MC 302

Entrada e Saída em Java

Prof. Fernando Vanini

IC-Unicamp

O pacote java.io

- Define as classes e métodos voltados às operações de entrada e saída em Java, de forma independente de plataforma.
- As operações de entrada e saída em Java se baseiam no conceito de *stream*.

Stream

- É uma sequência de dados (bytes) usada como origem ou destino dos dados consumidos ou gerados por uma programa.
- Fisicamente pode ser
  - um arquivo
  - área de memória compartilhada (*pipe*)
  - uma conexão de rede
  - entrada padrão ou saída padrão (*console*)

OutputStream e *InputStream*

- *OutputStream*: usado por um programa Java para a escrita de dados.
- *InputStream*: usado para leitura de dados por um programa Java.

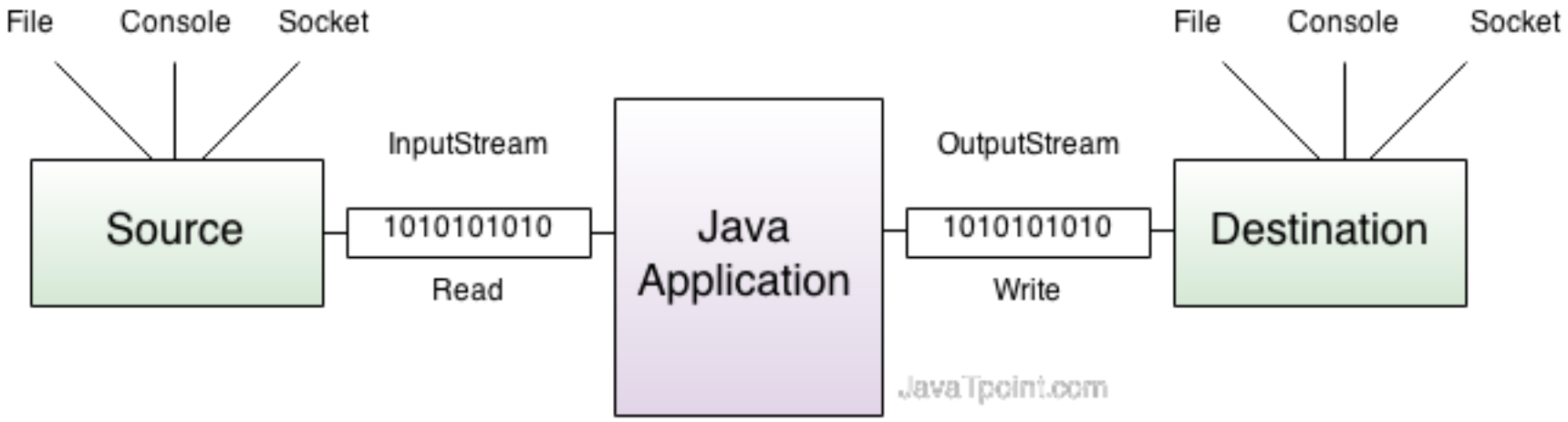


Imagem extraída de: http://www.javatpoint.com/java-io

- Principais métodos

```
public void write(int) throws IOException;
public void write(byte[]) throws IOException;
public void flush() throws IOException;
public void close() throws IOException;
```

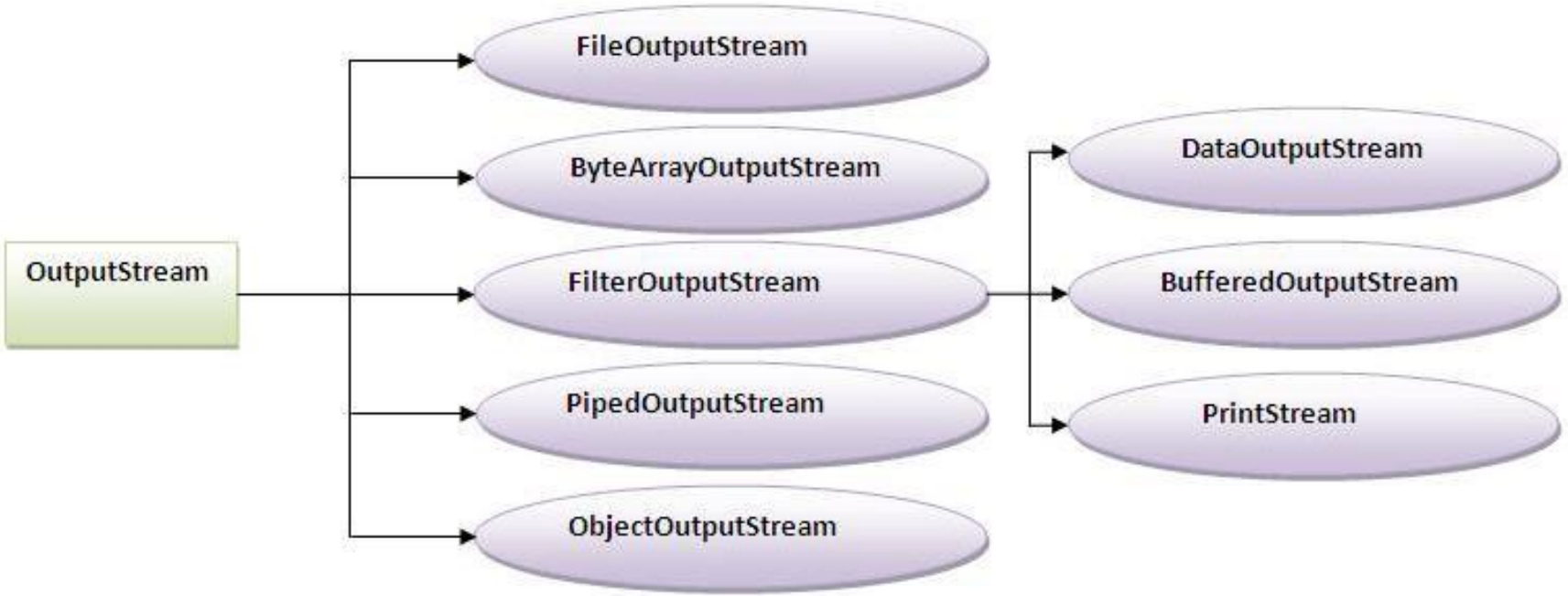


Imagem extraída de: http://www.javatpoint.com/java-io

- Principais métodos

```
public abstract int read() throws IOException;
public int available() throws IOException;
public void close() throws IOException;
```

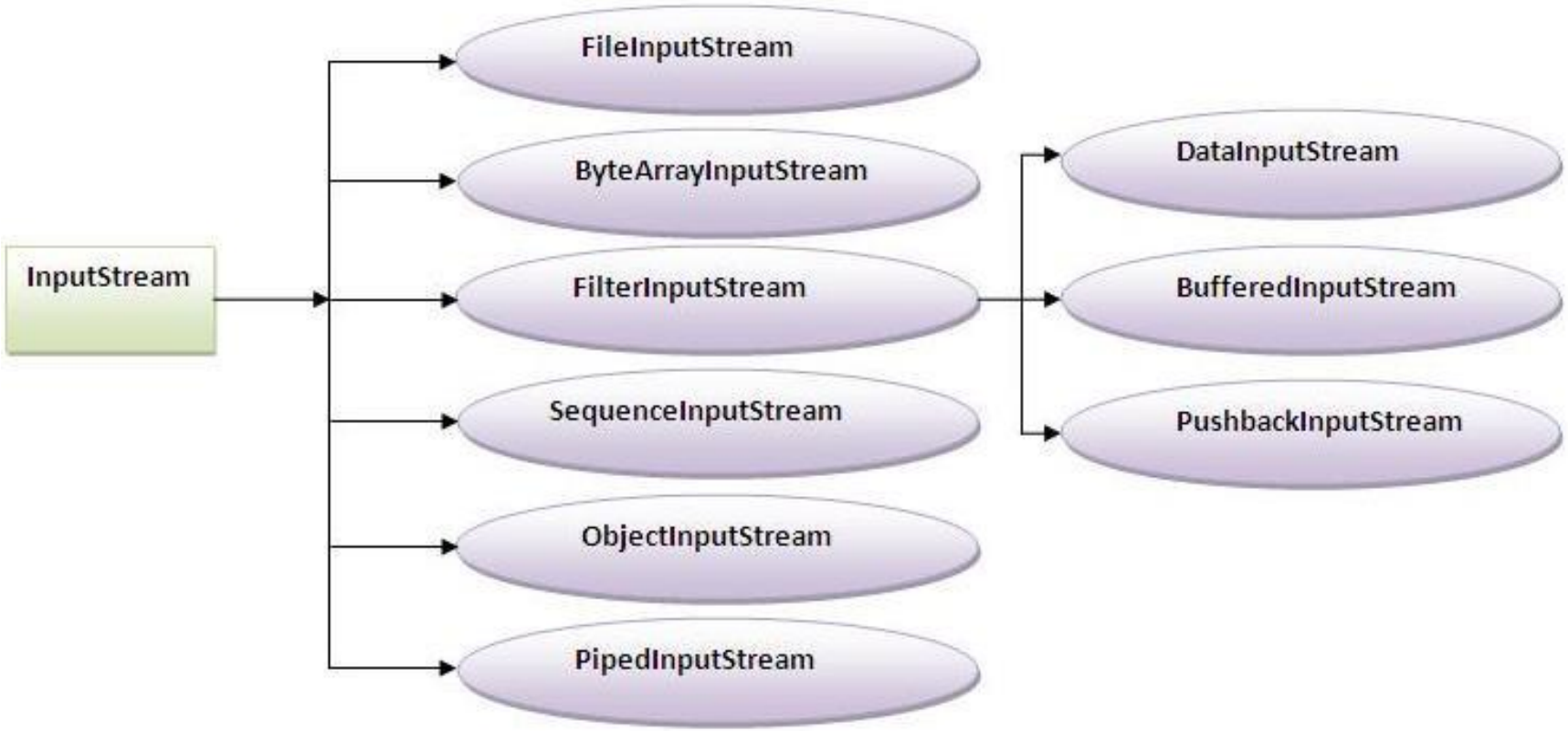


Imagem extraída de: http://www.javatpoint.com/java-io

- Voltado à escrita de bytes num arquivo.

```
import java.io.*;

public class ExemploOutputStream{

        static String txt = "era uma vez um gato xadres";
        static byte[] b = txt.getBytes(); // what?

        public static void main(String[] args){
          try{
                FileOutputStream out =
                        new FileOutputStream("stream1.txt");
                out.write(b);
                out.close();
                System.out.println("fim");
          } catch(IOException ex){
                ex.printStackTrace();
          }
        }
}
```

- Voltado à leitura de bytes num arquivo.

```
import java.io.*;

public class ExemploInputStream {

   public static void main(String[] args){
       try{
               FileInputStream input =
                       new FileInputStream("stream1.txt");
               int i = 0;
               while((i = input.read()) != -1){
                       char c = (char)i;
                       System.out.print(c);
               }
               System.out.println();
               input.close();
        } catch(IOException ex){
               ex.printStackTrace();
        }
   }
}
```

- Voltado à ‘escrita’ de bytes num vetor(de bytes) em memória.

```
import java.io.*;

public class ExemploByteArrayOutputStream {

  static String txt = "Era uma vez um gato xadrez";
  static byte[] bytes = txt.getBytes();

  public static void main(String[] args) throws IOException{
    FileOutputStream out = new FileOutputStream("stream2.txt");
    ByteArrayOutputStream ba = new ByteArrayOutputStream();

    ba.write(bytes);
    ba.writeTo(out);  // escreve no arquivo o ‘conteúdo’ de ba

    out.close();
    ba.close(); // na prática, sem efeito.
  }
}
```

- Voltado à escrita texto num arquivo (texto).

```
import java.io.*;

public class ExemploFileWriter {

  static String txt = "Era uma vez um gato xadrez";

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

    FileWriter writer = new FileWriter("fwriter1.txt");

    writer.write(txt);

    writer.close();

    System.out.println("fim");

  }

}
```

FileReader

- Voltado à leitura de dados mantidos num arquivo texto.

```
import java.io.*;

public class ExemploFileReader {

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

     FileReader reader = new FileReader("fwriter1.txt");
     int i = 0;
     while((i = reader.read())!= -1){
         char c = (char)i;
         System.out.print(c);
     }
     reader.close();
  }
}
```

Lendo dados do teclado

- InputStreamReader: usado para leitura de dados via teclado
- BufferedReader: permite a leitura linha a linha via readLine()

```
import java.io.*;

public class ExemploLeituraViaTeclado {

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

    InputStreamReader in = new InputStreamReader(System.in);
    BufferedReader br = new BufferedReader(in);

    System.out.println("Nome:");
    String nome = br.readLine();
    System.out.println(
                        "Parabéns " +
                        nome +
                        ", você acertou seu nome“
                      );
  }

}
```

- PrintStream: oferece métodos para escrita de texto em outro ‘stream’.

```
import java.io.*;

public class ExemploPrintStream {

  static String str = "Atirei um pau no gato (da vizinha).";

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

    PrintStream ps = new PrintStream(
                              new FileOutputStream("fOutStr1.txt")
                     );
    ps.print("texto:");
    ps.println(str);
    ps.close();
    System.out.println("fim");
  }

}
```

RandomAccessFile

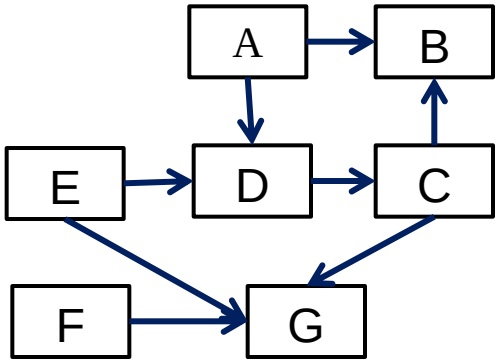
- Os arquivos descritos até aqui são voltados ao ‘acesso sequencial’.
- O acesso aleatório permite que se posicione o ‘ponto de acesso’ antes de cada operação de leitura e escrita.
- Operações
  - Obter o ponto de acesso
  - Alterar o ponto de acesso
  - Leitura ou escrita

```
...
RandomAccessFile rf =  new RandomAccessFile("f1.xyz", "rw");
...
long p = rf.getFilePointer();
rf.writeDouble(48.123);
...
rf.seek(p);
double dd = r.readDouble();
...
rf.close();
```

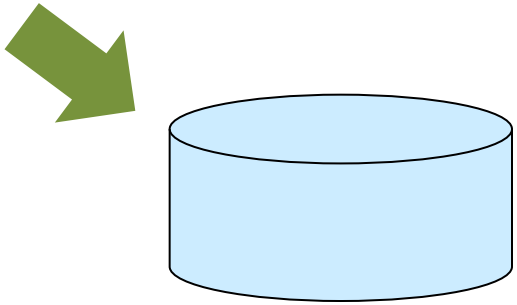
- Permite associar um 'stream' a uma URL (*Universal Resource Locator*)

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

public class ExemploURL {

  public static void main(String[] args) throws IOException {
    URL url = new URL("http://www.ic.unicamp.br/~vanini/mc302");
    InputStream is = url.openConnection().getInputStream();

    BufferedReader reader =
                    new BufferedReader(new InputStreamReader( is ));

    String line = null;
    while( ( line = reader.readLine() ) != null )  {
       System.out.println(line);
    }
    reader.close();
  }
}
```

Problema Geral

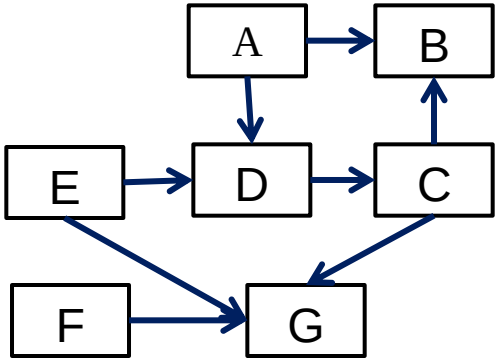


Estrutura em memória



Meio Persistente

Estrutura em memória



- Serialização: consiste em representar uma estrutura de dados como uma sequência de bytes.
- De-serialização: consiste em reconstruir uma estrutura de dados a partir de uma sequência de bytes.

- o pacote java.io oferece diversas classes de apoio à serialização de objetos.
- A interface **`java.io.Serializable`** deve ser implementada pelas classes a serem serializadas.
- As classes
  - **`ObjectOutputStream`**: permite a serialização de uma estrutura de objetos num dispositivo de saída.
  - **`ObjectInputStream`**: permite a de-serialização de objetos a partir de dados lidos de um dispositivo de entrada.

Um exemplo: serialização

```
/**
 * Grava em arquivo a representação serializada de um mapa
 * @param map mapa a ser serializado (Map implementa Serializable)
 * @param fName nome do arquivo
 * @throws IOException caso haja erro no acesso ao arquivo.
 */

static void persiste(Map<String,String> map, String fName)
   throws IOException{
        FileOutputStream f_out = new FileOutputStream(fName);

        ObjectOutputStream obj_out = new ObjectOutputStream (f_out);
        obj_out.writeObject (map);

        obj_out.close();
}
```

Um exemplo: de-serialização

```
/**
 * Reconstrói um mapa a partir da sua representação
 * serializada em arquivo.
 * @param fName nome do arquivo
 * @return mapa reconstruído
 * @throws IOException caso haja erro no acesso ao arquivo
 * @throws ClassNotFoundException caso os dados serializados
 *                                não representem um mapa.
 */

static Map<String,String> carrega(String fName)
        throws IOException, ClassNotFoundException {
        FileInputStream f_in = new FileInputStream(fName);

        ObjectInputStream obj_in = new ObjectInputStream (f_in);
        Object obj = obj_in.readObject();

        return (Map<String,String>)obj;
}
```

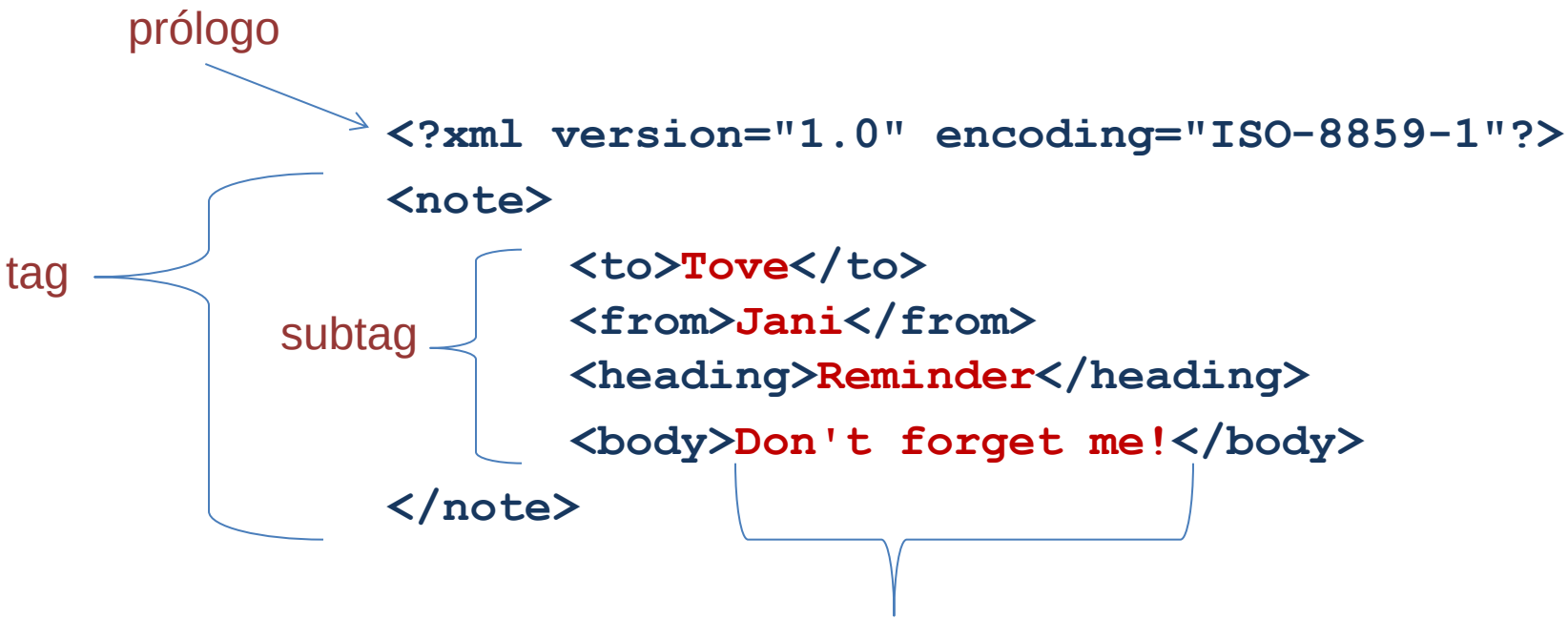
A interface Serializable

- Não define nenhum método específico.
- Os métodos de ObjectInputStream e ObjectOutputStream tratam apenas objetos 'Serializable'.
- No exemplo abaixo, o modificador 'transient' indica que o atributo não deve ser persistido.

```
public class Aluno implements java.io.Serializable {
        String nome;
        int idade;
        String ra;
        transient float nota;

        public Aluno(...) { ... }
        public String toString() { ... }

}
```

- XML (Extensible Markup Language)
  - Notação baseada em texto usada para representar objetos.
  - Padrão definido pelo w3c (World Wide Web Consortium) e amplamente adotado em aplicações via web para a transferência de objetos.

- Estrutura geral de um documento:



XML

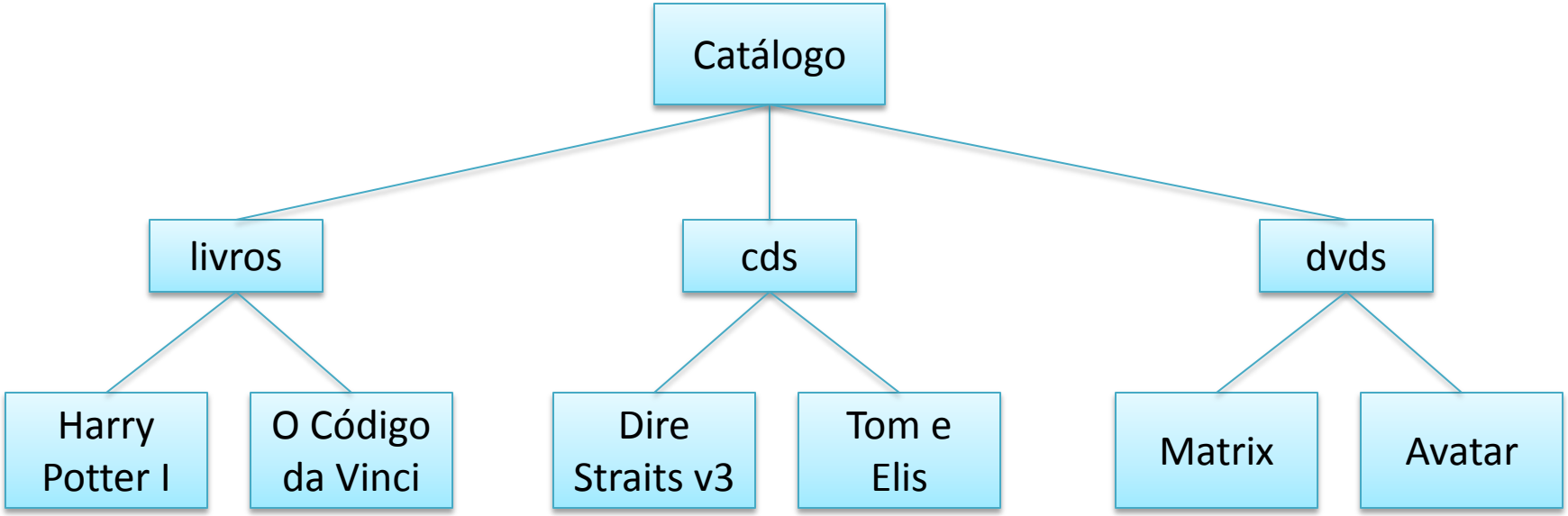
- Não existem tags pré-definidas. São definidas pela aplicação de acordo com a necessidade.
- Uma tag pode ser vazia: sem conteúdo

```
<colour/>
```

- Uma tag, vazia ou não, pode ter atributos. Exemplo:

```
<colour value="AAB1C2" transp = "0"/>
```

- Representação hierárquica dos dados:

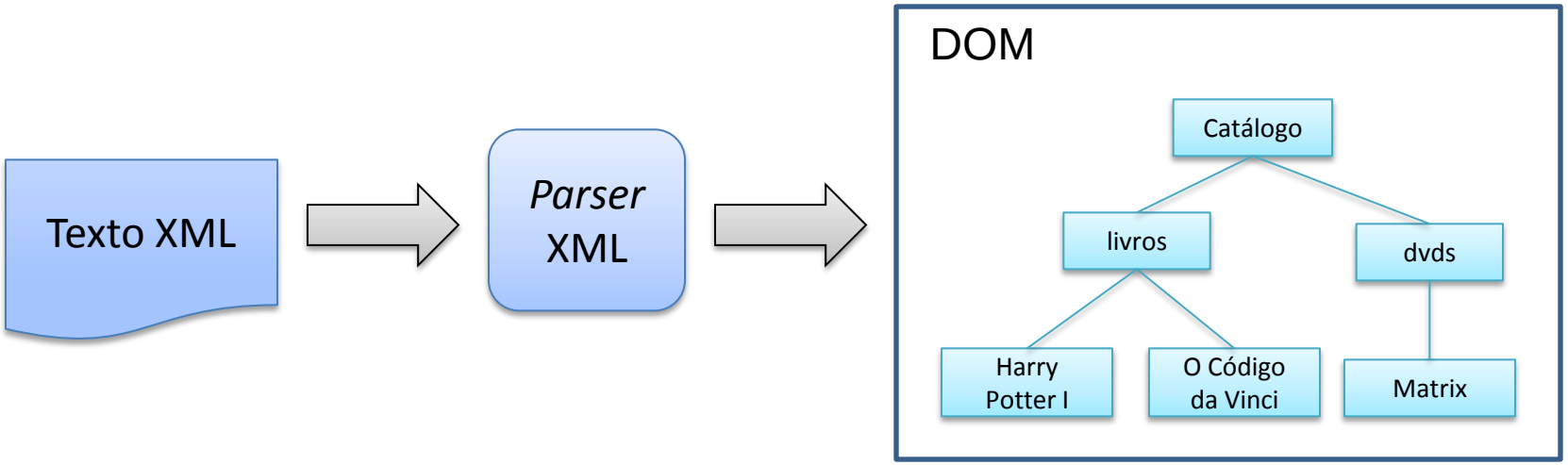


XML : um exemplo

```
<catalogo>
   <livros>
        <livro titulo = "Harry Potter I" />
        <livro titulo = "O Código da Vinci" />
   </livros>
   <cds>
        <cd nome = "Dire Straits v3" />
        <cd nome = "Tom e Elis" />
   </cds>
   </dvds>
        <dvd titulo = "Matrix" />
        <dvd titulo = "Avatar" />
   <dvds>
catalogo>
```

As ferramentas disponíveis para operação com XML, chamadas de *parsers*, na sua grande maioria operam sob dois modos:

- DOM (*Document Object Model*): o texto XML é traduzido para uma estrutura de dados que reflete a estrutura do texto.
- SAX (*Simple Api for XML*): à medida em que o texto XML é lido pelo *parser*, este gera eventos informando a respeito das *tags* e atributos encontrados. A aplicação é responsável pelo tratamento adequado dos evento.



- DOM: a estrutura criada pelo *parser* para representar os dados no texto XML é mantida em memória. A aplicação deve ‘navegar’ nessa estrutura para obter a informação que precisar.

XML Parser: DOM

- A operação de ‘parsing’ de um arquivo XML executada por um parser em modo DOM, gera uma árvore de objetos cuja raiz é um ‘documento’ a partir do qual é possível acessar seus elementos.
- Exemplo usando javax.xml.* :

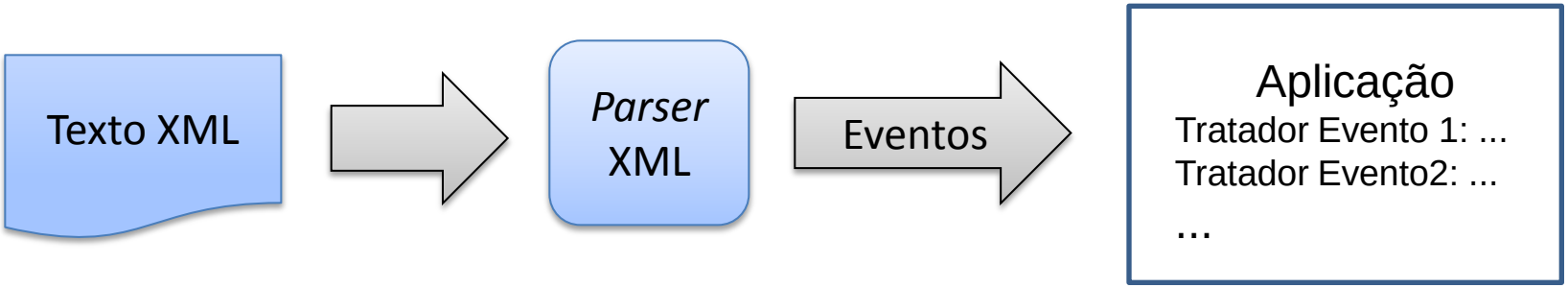
```
/**
 * Le o 'documento' contido num arquivo XML.
 * @param fName nome do arquivo XML
 * @return objeto 'org.w3c.dom.Document' que representa o documento
 * no padrão 'w3c.dom'.
 * @throws Exception caso haja algum problema na leitura do arquivo
 * ou no formato xml do mesmo.
 */
static Document docRead(String fName) throws Exception{
        File fXmlFile = new File(fName);
        DocumentBuilderFactory dbFactory =
                DocumentBuilderFactory.newInstance();
        DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
        Document doc = dBuilder.parse(fXmlFile);
        return doc;
}
```

- Acesso aos elementos que constituem o documento

```
// le o documento representado no arquivo
Document doc = docRead("alunos.xml");

// pega a lista de alunos contida no documento
NodeList lista = doc.getElementsByTagName("aluno");
System.out.println("tamanho da lista:"+lista.getLength());

// para cada aluno da lista ...
for (int i = 0; i < lista.getLength(); i++){
        Element n = (Element)lista.item(i);
        String nome  = n.getAttribute("nome");
        String doc   = n.getAttribute("doc");
        String ra    = n.getAttribute("ra");
        String curso = n.getAttribute("curso");
        System.out.println ( "aluno nome:" + nome +
                             " doc:" + doc +
                             " ra:" + ra +
                             " curso:" + curso
      );
}
```

- Acesso às 'sub-tags' pelo nome

```
  static void mostraDisciplinas(Element c){
        // pega a lista de nós filhos sob o nome 'disciplina'
        NodeList disciplinas = c.getElementsByTagName("disciplina");
        if(disciplinas.getLength() > 0) {
           System.out.print("     disciplinas cursadas: ");

           // para cada disciplina representada na lista..
           for(int i = 0; i < disciplinas.getLength(); i++){
           Element disc = (Element)disciplinas.item(i);
           String str = disc.getTextContent();// conteúdo da tag
           System.out.print(" "+str);
        }
        System.out.println();
    }
}
```



- SAX: à medida em que o texto XML é lido pelo *parser* este gera eventos, que na verdade são chamadas a funções que devem ser implementadas pela aplicação. Os eventos indicam tipicamente início de *tag* e fechamento de *tag*. Ao acionar o *parser* em modo SAX a aplicação deve informar qual o módulo que será responsável pelo tratamento dos eventos. O parser não mantém nenhuma representação dos dados em memória.

Usando a classe javax.xml.parser.SAXParser:

- Em modo SAX o *parser* opera acoplado a um tratador de eventos, que estende uma classe padrão que acompanha o *parser*, DefaultHandler.
- A classe que estende DefaultHandler deve implementar os métodos responsáveis pelo tratamento dos eventos gerados pelo parser durante o tratamento do texto xml:
  - **`startElement()`**: gerado pelo parser toda vez que a uma nova tag é 'aberta' no texto xml. Entre os parâmetros estão o nome da tag e os atributos.
  - **`endElement()`**: gerado pelo parser ao encontrar o 'fechamento' de uma tag. Entre os parâmetros está o nome da tag.
  - **`characters()`**: gerado pelo parser ao encontrar um texto dentro de uma tag. Tem como parâmetros um vetor de caracteres e os índices dos pontos inicial e final do texto encontrado.

- Exemplo de criação do tratador de eventos

```
static DefaultHandler makeHandler(){
   return new DefaultHandler(){

      public void startElement( String uri, String localName,
                                String tagName, Attributes attribs
                              ) throws SAXException {
        System.out.println("<<"+tagName);
      }

      public void endElement( String uri, String localName,
                              String tagName ) throws SAXException{
        System.out.println("//"+tagName);
      }

      public void characters( char[] ch, int start, int length)
                            throws SAXException{
        String txt = new String(ch, start, length).trim();
        System.out.println("'" + txt + "'");
      }
   }
}
```

- Em uso:

```
public static void main(String argv[]) throws Exception {

  // criação do SAXParser()
  SAXParserFactory factory = SAXParserFactory.newInstance();
  SAXParser saxParser = factory.newSAXParser();

  // criação de um tratador de eventos SAX
  DefaultHandler handler = makeHandler();

  // chamada ao parser, associando o tratador SAX ao mesmo.
  saxParser.parse("alunos.xml", handler);

}
```