlunos=input.nextInt();
```

  - Essas instruções devem estar dentro de um bloco try, para que os possíveis erros de leituras possam ser tratados.

# Serialização de objetos

- É possível, em Java, ler ou gravar um objeto inteiro (todos os dados de um objeto), de uma só vez, através da serialização de objetos.
- Um objeto serializado é uma sequência de bytes que inclui os dados dos atributos do objeto e as informações sobre seus tipos.
- Um objeto serializado pode ser gravado em um arquivo ou enviado pela rede para outro aplicativo.
- Depois, os dados podem ser lidos do arquivo ou recebidos em outro aplicativo (caso tenha sido enviado pela rede) e desserializados, recriando-se o objeto na memória (como se criasse o objeto novamente, inicializado com os dados lidos).

- **Atenção**: dessa forma, o arquivo não é texto

# Serialização de objetos

- Para que o objeto de uma classe seja serializável, deve-se especificar na declaração da classe as palavras **implements Serializable**
- Veja o exemplo DisciplinaSerializavel.java

# Criando/Abrindo Arquivos para Gravar Objeto Serializável

- Criar / abrir arquivo para escrita de objeto serializável
  - A forma que utilizaremos é criando um objeto da classe ObjectOutputStream, fornecendo um objeto da classe FileOutputStream, que é criado fornecendo o nome do arquivo
  - Se o arquivo não existir, ele criará o arquivo com o nome fornecido. Se o arquivo já existir, ele abrirá o arquivo.
  - Exemplo:

```
ObjectOutputStream output= null;  //declara a variável output fora do bloco try
try {
  output = new ObjectOutputStream(new FileOutputStream("arquivo.ser"));
}
catch ....
```

  - Erros tratados na criação/abertura do arquivo:
    - IOException – erro ao abrir o arquivo

# Gravando Objeto Serializável em Arquivo

- Gravar objeto serializável em arquivo
  - Para gravar objeto serializável em um arquivo texto, utilizaremos o método writeObject da classe ObjectOutputStream, fornecendo o nome do objeto que se quer gravar (este objeto deve ser de uma classe marcada como Serializable)
  - Por exemplo: supondo que output seja o objeto da classe ObjectOutputStream criado associado ao arquivo e que obj seja o objeto que se deseja gravar:

```
output.writeObject(obj);
```

  - Essa instrução deve estar dentro de um bloco try, para que os possíveis erros de gravação possam ser tratados.
  - Erros tratados:
    - IOException – erro ao gravar no arquivo

# Fechando arquivo utilizado para gravar Objeto Serializável

- Fechar arquivo
  - Todo arquivo, após criado ou aberto, deve ser fechado quando não se deseja mais utilizá-lo.
  - Para se fechar um arquivo, utiliza-se o método close.
  - Por exemplo:

```
output.close( );
```

- Verifique os códigos exemplos:
  - DisciplinaSerializable.java
  - CriaArquivoSerializavel.java
  - LeArquivoSerializavel.java

# Abrindo arquivo com dados de objetos serializáveis para leitura

- Abrir arquivo para desserializar objetos:
  - A forma que utilizaremos é criando um objeto da classe ObjectInputStream, fornecendo ao seu construtor um objeto da classe FileInputStream criado com o nome do arquivo a ser aberto
  - Ex:

```
ObjectInputStream input= null; //declara a variável input fora
                               //do bloco try
try {
 input=new ObjectInputStream(new FileInputStream("arquivo.ser"));
}
catch ....
```

  - Erro tratado:
    - IOException – erro ao abrir arquivo

# Desserializando objetos de arquivos

- Desserializar objetos de um arquivo
  - Para desserializar objetos de um arquivo, utilizaremos o método readObject( ) da classe ObjectInputStream. Esse método criará um objeto da classe fornecida entre parênteses antes da chamada do método, e inicializará os atributos do objeto criado com os valores lidos do arquivo
  - Por exemplo: supondo que input seja o objeto da classe ObjectInputStream criado associado ao arquivo com dados gravados do objeto serializável, a instrução abaixo desserializa o objeto:

```
obj=( nome da classe) input.readObject();
```

  - Essa instrução deve estar dentro de um bloco try, para que os possíveis erros possam ser tratados.
  - Erros tratados:
    - EOFException – indica que chegou ao final do arquivo
    - IOException – erro durante leitura do arquivo
    - ClassNotFoundException – erro ao criar o objeto

# Referências bibliográficas

- Esse material foi elaborado com base nos livros:
  - Java: como programar. Deitel, Paul; Deitel, Harvey. 10ª edição. São Paulo: Pearson Education do Brasil. 2017.

# Obrigada!

BandTec DIGITAL SCHOOL

Em caso de dúvidas, entre em contato com:
celia.taniwaki@bandtec.com.br