

# Programação Orientada a Objetos – Aula 10

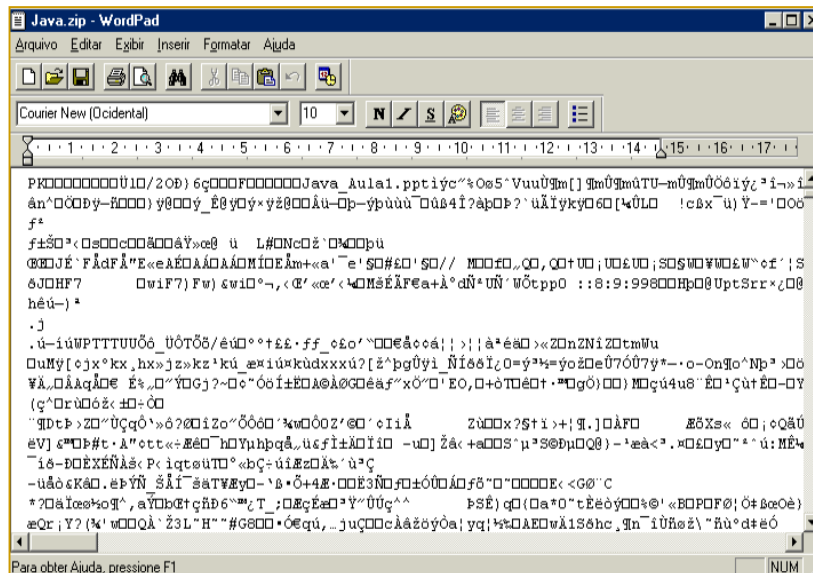
Prof. Dr. Eduardo Takeo Ueda  
*[eduardo.tueda@sp.senac.br](mailto:eduardo.tueda@sp.senac.br)*

# Arquivos e Fluxos

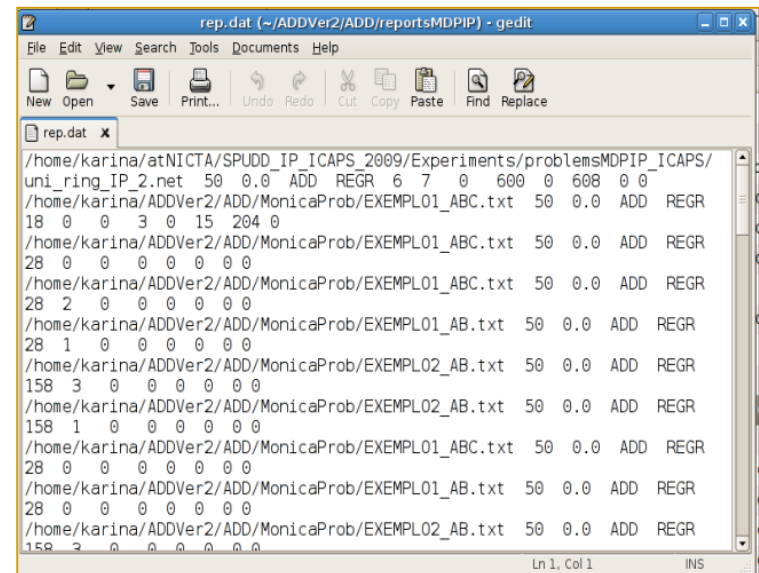
- Programadores/desenvolvedores utilizam arquivos para armazenar dados a longo prazo
- Java vê cada arquivo como um fluxo sequencial (existe um marcador de fim de arquivo)
- O termo fluxo se refere a uma fonte sequencial de dados que, por exemplo, são lidos ou gravados em um arquivo

# Arquivos binários x Arquivos de texto

Criados com base em **fluxo de bytes**, podem ser lidos por um programa que converte os dados em um formato legível por humanos



Criados com base em **fluxo de caracteres**, podem ser lidos por editores de texto



# Arquivos e Fluxos

- Um programa Java abre um arquivo criando e associando um objeto ao fluxo de bytes ou caracteres, e o construtor do objeto interage com o sistema operacional para abrir esse arquivo
- Java também pode associar fluxos a diferentes dispositivos

# Exemplos de Fluxos

- **System.in**: objeto de fluxo de entrada padrão, permite inserir bytes a partir do teclado
- **System.out**: objeto de fluxo de saída padrão, permite enviar caracteres para a tela
- **System.err**: objeto de fluxo de erro padrão, permite enviar mensagens de erro para a tela

# Classes de Fluxos

- O processamento de arquivos é realizado utilizando o **pacote java.io** que inclui classes de fluxo:
  - **FileInputStream**
  - **FileOutputStream**
  - **FileReader**
  - **FileWriter**

# Classes de Fluxos

- **Arquivos binários**

- **FileInputStream**: para entrada baseada em **bytes**
- **FileOutputStream**: para saída baseada em **bytes**

- **Arquivos de texto**

- **FileReader**: para entrada baseada em **caracteres**
- **FileWriter**: para saída baseada em **caracteres**

# Classes de Fluxos

- Além das classes explicadas anteriormente também temos em **java.util**:
  - **Scanner**: para entrada baseada em caracteres a partir do teclado ou de um arquivo
  - **Formatter**: para saída baseada em caracteres na tela ou em arquivo



# Fluxos de Bytes

- Todos os fluxos de bytes são subclasses das classes abstratas **InputStream** ou **OutputStream**
- São utilizados para manipulação de arquivos binários, como áudio, imagem ou dados em geral

# Fluxos de Bytes - Leitura

- Para ler um byte de um arquivo usamos `FileInputStream`
- Primeiro criamos um fluxo de entrada

```
InputStream fluxo = new FileInputStream ("arquivo.dat");
```

- O argumento deverá ser o `nome do arquivo`, podendo incluir o caminho

# Fluxos de Bytes - Escrita

- Para escrever um byte em um arquivo usamos **FileOutputStream**
- Primeiro criamos um fluxo de saída

**OutputStream** fluxo = new **FileOutputStream** (“arquivo.dat”);

- O argumento deverá ser o **nome do arquivo**

# Serialização de Objetos

- A linguagem Java permite ler e escrever objetos inteiros em arquivos
- Isso é realizado com fluxos de bytes (arquivos binários)
- A classe deve implementar a **interface Serializable**

# Serialização de Objetos

- **Serialização** é um mecanismo para ler ou escrever um objeto inteiro a partir de um arquivo
- Um objeto **serializado** é um objeto representado como uma sequência de bytes que inclui:
  - Dados do objeto;
  - Informações sobre o tipo do objeto;
  - Tipos dos dados armazenados no objeto.
- Um objeto pode ser **desserializado** a partir do arquivo

# Serialização de Objetos

- As classes **ObjectInputStream** e **ObjectOutputStream** permitem que objetos sejam lidos ou gravados em um fluxo
- Para usar a serialização com arquivos inicializamos esses objetos de fluxo com objetos de fluxo que lêem e escrevem bytes em arquivos: **FileInputStream** e **FileOutputStream**

# Serialização de Objetos

- Abertura do fluxo para leitura:

```
ObjectInputStream entrada =  
    new ObjectInputStream(  
        new FileInputStream("meusobjetos.ser"));
```

- Abertura do fluxo para escrita:

```
ObjectOutputStream saida =  
    new ObjectOutputStream(  
        new FileOutputStream("meusobjetos.ser"));
```

# Serialização de Objetos

- Leitura de dados:

**objeto = ( Tipo ) entrada.readObject();**

- Escrita de dados:

**saida.writeObject(objeto);**

- Fechamento dos arquivos:

**entrada.close();**

**saida.close();**



# Exemplo 1

```
Cliente.java
1 package senac.poo.aula10;
2
3 import java.io.Serializable;
4
5 public class Cliente implements Serializable {
6
7     private String cpf;
8     private String nome;
9
10    public String getCpf() {
11        return cpf;
12    }
13
14    public String toString() {
15        return "CPF: " + cpf + " NOME: " + nome;
16    }
17
18    public void setCpf(String cpf) {
19        this.cpf = cpf;
20    }
21    public String getNome() {
22        return nome;
23    }
24    public void setNome(String nome) {
25        this.nome = nome;
26    }
27
28
29    public Cliente(String cpf, String nome) {
30        this.cpf = cpf;
31        this.nome = nome;
32    }
33 }
```

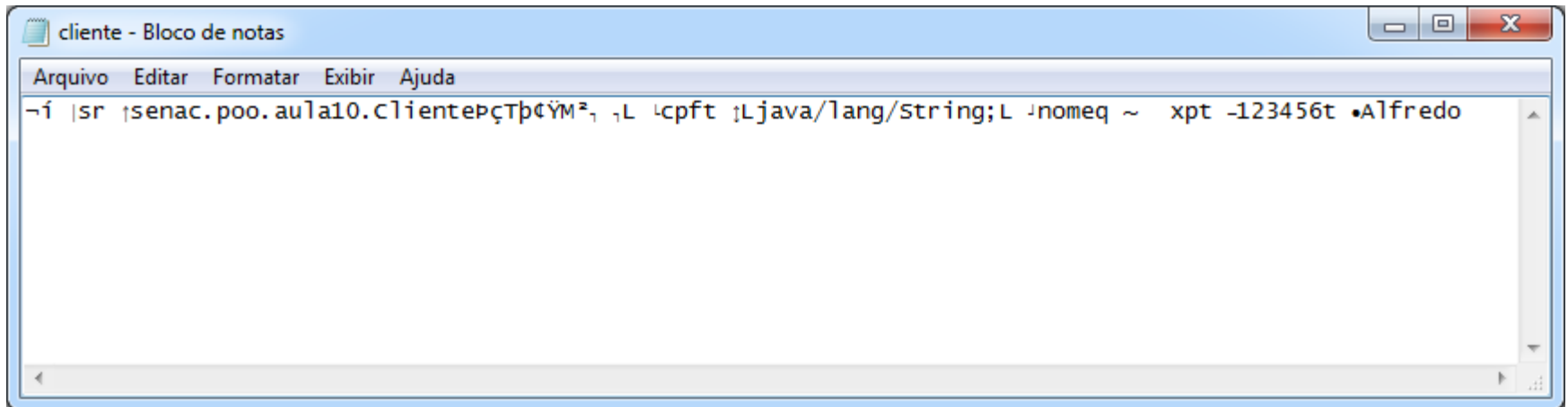
# Antes de continuarmos ...

- A interface **Serializable** é uma interface de marcação, não tem nenhum método
- A classe é **marcada** para permitir que os objetos sejam serializáveis
- Devemos verificar que cada atributo da classe também seja serializável

# Exemplo 1

```
Programa.java ✕
1 package senac.poo.aula10;
2
3 import java.io.*;
4
5 public class Programa {
6
7     public static void main(String[] args) {
8
9         Cliente cliente = new Cliente("123456", "Alfredo");
10        FileOutputStream fluxo;
11
12        try {
13            fluxo = new FileOutputStream("cliente.ser");
14            ObjectOutputStream objarq = new ObjectOutputStream(fluxo);
15            objarq.writeObject(cliente);
16            objarq.close();
17            System.out.println("Arquivo gravado!");
18        } catch (FileNotFoundException e) {
19            e.printStackTrace();
20        } catch (IOException e) {
21            e.printStackTrace();
22        }
23    }
24 }
```

# Exemplo 1

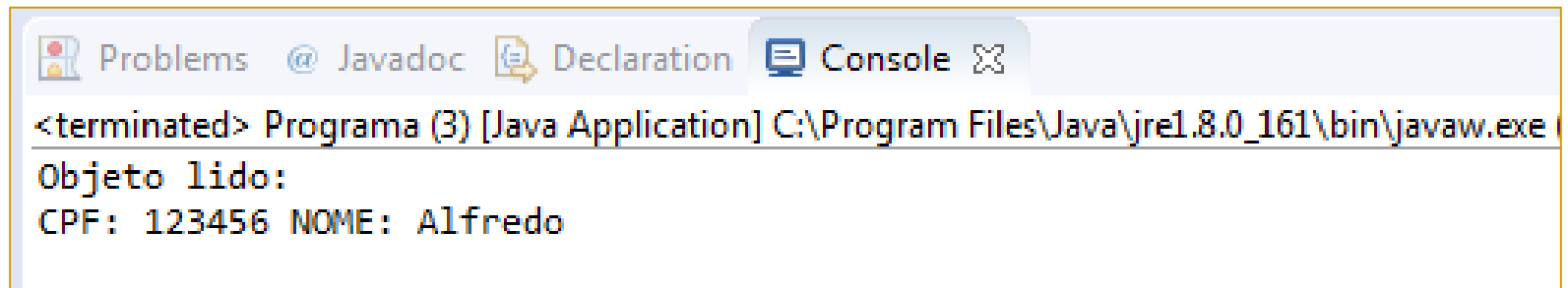


A screenshot of a Windows Notepad window titled "cliente - Bloco de notas". The window has a menu bar with "Arquivo", "Editar", "Formatar", "Exibir", and "Ajuda". The text area contains a single line of Java code: `~f |sr |senac.poo.aula10.ClientePçTbçYM², |L |cpft |Ljava/lang/String;L |nomeq ~ xpt -123456t •Alfredo`. The code is displayed in a monospaced font. The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

# Exemplo 1

```
Programa.java
1 package senac.poo.aula10;
2
3 import java.io.*;
4
5 public class Programa {
6
7     public static void main(String[] args) {
8
9         Cliente cliente;
10        FileInputStream fluxo;
11
12        try {
13            fluxo = new FileInputStream("cliente.ser");
14            ObjectInputStream objarq = new ObjectInputStream(fluxo);
15            cliente = (Cliente) objarq.readObject();
16            objarq.close();
17            System.out.println("Objeto lido: \n"+cliente);
18        } catch (FileNotFoundException e) {
19            e.printStackTrace();
20        } catch (IOException e) {
21            e.printStackTrace();
22        } catch (ClassNotFoundException e) {
23            e.printStackTrace();
24        }
25    }
26 }
```

# Exemplo 1



The screenshot shows a console window from an IDE. The title bar includes tabs for 'Problems', 'Javadoc', 'Declaration', and 'Console'. The console text indicates a terminated Java application and displays the output of a program that reads an object.

```
<terminated> Programa (3) [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe  
Objeto lido:  
CPF: 123456 NOME: Alfredo
```

# Serialização de Objetos

- Podemos ler mais de um objeto a partir de um fluxo de objetos
- Se existir uma tentativa de leitura depois do final do arquivo, `readObject()` lança uma `EOFException`
- Se a classe do objeto sendo lido não puder ser localizada, `readObject()` lança uma exceção `ClassNotFoundException`

# Exemplo 2

```
Programa.java
1 package senac.poo.aula10;
2
3 import java.io.*;
4
5 public class Programa {
6
7     public static void main(String[] args) {
8
9         Cliente cliente;
10        FileInputStream fluxo;
11        ObjectInputStream objarq = null;
12
13        try {
14            fluxo = new FileInputStream("cliente.ser");
15            objarq = new ObjectInputStream(fluxo);
16            while (true) {
17                cliente = (Cliente) objarq.readObject();
18                System.out.println(cliente);
19            }
20        } catch (EOFException ex) {
21            System.out.println("Chegou no final do arquivo!");
22        } catch (FileNotFoundException e) {
23            e.printStackTrace();
24        } catch (IOException e) {
25            e.printStackTrace();
26        } catch (ClassNotFoundException e) {
27            e.printStackTrace();
28        }
29    }
30 }
```



# Serialização de Objetos

- Estruturas de dados do tipo **Collection** podem ser lidas ou escritas na sua totalidade sem necessidade de iteração

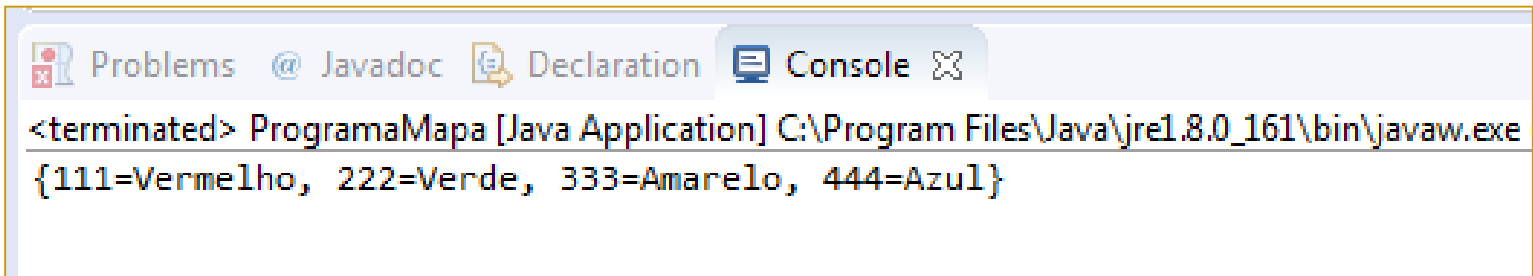
# Exemplo 3

```
ProgramaMapa.java  ✖
1  package senac.poo.aula10;
2
3  import java.util.*;
4  import java.io.*;
5
6  public class ProgramaMapa {
7
8      public static void main(String[] args) {
9
10         Map<Integer, String> mapa = new TreeMap<Integer, String>();
11
12         mapa.put(111, "Vermelho");
13         mapa.put(222, "Verde");
14         mapa.put(333, "Amarelo");
15         mapa.put(444, "Azul");
16     }
```

# Exemplo 3

```
16
17     try {
18         FileOutputStream fluxoOut = new FileOutputStream("meuArquivo.ser");
19         ObjectOutputStream fOut = new ObjectOutputStream(fluxoOut);
20         fOut.writeObject(mapa);
21
22         FileInputStream fluxoIn = new FileInputStream("meuArquivo.ser");
23         ObjectInputStream fIn = new ObjectInputStream(fluxoIn);
24         TreeMap<Integer, String> mapaNovo = (TreeMap) fIn.readObject();
25
26         fIn.close();
27         fOut.close();
28         System.out.println(mapaNovo);
29     } catch (FileNotFoundException e) {
30         e.printStackTrace();
31     } catch (IOException e) {
32         e.printStackTrace();
33     } catch (ClassNotFoundException e) {
34         e.printStackTrace();
35     }
36 }
37 }
```

# Exemplo 3



The screenshot shows a console window with a tab bar at the top containing 'Problems', 'Javadoc', 'Declaration', and 'Console'. The 'Console' tab is active. The output text in the console is: `<terminated> ProgramaMapa [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe {111=Vermelho, 222=Verde, 333=Amarelo, 444=Azul}`