Programação Orientada à Objetos (POO)

CIÊNCIA DA COMPUTAÇÃO

Prof. Dr. João Paulo Aramuni

UNIVERSIDADE FUMEC

Sumário

* **Arquivos**
  * Leitura
  * Gravação

Arquivos

* Para representar arquivos ou diretórios através da Classe **File:**

```
File a1 = new File("arq1.txt");
File a2 = new File("/pasta", "arq2.txt");
File d = new File("/pasta");
File a3 = new File(d, "arq3.txt");
```

Arquivos

* Possui métodos úteis para manipulação:

```
canRead(), canWrite(), createNewFile(), delete(),
exists(), getName(), getParentFile(), getPath(),
isDirectory(), isFile(), isHidden(),
lastModified(), length(), list(), listFiles(),
mkdir(), mkdirs(), renameTo(), setLastModified(),
setReadOnly(), etc.
```

Arquivos

* **Entrada e Saída**
  * Em Java, um objeto do qual se pode ler uma sequência de bytes é chamado de um fluxo de entrada (implementado pela classe abstrata **InputStream**).
  * Um objeto no qual se pode escrever uma sequência de bytes é chamado de um fluxo de saída (implementado pela classe abstrata **OutputStream**).
  * Esses objetos fonte e destino de sequências de bytes são normalmente arquivos, mas também podem ser conexões de rede e mesmo blocos de memória.
    * vantagem dessa generalidade: informações armazenadas em arquivos, por exemplo, são tratadas essencialmente da mesma forma que aquelas obtidas de uma conexão de rede.

Arquivos

* **Entrada e Saída**
  * Os dados, no final das contas, são armazenados como uma série de bytes, mas pode-se pensar neles, em um nível mais alto, como sendo uma sequência de caracteres ou de objetos.
  * Java fornece uma hierarquia de classes de entrada e saída cuja base são as classes InputStream e OutputStream.
  * Pode-se manipular fluxos de diversos formatos: compactados, textos, registros de tamanho fixo com acesso aleatório e objetos.
  * A seguir, será apresentada uma forma para se manipular fluxos de objetos usando o mecanismo de **serialização de objetos**.

Arquivos

* **Fluxo de Objetos**
  * Usando-se as classes ObjectOutputStream e ObjectInputStream pode-se gravar e ler objetos em um fluxo, de forma automática, através de um mecanismo chamado serialização de objetos.

  * Classe java.io.ObjectOutputStream:
    * ObjectOutputStream (OutputStream saida)
      * cria um ObjectOutputStream de modo que se possa escrever objetos no OutputStream (fluxo de saída) especificado.
    * void **writeObject** (**Object** obj)
      * escreve o objeto especificado no ObjectOutputStream. A classe do objeto, a assinatura da classe e os valores dos campos não marcados como estáticos da classe são escritos e de todas as suas superclasses.

Arquivos

- **Fluxo de Objetos**
  - Classe java.io.**ObjectInputStream:**
    - **ObjectInputStream** (**InputStream** entrada)
      - cria um ObjectInputStream para ler informações de objetos do InputStream (fluxo de entrada) especificado.
    - Object **readObject** ()
      - lê um objeto do ObjectInputStream. Em particular, lê a classe do objeto, a assinatura da classe e os valores dos campos **não estáticos** da classe e de todas as suas superclasses. Ele faz a desserialização para permitir que múltiplas referências de objetos possam ser recuperadas.
  - Classe java.io.**FileInputStream:**
    - **FileInputStream** (**String** nome)
      - cria um novo fluxo de entrada de arquivo, usando o arquivo cujo nome de caminho é especificado pela string nome.

Arquivos

- **Fluxo de Objetos**
  - Classe java.io.**FileOutputStream**:
    - **FileOutputStream** (**String** nome)
      - cria um novo fluxo de saída de arquivo especificado pela string nome. Os nomes de caminhos que não são absolutos são interpretados como relativos ao diretório de trabalho.
      - **Atenção**: apaga automaticamente qualquer arquivo existente com esse nome.
    - **FileOutputStream** (**String** nome, **boolean** anexar)
      - cria um novo fluxo de saída de arquivo especificado pela string nome. Os nomes de caminhos que não são absolutos são interpretados como relativos ao diretório de trabalho. Se o parâmetro anexar for true, então os dados serão adicionados ao final do arquivo. Um arquivo existente com o mesmo nome não será apagado.

Arquivos

* **Fluxo de Objetos**
  * Passos para salvar dados de um objeto:
    * abra um objeto da classe ObjectOutputStream

      ObjectOutputStream out = new ObjectOutputStream (new FileOutputStream("empregado.dat");
    * salve os objetos usando o método writeObject

      Empregado e1 = new Empregado("João",2500, new Day(1996,12,10));
      Gerente g1 = new Gerente("Maria",3800, new Day(1995,10,15);
      out.writeObject(e1);
      out.writeObject(g1);
  * Passos para ler dados de um objeto:

    abra um objeto da classe ObjectInputStream

    ObjectInputStream in = new ObjectInputStream (new FileInputStream("empregado.dat"));

Arquivos

* **Fluxo de Objetos**
  * leia os objetos usando o método readObject

```
Empregado e1 = (Empregado) in.readObject();
Gerente g1 = (Gerente) in.readObject();
```

  * Os objetos são gravados como Object e podem ser convertidos para o seu tipo original usando cast.

  * Para cada chamada de readObject() , lê um objeto na mesma ordem em que foram salvos. Para que uma classe possa ser gravada e restaurada num fluxo de objetos é necessário que ela implemente a interface Serializable.
    * Esta interface não tem métodos, portanto não é necessário alterar nada na classe.
    * public class Empregado implements Serializable { . . . }

Arquivos

- **Fluxo de Objetos**
  - Pode-se **salvar/restaurar um array de objetos** em uma única operação:

```
Empregado[ ] emp = new Empregado[3];
. . .
out.writeObject(emp);
. . .
Empregado[ ]  empNovo = (Empregado[ ]) in.readObject( );
```

  - Ao salvar um objeto que contém outros objetos, os mesmos são gravados automaticamente.
    - quando um objeto é compartilhado por vários outros objetos, somente uma cópia é gravada.
      - A técnica de **serialização** atribui um número de série a cada objeto. Após um objeto ser gravado, uma outra cópia compartilhada apenas recebe uma referência ao número de série já gravado anteriormente.

Arquivos

* **Fluxo de Objetos - Exemplo**

```
/** Clube - Classe para controlar os sócios do clube
* @author Aramuni
* @version 1.0
*/ @since – agosto de 2017
public class Clube {
    // Atributos
    private String nomeClube;
    private Vector <Socio> listaSocios = new Vector <Socio> ( );
```

Arquivos

* **Fluxo de Objetos - Exemplo**

```
/** Socio - Classe controlar os dados dos socios e a lista de seus dependentes
* @author Aramuni
* @version 1.0
*/ @since – agosto de 2017
public class Socio implements Serializable {
    // Atributos
    private String cpf;
    private String nome ;
    private String endereco;
    private GregorianCalendar dataEntrada;
    private GregorianCalendar dataSaida;
    private Vector <Dependente> listaDependentes = new Vector
<Dependente>( );
```

Arquivos

* **Fluxo de Objetos - Exemplo**

```
/** Dependente - Classe para controlar
* os dados do dependente
* @author Aramuni
* @version 1.0
*/ @since – agosto de 2017
public class Dependente implements Serializable {
   // Atributos
   private String nome;
   private String tipodependencia;
```

Fluxo de Objetos - Exemplo

```
public static void main ( String[ ] args )
    {
            InterfaceClube interfClub = new InterfaceClube();
            try    //  Capturar  Exceção
            {
            interfClub.club.leDados ( );
            }
            catch ( Exception erro ) {     //  Verificar Exceção
            System.out.println ("Erro na leitura do arquivo: " + erro.getMessage( ) );
            }
            interfClub.menu ( );
            try
            {
            interfClub.club.gravaDados ( );
            }
            catch ( Exception erro ) {
            System.out.println ("Erro na gravação do arquivo: " + erro.getMessage( ) );
            }
            System.out.println ( "Programa Encerrado !“ );
    }
```

Arquivos

* **Arquivo Texto - Gravação**
  * Classes **BufferedWriter** e **FileWriter :**

  * Principais Métodos:
    * **BufferedWriter** out = new BufferedWriter (new FileWriter (nomeArquivo))

      Abertura de Arquivo de texto para gravação
    * **Write**(String).

      Gravar uma linha no arquivo texto
    * **newLine**()

      Gravar um \n (line feed).
    * **close()**

      Fechamento do arquivo

Arquivo Texto – Gravação – Exemplo

```
private void gravaArqTexto() throws IOException {
 BufferedWriter out = new BufferedWriter(new FileWriter("cliente.txt"));
 String codigo = "";   // Tamanho  = 5
 String nome = "";    // Tamanho  = 30
 String email = "";    // Tamanho  = 60
 String telefone = ""; // Tamanho  = 12    // Tamanho da linha = 107
 for (int i = 0 ; i < clientes.size(); i++) {
 codigo = String.valueOf (((Cliente)clientes.get(i)).getCodigo());
 nome = ((Cliente)clientes.get(i)).getNome();
 telefone = ((Cliente)clientes.get(i)).getTelefone();
 email = ((Cliente)clientes.get(i)).getEmail();
 for (int j=0 ; codigo.length()<5  ; j++) codigo = "0" + codigo;
 for (int j=0; nome.length()<30; j++) nome += " ";
 for (int j=0; telefone.length()<12; j++) telefone += " ";
 for (int j=0; email.length()<60; j++) email += " ";
 out.write(codigo + nome + telefone + email); // Gravar linha
 out.newLine(); // Gravar /r /n
 }
 out.close();
 }
```

Arquivos

* **Arquivo Texto - Leitura**
  * Classes **BufferedReader** e **FileReader:**

  * Principais Métodos:
    * **BufferedReader** in = new BufferedReader (new FileReader (nomeArquivo))

      Abertura de Arquivo de texto para leitura
    * **readLine**().

      ler uma linha no arquivo texto
    * **read(array de char)** retorna -1 para EOF
    * **close()**

      Fechamento do arquivo

Arquivos

* **Arquivo Texto – Leitura – Exemplo**

```
private void lerArqTexto() throws IOException {
 BufferedReader in = new BufferedReader (new FileReader("cliente.txt"));
 char[] c = new char[109]; // Tamanho da Linha + 2 - /r/n
 while (in.read(c)!=-1) {
 String linha = "";
 for (int i = 0; i <c.length;i++) linha = linha + c[i];
 System.out.println("Código: " + linha.substring(0,5));
 System.out.println("Nome: " + linha.substring(5,35));
 System.out.println("Telefone: " + linha.substring(35,47));
 System.out.println("Email: " + linha.substring(47));
 Console.readLine("<Enter>");
 }
 in.close();
 }
```

Arquivos

* Para dados de objetos, evite utilizar arquivos **.txt**.

* Ao invés disso, utilize a gravação em arquivos **.dat**.

Obrigado.

joaopauloaramuni@gmail.com
joaopauloaramuni@fumec.br

UNIVERSIDADE FUMEC
50 Anos