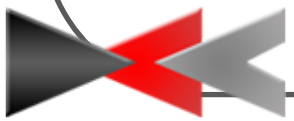
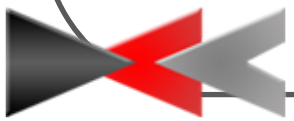
Entrada e Saída



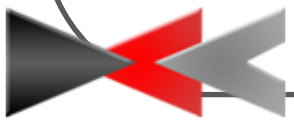
Motivação

- Comunicação das aplicações
  - Arquivos
  - Conexões de redes
  - Memória
- Diferentes tipos de informações
  - Bytes/caracteres, bits, estruturas de dados
- Unificação dos mecanismos de E/S
  - Streams de entrada e saída



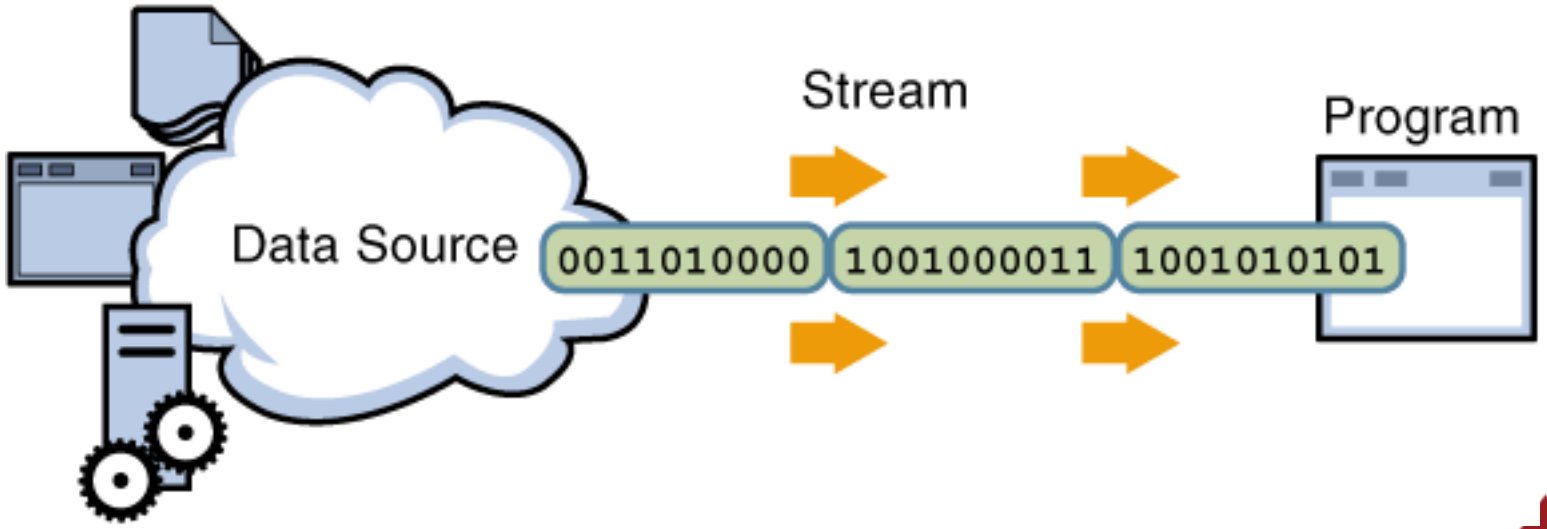
Motivação

- Poucas aplicações são funcionais apenas com dados transientes
- A grande maioria precisa armazenar informações em mídia de longa duração para recuperá-la tempos depois
  - Sistema de Arquivos x Banco de Dados



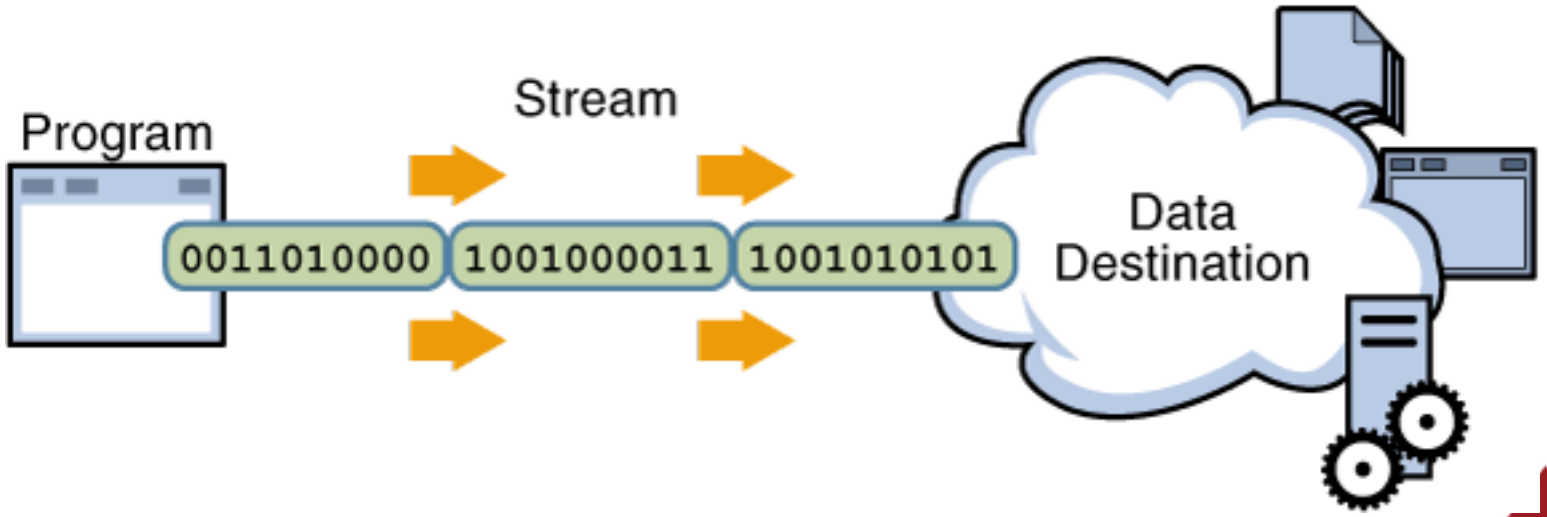
Stream de Entrada

- Para obter informações, uma aplicação abre um stream de uma fonte (arquivo, socket, memória, etc) e lê sequencialmente



Stream de Saída

- Para enviar informações, uma aplicação abre um stream para um destino (arquivo, socket, memória etc.) e escreve sequencialmente



Leitura e Escrita de Streams

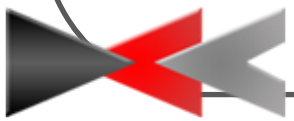
- Independente da fonte/destino e do tipo de informação, os algoritmos para leitura e escrita são basicamente os mesmos

Leitura

1. Abre um stream
2. Enquanto há informação:
   a. Lê informação
3. Fecha o stream

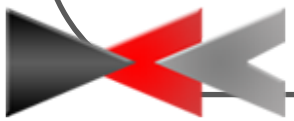
Escrita

1. Abre um stream
2. Enquanto há informação:
   a. Escreve informação
3. Fecha o stream



Pacote java.io

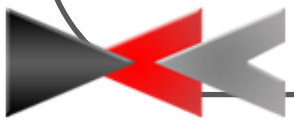
- Coleção de classes que suportam algoritmos de entrada e saída
- As classes são divididas em duas hierarquias, baseadas no tipo de dados (bytes ou caracteres) sobre as quais elas atuam
  - InputStream/OutputStream
  - Reader/Writer



java.io

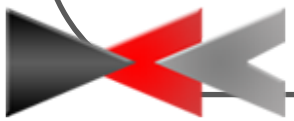
- São mais de 40 classes, divididas em:
  - Fluxos de entrada (input streams);
  - Fluxos de saída (output streams);
  - Leitores (readers);
  - Escritores (writers);
  - Arquivo de acesso aleatório (random access file).

- Classes podem indicar a mídia de I/O ou a forma de manipulação dos dados;



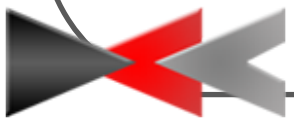
Streams de Bytes

- As classes InputStream e OutputStream são superclasses abstratas de todos os streams de bytes
  - InputStream define um método abstrado read para ler um byte de uma stream
  - OutputStream define um método abstrato write para escrever um byte em uma stream
- Suas subclasses provêem E/S especializada para cada tipo de fonte/destino



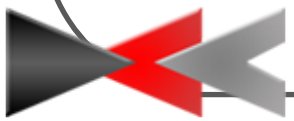
Streams de Caracteres

- As classes Reader e Writer são as superclasses abstratas de todos os streams de caracteres
  - Reader define um método abstrato real para ler uma sequência de caracteres de uma stream
  - Writer define um método abstrato write para escrever uma sequência de caracteres em uma stream
- Subclasses provêem E/S especializada para diferentes tipos de fonte/destino



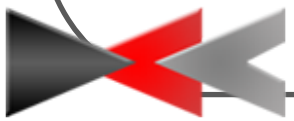
IOException

- É uma extensão da classe Exception
- Sinaliza a ocorrência de uma falha ou interrupção em uma operação de E/S
- Algumas subclasses:
  - EOFException, FileNotFoundException, InterruptedIOException, MalformedURLException, SocketException



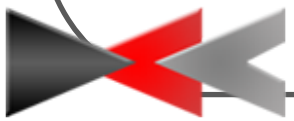
Buffered Streams

- Por default, os streams não são bufferizados
  - Essa funcionalidade pode ser obtida adicionando-se uma "camada" sobre o stream
- BufferedInputStream, BufferedOutputStream
  - Ex. public BufferedInputStream(InputStream in, int size)
- BufferedReader, BufferedWriter
  - Ex. public BufferedReader(Reader in, int size)



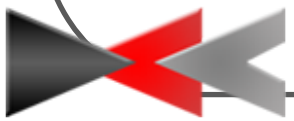
Entrada/Saída em Arquivos

- Acesso via streams
  - FileInputStream
  - FileOutputStream
  - FileReader
  - FileWriter
- Acesso aleatório
  - RandomAcessFile



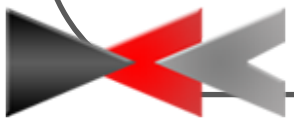
Classe FileInputStream

- Especialização de InputStream para leitura de arquivos
  - public FileInputStream(String name)
  - public FileInputStream(File file)
- Usando stream bufferizada
  - BufferedInputStream in = new BufferedInputStream
    (new FileInputStream("arquivo.dat");

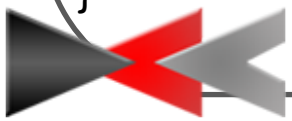


Classe FileOutputStream

- Especialização de OutputStream para leitura de arquivos
  - public FileOutputStream(String name)
  - public FileOutputStream(String name, boolean append)
  - public FileOutputStream(File file)

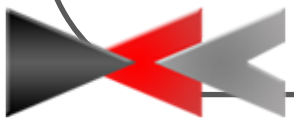


```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class CopyBytes {
    public static void main(String[] args) throws IOException {
        FileInputStream in = null;
        FileOutputStream out = null;
        try {
            in = new FileInputStream(”entrada.txt");
            out = new FileOutputStream(”saida.txt");
            int c;
            while ((c = in.read()) != -1) {
                out.write(c);
            }
        } finally {
            if (in != null) { in.close(); }
            if (out != null) { out.close(); }
        }
    }
}
```



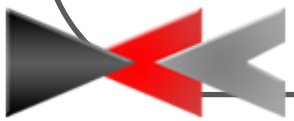
Classe FileReader

- É uma subclasse de InputStreamReader
  - public FileReader(String name)
  - public FileReader(File file)
- Usando stream bufferizada
  - BufferedReader in = new BufferedInputStream (new FileReader("arquivo.dat");

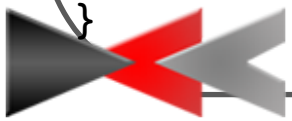


Classe FileWriter

- É uma subclasse de OutputStreamReader
  - public FileWriter(String name)
  - public FileWriter(String name, boolean append)
  - public FileWriter(File file)

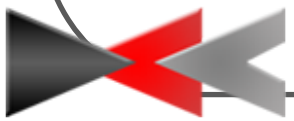


```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class CopyCharacters {
    public static void main(String[] args) throws IOException {
        FileReader inputStream = null;
        FileWriter outputStream = null;

        try {
          inputStream = new FileReader(”entrada.txt");
          outputStream = new FileWriter("characteroutput.txt");
          int c;
          while ((c = inputStream.read()) != -1) {
             outputStream.write(c);
          }
        } finally {
          if (inputStream != null) { inputStream.close(); }
          if (outputStream != null) { outputStream.close(); }
        }
    }
}
```



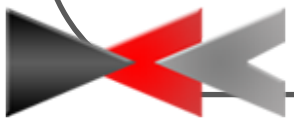
Classe File

- Representa um arquivo (ou diretório) no sistema de arquivos nativo
- Permite obter informações sobre arquivos e diretórios
- Permite também executar operações como criar, renomear e apagar arquivos e diretórios



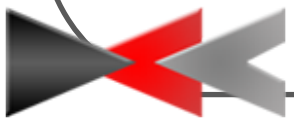
Classe File

```
import java.io.File;

...

File arquivo = new File("texto.txt");

if( ! arquivo.exists() ){
    System.out.println("até aqui não foi criado um arquivo");
}

try{
    if( arquivo.createNewFile() ){
        System.out.println("O arquivo foi criado");
    }else{
        System.out.println("O arquivo não foi criado, talvez ele já exista");
    }
}catch(IOException ex){
    ex.printStackTrace();
}
```



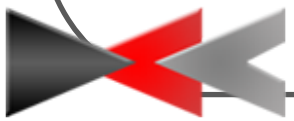
Classe File

```
File dir = new File( "dir");

if( dir.mkdir() ){
    System.out.println("Diretório criado");
}else{
    System.out.println("Diretório não criado");
}
try{
    File file = new File(dir, "file.txt");
    if( file.createNewFile() ){
        System.out.println("Arquivo criado");
    }else{
        System.out.println("Arquivo não criado");
    }
}catch(IOException ex){
    ex.printStackTrace();
}
```



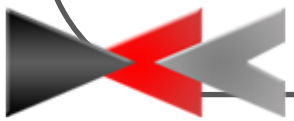
Classe Path

- Apresentada no JDK7
  - Pacote java.nio.file
- Serve para representar um caminho no sistema de arquivos
  - Representação dependente do sistema
  - /home/arquivo – Linux
  - C:\home\arquivo - Windows



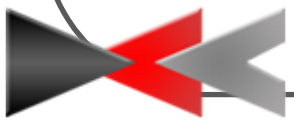
Path

```
Path arquivo = Paths.get(“/tmp/teste.txt”);;
try {
    arquivo.createFile();   //Cria o arquivo vazio com as
permissões padrão, etc.
} catch (FileAlreadyExists x) {
    System.err.format("O arquivo nomeado %s já existe
%n", arquivo);
} catch (IOException x) {
    //Coloque aqui outro tipo de ocorrência, tal como
permissões.
    System.err.format("createFile error: %s%n", x);
}
```



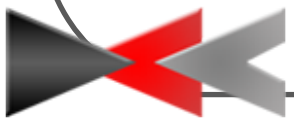
Path

```
Path file = Paths.get(“/tmp/teste.txt”);
InputStream in = null;
try {
    in = file.newInputStream();
    BufferedReader reader = new BufferedReader(new
InputStreamReader(in));
    String line = null;
    while ((line = reader.readLine()) != null) {
        System.out.println(line);
    }
} catch (IOException x) {
    System.err.println(x);
} finally {
    if (in != null) in.close();
}
```



Streams de Dados

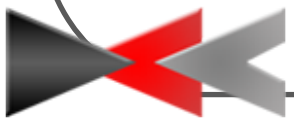
- Definidos por interfaces
  - DataInput
  - DataOutput
- Permitem escrita e leitura de tipos básicos
- Essas interfaces são implementadas por
  - DataInputStream
  - DataOutputStream
  - RandomAccessFile



Exemplo de Stream de Dados

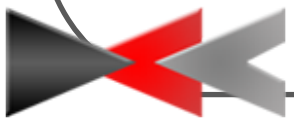
- Imagine uma estrutura de dados sobre itens em um estoque, contento:
  - Uma descrição
  - Uma quantidade, e
  - Um valor unitário

double	Item price	DataOutputStream.writeDouble	DataInputStream.readDouble	19.99
int	Unit count	DataOutputStream.writeInt	DataInputStream.readInt	12
String	Item description	DataOutputStream.writeUTF	DataInputStream.readUTF	"Java T-Shirt"



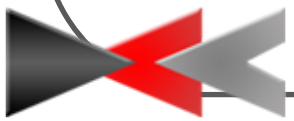
Descrição dos Itens

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.IOException;
import java.io.EOFException;

public class DataStreams {
    static final String dataFile = "fatura";

    static final double[] precos = { 19.99, 9.99, 15.99,
3.99, 4.99 };
    static final int[] units = { 12, 8, 13, 29, 50 };
    static final String[] descs = { "Java T-shirt”,  "Java
Mug”, "Duke Juggling Dolls”, "Java Pin”,  "Java Key
Chain" };
```



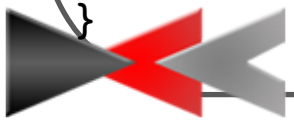
Escrevendo

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

  DataOutputStream out = null;

  try {
     out = new DataOutputStream(new
      BufferedOutputStream(new FileOutputStream(dataFile)));

     for (int i = 0; i < precos.length; i ++) {
        out.writeDouble(precos[i]);
        out.writeInt(units[i]);
        out.writeUTF(descs[i]);
     }
  } finally {
     out.close();
  }
```



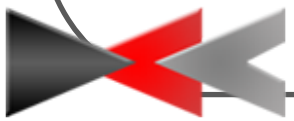
Lendo

```
DataInputStream in = null;
double total = 0.0;
  try {
     in = new DataInputStream(new
         BufferedInputStream(new FileInputStream(dataFile)));

     double preco; int unit; String desc;

     try {
       while (true) {
         price = in.readDouble(); unit = in.readInt(); desc =
in.readUTF();
         System.out.format("Você adquiriu %d unidades de %s a
$%.2f%n”, unit, desc, preco);
         total += unit * preco;
       }
       } catch (EOFException e) { }
        System.out.format("Para um TOTAL de: $%.2f%n", total);
       }
       finally { in.close();}
    }
}
```



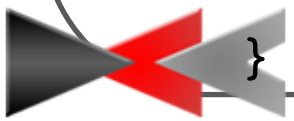
Classe RandomAccessFile

- Permite a leitura e escrita em um arquivo de acesso randômico
- Implementa as interfaces DataInput e DataOutput
- Possui um *file pointer* que indica a posição (índice) corrente
  - Esse file pointer pode ser obtido através do método *getFilePointer* e alterado através do método seek



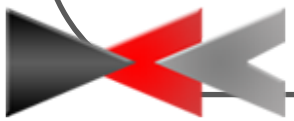
Exemplo

```
import java.io.*;
public class TesteRandom {
   public static void main(String argv[]) {
       try {
          TesteRandom r = new TesteRandom();
          RandomAccessFile raf = new
   RandomAccessFile (“teste.txt”, “rw”);
          r.escreve(raf);
          r.leUm(raf, 2); // Lê b
          r.escreveUm(raf, 2, ‘x’);
          r.leUm(raf, 2); // Lê x
       } catch (IOException ioe) {
          System.out.println(ioe);
       }
}
```



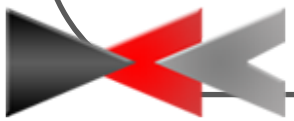
Exemplo

```
 public void escreve(RandomAccessFile raf)
throws IOException {
    char[] letras = {'a', 'b', 'c', 'd'};
    for(int i=0; i<4; i++) {
       raf.writeChar(letras[i]);
    }
 }

public void leUm(RandomAccessFile raf, int
pos) throws IOException {
    raf.seek(pos);
    System.out.println(raf.readChar());
}
```



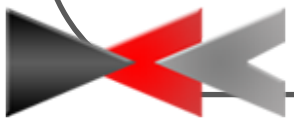
Exemplo

```
    public void escreveUm (RandomAccessFile raf,
   int pos, char c) throws IOException {
       raf.seek(pos);
       raf.writeChar(c);
    }
}
```



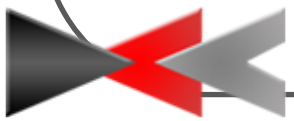
Streams de Objetos

- Definidos pelas interfaces ObjectInput e ObjectOutput
  - Implementadas por ObjectInputStream e ObjectOutputStream
- ObjectInput estende DataInput para incluir objetos, arrays e Strings
- ObjectOutput estende DataOutput para incluir objetos, arrays e Strings



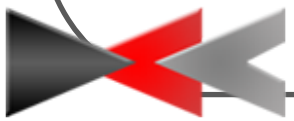
Utilização de streams de objetos

- Um ObjectInputStream “deserializa” dados e objetos anteriormente escritos através de um ObjectOutputStream
- Cenários:
  - Persistência de objetos, quando esses streams são usados em conjunto com FileInputStream e FileOutputStream
  - Transferência de objetos entre hosts
  - ...



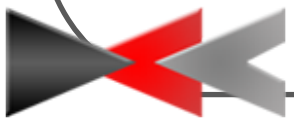
Streams de Objetos

- ObjectInputStream
  - public final Object readObject()
- ObjectOutputStream
  - public final void writeObject(Object obj)



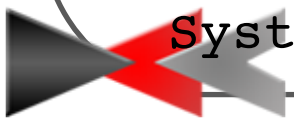
Exemplo

```
FileOutputStream ostream = new
                        FileOutputStream("t.tmp");

ObjectOutputStream out = new
  ObjectOutputStream(ostream);

out.writeInt(12345);
out.writeObject("Hoje");
out.writeObject(new Date());
out.flush();
ostream.close();
```



Recupendando

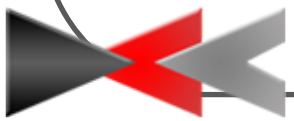
```
FileInputStream istream = new
   FileInputStream(“t.tmp”);

ObjectInputStream in = new
   ObjectInputStream(istream);

int i = in.readInt();
String hoje = (String)in.readObject();
Date date = (Date)in.readObject();

istream.close();
System.out.println(i);
System.out.println(hoje);
System.out.println(date);
```



Interface Serializable

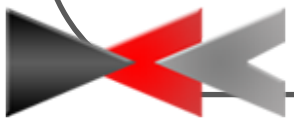
- Somente objetos cujas classes implementem a interface Serializable podem ser serializados

- Essa interface não tem métodos, mas uma classe “serializable” pode definir métodos readObject e writeObject para fazer validações no estado do objeto



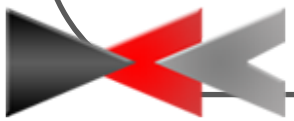
Exemplo

```
class Funcionario implements Serializable {

   private void readObject(ObjectInputStream is)
   throws ClassNotFoundException, IOException {
       is.defaultReadObject();
       if (!isValid())
        throw new IOException("Objecto inválido");
   }

   private boolean isValid() {
   }
}
```



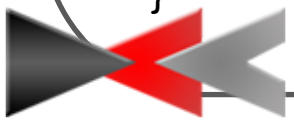
Serializando um objeto

```
public class Funcionario implements Serializable
   {
    public String nome;
    public String endereco;
    public int CPF;
}
```



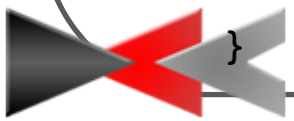
Serialize Demo

```
public static void main(String[] args) {
        // TODO code application logic here
    Funcionario f = new Funcionario();
    f.nome = "Romualdo";
    f.endereco = "romualdomrc@gmail.com";
    f.CPF = 11122333;
    try {
      FileOutputStream fileOut = new FileOutputStream("/tmp/
  funcionario.ser");
      ObjectOutputStream out = new
  ObjectOutputStream(fileOut);
      out.writeObject(f);
      out.close();
      fileOut.close();
      System.out.printf("Funcionario salvo");
    } catch(IOException i) { i.printStackTrace();}
}
```



Deserialize Demo

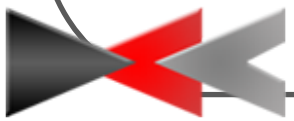
```
public static void main(String[] args) throws
  ClassNotFoundException {
        Funcionario f = null;
        try {
            FileInputStream fileIn = new FileInputStream("/
  tmp/funcionario.ser");
            ObjectInputStream in = new
  ObjectInputStream(fileIn);
            f = (Funcionario) in.readObject();
            in.close();
            fileIn.close();
            }catch(IOException i) {
               i.printStackTrace(); return;
            }
         System.out.println("Nome: " + f.nome);
         System.out.println("Endereço: " + f.endereco);
         System.out.println("CPF: " + f.CPF);
}
```



Uniform Resource Locator

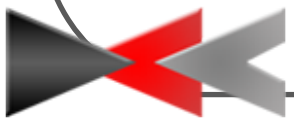
- A classe URL modela URLs, permitindo a obtenção de informações e conteúdo de páginas na Web

- Parte do pacote java.net



Exemplo

```
import java.io.*;
import java.net.*;

public class PegaPagina {
   public static void main(String[] args) throws
   Exception {
       if (args.length ==0) {
          System.err.println("Forneça o endereço");
          return;
       }
       URL url = new URL(args[0]);//http://www.ufjf.br
       InputStream is = url.openStream();
       Reader r = new InputStreamReader(is);
```



Exemplo

```
        BufferedReader br = new BufferedReader(r);
        String linha;
        while ((linha = br.readLine()) != null) {
            System.out.println(linha);
        }
    }
}
```