Departamento de Engenharia de Software
Centro de Educação Superior do Alto Vale do Itajaí - CEAVI
Universidade do Estado de Santa Catarina - UDESC

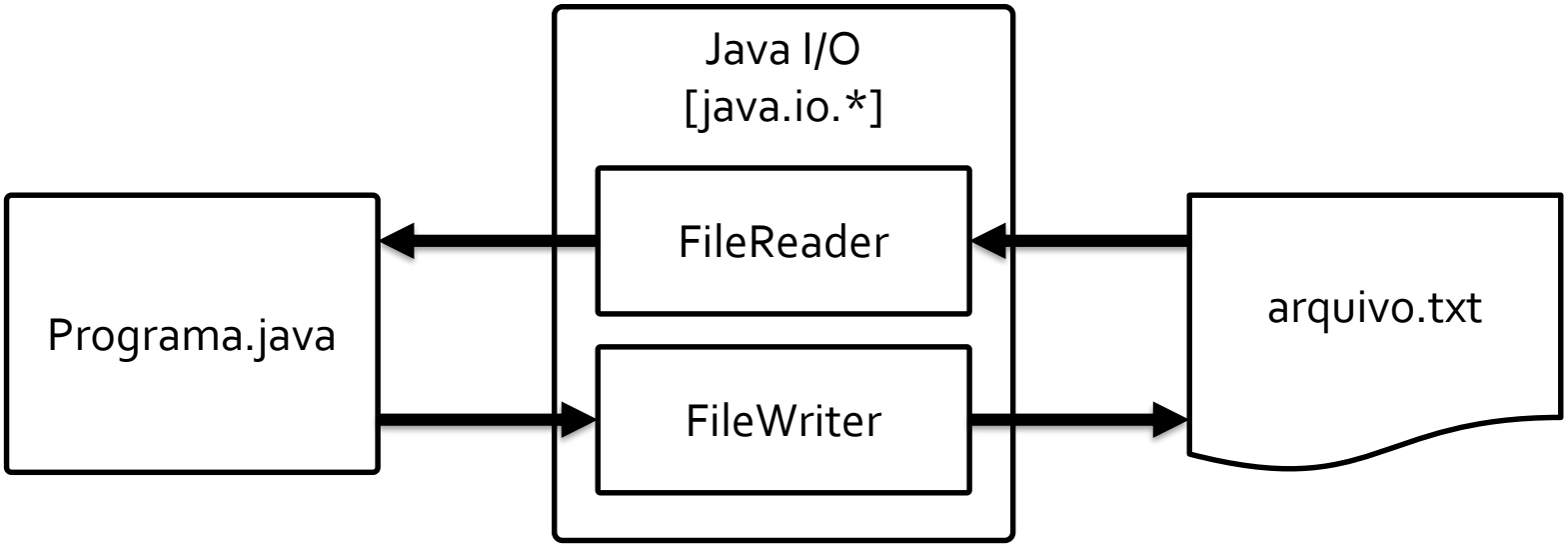
Tópicos adicionais
Persistência em arquivo texto

Persistência em arquivo texto

- Aplicações precisam de algum mecanismo de persistência de dados, pois informações armazenadas na memória são perdidas sempre que a aplicação é encerrada.
- A forma mais simples de fazer isso em Java é salvar os dados em arquivos.
- As operações de leitura e escrita de dados em arquivos são feitas através das bibliotecas de entrada e saída do Java (ou **bibliotecas I/O** – Input/Output).

Persistência em arquivo texto

- Aplicações precisam de algum mecanismo de persistência de dados, pois informações armazenadas na memória são perdidas sempre que a aplicação é encerrada.
- A forma mais simples de fazer isso em Java é salvar os dados em arquivos.
- As operações de leitura e escrita de dados em arquivos são feitas através das bibliotecas de entrada e saída do Java (ou **bibliotecas I/O** – Input/Output).



Operação de escrita

- A classe **FileWriter** fornece os métodos necessários para a escrita de dados em arquivos. Na sua criação, é passado o nome do arquivo e o argumento **append**. Se verdadeiro, caso existam informações no arquivo, elas são mantidas e os novos dados são inseridos no final do arquivo. Se falso, caso existam informações no arquivo, elas são apagadas e os novos dados são inseridos no seu lugar.

```
try {
    FileWriter writer = new FileWriter("pessoas.txt", true);
    writer.write("Este texto será inserido no arquivo");
    writer.write("\n");
    writer.close();
} catch (IOException ex) {
    Logger.getLogger(PersistenciaArquivos.class.getName()).log(Level.SEVERE, null, ex);
}
```

- O método **write("texto")** escreve o valor ("texto") no arquivo vinculado ao writer (neste exemplo, "pessoas.txt").
- É necessário fechar o objeto **writer** após seu uso através do método **close()**.

Operação de escrita

- A classe **FileWriter** fornece os métodos necessários para a escrita de dados em arquivos. Na sua criação, é passado o nome do arquivo e o argumento **append**. Se verdadeiro, caso existam informações no arquivo, elas são mantidas e os novos dados são inseridos no final do arquivo. Se falso, caso existam informações no arquivo, elas são apagadas e os novos dados são inseridos no seu lugar.

```
try {
    FileWriter writer = new FileWriter("pessoas.txt", true);
    writer.write("Este texto será inserido no arquivo");
    writer.write("\n");
    writer.close();
} catch (IOException ex) {
    Logger.getLogger(PersistenciaArquivos.class.getName()).log(Level.SEVERE, null, ex);
}
```

- O método **write("texto")** escreve o valor
(neste exemplo, "pessoas.txt").

- É necessário fechar o objeto **writer** após

É necessário circundar o código com uma instrução **try-catch**, uma vez que uma exceção pode ser disparada na abertura do arquivo.

Operação de leitura

- A classe **FileReader** fornece métodos para abertura de arquivos e leitura dos seus dados. Porém, seus métodos permitem apenas a leitura de caracteres. Para ler os dados por linha, utilizaremos um objeto da classe **BufferedReader**. Na criação, instanciamos um novo **FileReader** passando o nome do arquivo.

```
try {
    BufferedReader reader = new BufferedReader(new FileReader("pessoas.txt"));
    String linha;
    while((linha = reader.readLine()) != null) {
        System.out.println(linha);
    }
    reader.close();
} catch (FileNotFoundException ex) {} catch (IOException ex) {}
```

- O método **readLine()** faz a leitura de toda a linha e passa para a próxima. Após ler a última linha do arquivo, o método devolve **null**. Por isso, enquanto a linha lida for diferente de nulo, a linha é impressa em tela. Após o término, o **reader** é fechado.
- Assim como na operação de escrita, o código deve ser circundado (**try-catch**).

Exemplo – armazenamento de pessoas

- Classe **Pessoa**

```
public class Pessoa {
    private String nome;
    private int idade;
    private char sexo;

    //Métodos construtores omitidos

    public String toWriteString() {
        return nome + ";" + idade + ";" + sexo;
    }

    public String toString() {
        String genero;
        if(sexo == 'M') genero = "masculino";
        else genero = "feminino";

        return nome + ", possui " + idade + " anos de idade e é do sexo " + genero + ".";
    }

    //...
}
```

Exemplo – armazenamento de pessoas

- Classe **Pessoa**

```
public class Pessoa {
    private String nome;
    private int idade;
    private char sexo;

    //Métodos construtores omitidos

    public String toWriteString() {
        return nome + ";" + idade + ";" + sexo;
    }

    public String toString() {
        String genero;
        if(sexo == 'M') genero = "masculino";
        else genero = "feminino";

        return nome + ", possui " + idade + " anos de idade e é do sexo " + genero + ".";
    }

    //...
}
```

O método **toWriteString()** retorna o texto que será usado para armazenar a pessoa no arquivo, com os atributos separados por "**;**".

Classe para persistência dos dados

- O método **inserir** armazena uma pessoa no arquivo. O método **ler** recupera uma pessoa do arquivo pelo nome. O método **lerTodasPessoas** devolve uma lista com todas as pessoas armazenadas no arquivo. O método **excluir** remove uma pessoa do arquivo pelo nome.

```
public class PersistenciaArquivos {

    public static void excluir(String nome) {
    //...
    }

    public static List<Pessoa> lerTodasPessoas() {
    //...
    }

    public static Pessoa ler(String nome) {
    //...
    }

    public static void inserir(Pessoa p){
        //...
    }
}
```

Exemplo – escrita de uma pessoa

- O conteúdo retornado pelo método **toWriteString()** é armazenado no arquivo.
- Cada pessoa é escrita em uma linha, pois após armazenada a pessoa é armazenada uma quebra de linha ("**\n**").

```
public static void inserir(Pessoa p){
    try {
        FileWriter writer = new FileWriter("pessoas.txt", true);
        writer.write(p.toWriteString());
        writer.write("\n");
        writer.close();
    } catch (IOException ex) {
        Logger.getLogger(PersistenciaArquivos.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Exemplo – leitura de uma pessoa

- O arquivo é lido linha por linha, pois cada linha armazena uma pessoa diferente.

```
public static Pessoa ler(String nome) {
    try {
        BufferedReader reader = new BufferedReader(new FileReader("pessoas.txt"));
        String linha;
        while((linha = reader.readLine()) != null) {
            String[] conteudo = linha.split(";");
            if(conteudo[0].equals(nome)) {
                Pessoa p = new Pessoa();
                p.setNome(conteudo[0]);
                p.setIdade(Integer.parseInt(conteudo[1]));
                p.setSexo(conteudo[2].charAt(0));
                reader.close();
                return p;
            }
        }
        reader.close();
    } catch (FileNotFoundException ex) {} catch (IOException ex) {}

    return null;
}
```

Exemplo – leitura de uma pessoa

- O método **split** divide a String conforme o caracter recebido (no exemplo, é usado o caracter "**;**"). Ele separa cada valor e adiciona em uma posição de um vetor de String, retornando-o.

- **Exemplo:**

```
String texto = "Este texto;está separado;para posterior;recuperação!";
String[] conteudo = texto.split(";");
```

- **Resultado:**
  - conteudo[0] → "Este texto"
  - conteudo[1] → "está separado"
  - conteudo[2] → "para posterior"
  - conteudo[3] → "recuperação!"

Exemplo – leitura de uma pessoa

- O arquivo é lido linha por linha, pois cada linha armazena uma pessoa diferente.

```
public static Pessoa ler(String nome) {
    try {
        BufferedReader reader = new BufferedReader(new FileReader("pessoas.txt"));
        String linha;
        while((linha = reader.readLine()) != null) {
            String[] conteudo = linha.split(";");
            if(conteudo[0].equals(nome)) {
                Pessoa p = new Pessoa();
                p.setNome(conteudo[0]);
                p.setIdade(Integer.parseInt(conteudo[1]));
                p.setSexo(conteudo[2].charAt(0));
                reader.close();
                return p;
            }
        }
        reader.close();
    } catch (FileNotFoundException ex) {} catch (IOException ex) {}

    return null;
}
```

O nome está sempre na primeira posição da String separada, enquanto a idade está na segunda e o sexo está na terceira.

Exemplo – leitura de uma pessoa

- O arquivo é lido linha por linha, pois cada linha armazena uma pessoa diferente.

```
public static Pessoa ler(String nome) {
    try {
        BufferedReader reader = new BufferedReader(new FileReader("pessoas.txt"));
        String linha;
        while((linha = reader.readLine()) != null) {
            String[] conteudo = linha.split(";");
            if(conteudo[0].equals(nome)) {
                Pessoa p = new Pessoa();
                p.setNome(conteudo[0]);
                p.setIdade(Integer.parseInt(conteudo[1]));
                p.setSexo(conteudo[2].charAt(0));
                reader.close();
                return p;
            }
        }
        reader.close();
    } catch (FileNotFoundException ex) {} catch (IOException ex) {}

    return null;
}
```

Cada atributo é recuperado e um objeto **Pessoa** é montado e retornado como resultado do método de leitura.

Exemplo – leitura de uma pessoa

- O arquivo é lido linha por linha, pois cada linha armazena uma pessoa diferente.

```
public static Pessoa ler(String nome) {
    try {
        BufferedReader reader = new BufferedReader(new FileReader("pessoas.txt"));
        String linha;
        while((linha = reader.readLine()) != null) {
            String[] conteudo = linha.split(";");
            if(conteudo[0].equals(nome)) {
                Pessoa p = new Pessoa();
                p.setNome(conteudo[0]);
                p.setIdade(Integer.parseInt(conteudo[1]));
                p.setSexo(conteudo[2].charAt(0));
                reader.close();
                return p;
            }
        }
        reader.close();
    } catch (FileNotFoundException ex) {} catch (IOException ex) {}

    return null;
}
```

Neste caso, a pessoa é retornada apenas quando o nome for o buscado.

Exemplo – leitura de todas as pessoas

- A leitura é igual à de uma pessoa, mas todas as linhas são lidas, atribuídas ao objeto **Pessoa** e incluídos na lista, que é devolvida ao final do método.

```
public static List<Pessoa> lerTodasPessoas() {
    List<Pessoa> listaPessoas = new ArrayList<Pessoa>();
    try {
        BufferedReader reader = new BufferedReader(new FileReader("pessoas.txt"));
        String linha;
        while((linha = reader.readLine()) != null) {
            String[] conteudo = linha.split(";");
            Pessoa p = new Pessoa();
            p.setNome(conteudo[0]);
            p.setIdade(Integer.parseInt(conteudo[1]));
            p.setSexo(conteudo[2].charAt(0));
            listaPessoas.add(p);
        }
        reader.close();
    } catch (FileNotFoundException ex) {} catch (IOException ex) {}

    return listaPessoas;
}
```

Exemplo – exclusão de uma pessoa

- Como não existe um método para excluir uma linha única de um arquivo, a estratégia de exclusão consiste em ler todos os dados, apagar todo o arquivo e reescrever todos os registros novamente, exceto aquele que deve ser excluído.

```
public static void excluir(String nome) {
    List<Pessoa> todasPessoas = lerTodasPessoas();
    String conteudo = "";
    for(Pessoa p : todasPessoas) {
        if(!p.getNome().equals(nome))
            conteudo += p.toWriteString() + "\n";
    }
    try {
        FileWriter writer = new FileWriter("pessoas.txt", false);
        writer.write(conteudo);
        writer.close();
    } catch (IOException ex) {
        Logger.getLogger(PersistenciaArquivos.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

- Uma estratégia similar pode ser adotada na edição de dados, onde todos os registros são reescritos, atualizando os atributos do registro buscado.

Referências

DEITEL, H. M. **Java: como programar**. H. M Deitel e P. J. Deitel - 8a ed. Porto Alegre: Prentice-Hall, 2010.

Leitura complementar

TutorialsPoint Java (http://www.tutorialspoint.com/java).