Processamento de dados XML

Vanessa Braganholo

Introdução

- Dois pontos básicos:
  - Como escrever um documento XML?
  - Como ler um documento XML?

Escrevendo um documento XML

- Documentos XML
  - Podem ser escritos à mão usando um editor
  - Podem ser gerados automaticamente
    - Troca de dados, protocolos entre aplicativos
    - Importação/exportação entre diferentes formatos de dados (relacional -> xml, xml -> relacional, etc.)
    - Marcação de saída é relativamente fácil:

```
fprintf(sdtout, "<para>paragrafo</para>"\n);
System.ou.printl("<para>paragrafo</para>"n\);
Clibs.puts("<para>paragrafo</para>"\n);
```

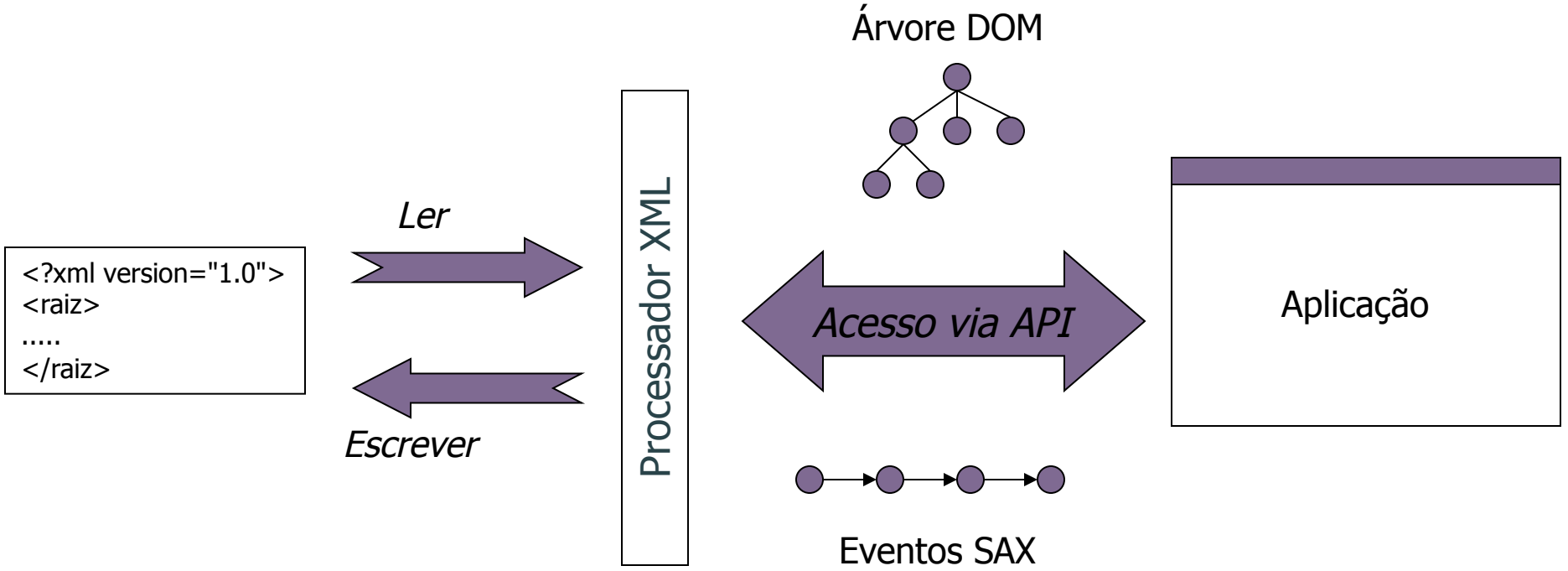
Lendo um documento XML

- Software **pode** ser usado para escrever um documento XML
- Software **deve** ser usado para ler um documento XML
  - Infelizmente a leitura é mais complexa do que a escrita
  - Várias questões:
    - Caracteres como "enter" "espaços em branco" devem ser sempre tratados
    - Substituir as entidades no texto
    - Processar DTD/XML Schema

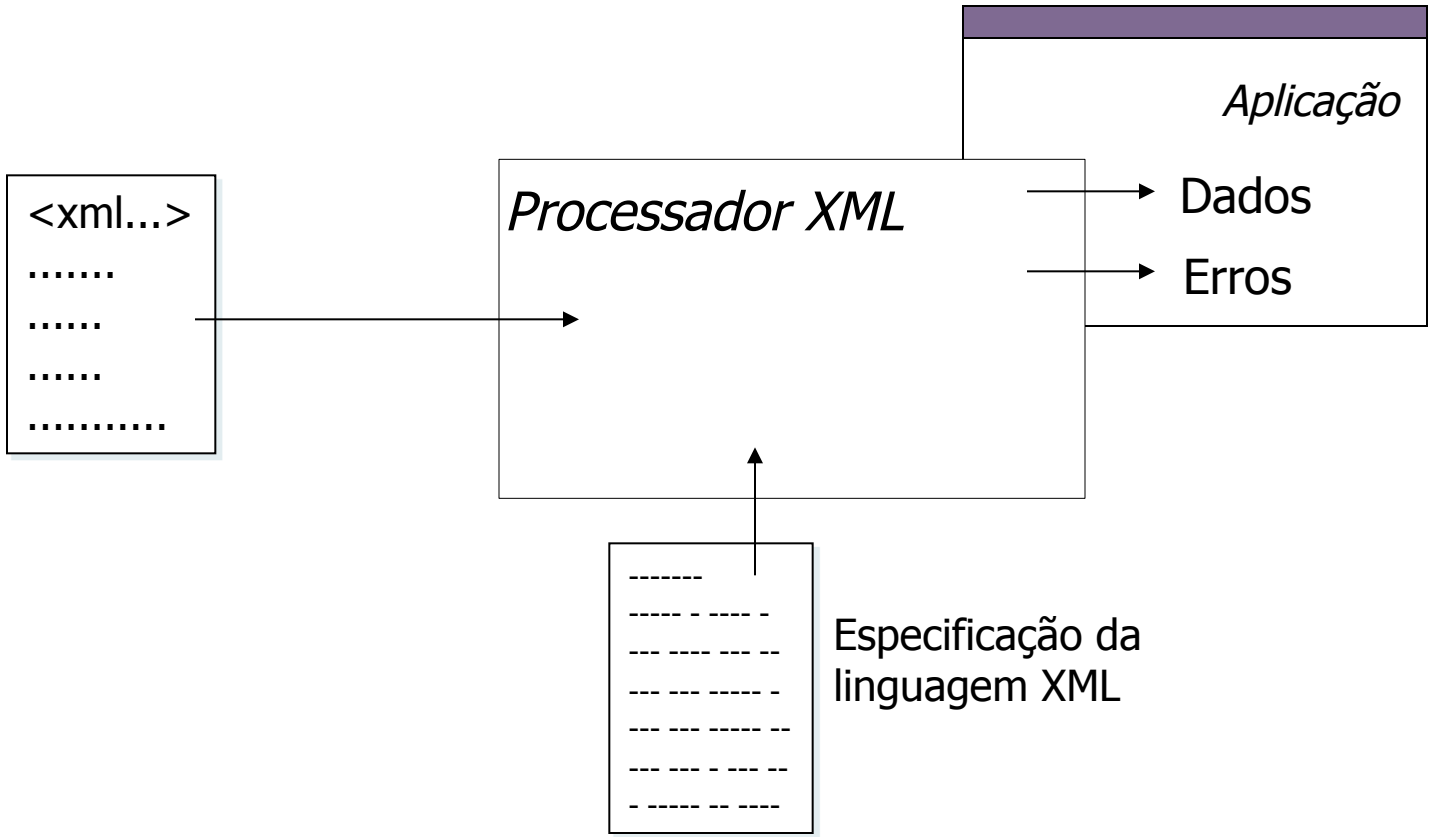
Lendo um documento XML

- Especificação XML -- na W3C --
  - Fornece todas as informações
    - O que deve ser tratado durante a leitura de um documento XML
- Para ler um documento XML
  - Necessidade de um módulo de processamento "XML-sensitive"
    - Processador XML
      - Torna o documento XML acessível por uma aplicação
      - Detecta formatos que não podem ser processados
      - Entidades que não são recursos válidos

Processamento XML



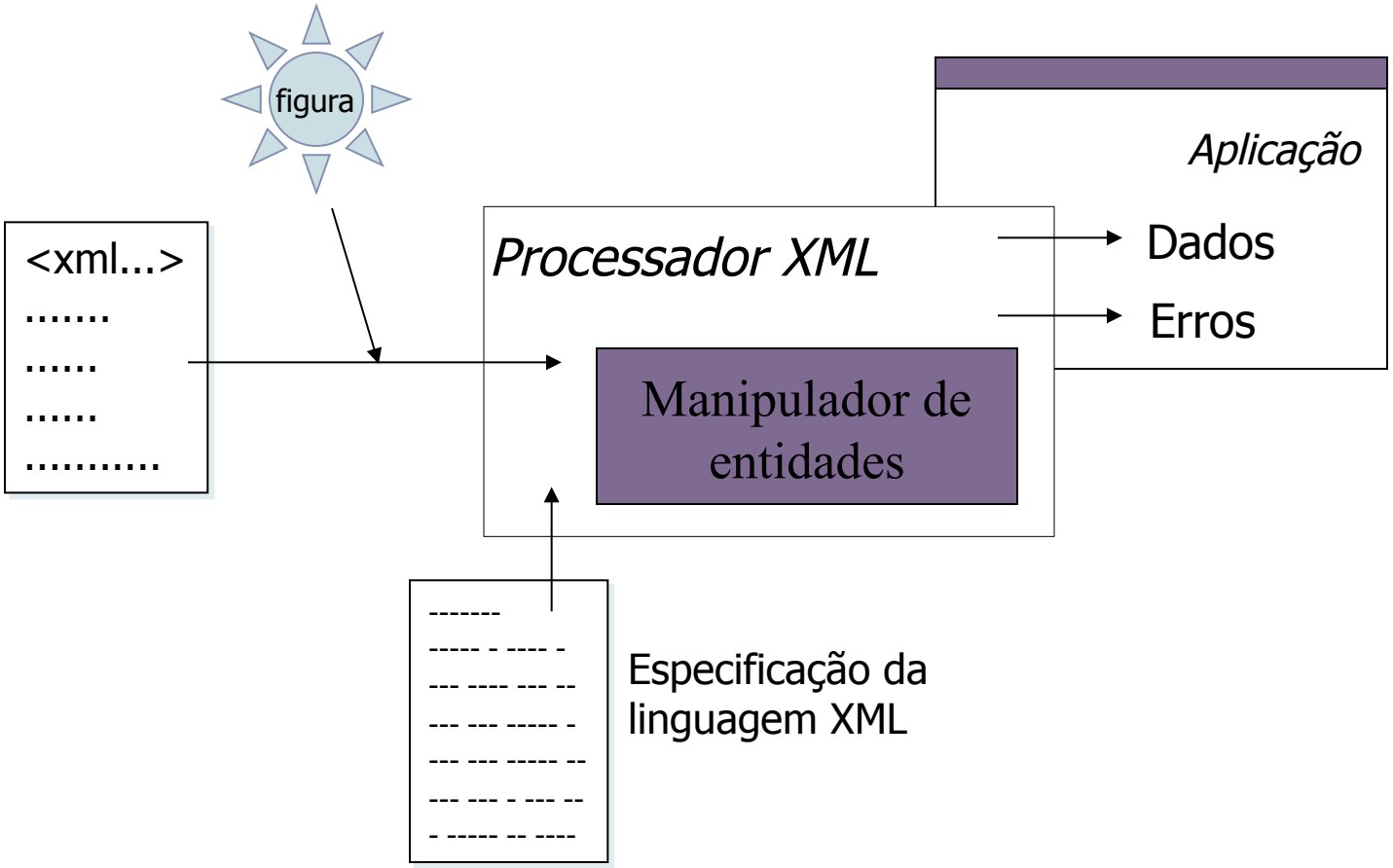
Processador XML



Manipulador de Entidades

- Manipulador de entidades
  - parte do processador XML
    - responsável por localizar fragmentos de documentos
    - responsável por manipular a substituição das referências
- Os fragmentos de documentos podem ser
  - declarações de entidades
  - outros arquivos de dados

Manipulador de Entidades



Parser

- O Parser é a parte do processador XML responsável por verificar a integridade dos dados XML
- Um parsing pode ser executado de dois modos:
  - sem validação
  - com validação.
- Parsing sem validação
  - verifica se o documento é bem formado
- Parsing com validação
  - verifica se o documento é bem formado
  - verifica se o documento é válido

Parser

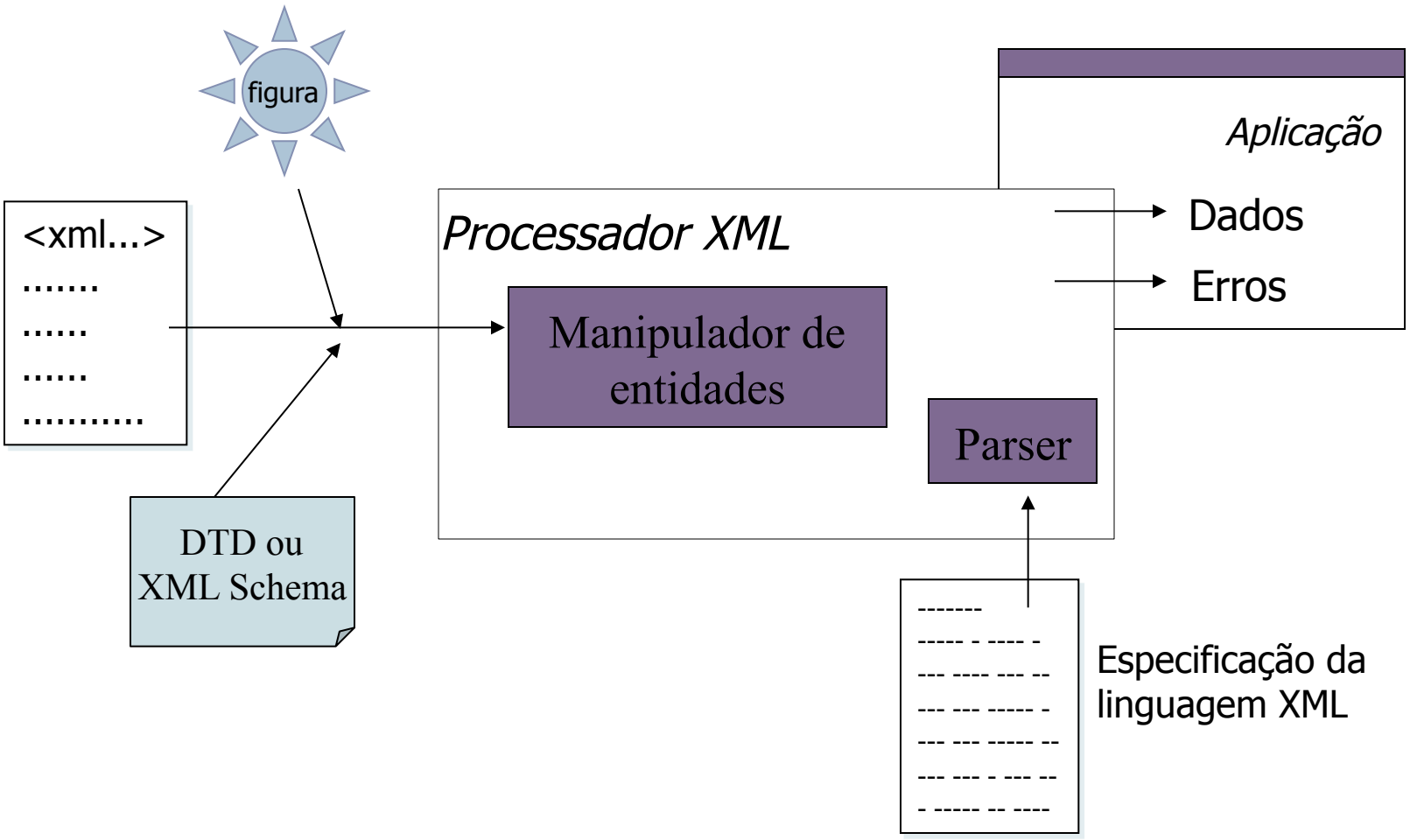


figura
<xml...>
.....
.....
.....
.....
DTD ou
XML Schema
Processador XML
Manipulador de
entidades
Parser
Aplicação
Dados
Erros

----- - ---- -
--- ---- --- --
--- --- ----- -
--- --- ----- --
--- --- - --- --
- ----- -- ----
Especificação da
linguagem XML

Exemplo – Parser Online

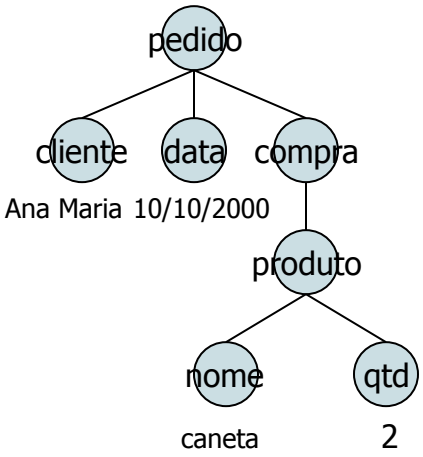
- http://www.w3.org/2001/03/webdata/xsv

APIs e Processadores

- APIs
  - DOM
  - SAX

- Processadores
  - Xerces
  - Oracle XML Processor
  - Microsoft XML Processor
  - Outros...

SAX vs. DOM




```
<?xml version="1.0"?>
<pedido>
  <cliente>Ana Maria</cliente>
  <data>10/10/2000</data>
  <compra>
    <produto>
      <nome>caneta</nome>
      <qtd>2</qtd>
    </produto>
  </compra>
</pedido>
```

```
startDocument
startElement pedido
stabrtElement cliente
caracteres Ana Maria
endElement cliente
startElement data
caracteres 10/10/2000
endElement data
startElement compra
startElement produto
startElement nome
caracteres caneta
endElement nome
startElement qtd
caracteres 2
endElement qtd
endElement produto
endElement compra
endElement pedido
endDocument
```

SAX vs. DOM: como escolher?

- DOM é melhor quando:
  - A estrutura de um documento XML precisa ser modificada
    - Alterar posições de nodos
      - inserir, excluir, ordenar
    - Trocar nodos de uma árvore para outra
  - Compartilhar o documento na memória com outras aplicações
    - As aplicações compartilham a mesma instância do objeto
  - O tamanho do documento não é muito grande
  - A aplicação precisa acessar a mesma parte do documento várias vezes

SAX vs. DOM: como escolher?

- SAX é melhor quando:
  - Questões de memória e performance são críticas
    - Documento é muito grande para ficar em memória
  - A aplicação não precisa reconhecer a estrutura do documento XML
    - SAX "varre" o documento XML uma única vez
    - O status de "qual é o contexto no momento" precisa ser mantido

SAX – Simple API for XML

Introdução

- Padrão desenvolvido na lista de e-mail XML-DEV
- http://www.saxproject.org/
- Apesar de não ter sido desenvolvida por um órgão oficial regulador de padrões, SAX tornou-se um padrão *de facto* e é um software livre para uso privado e comercial
- Primeira versão: maio de 1998

Introdução

- SAX é uma API padrão para processamento de dados XML baseado em eventos
- O *parser* entrega informação para o aplicativo disparando eventos

Call-backs e interfaces

- O aplicativo
  - instancia um objeto *parser*, fornecido por um desenvolvedor
  - manda executar o *parsing* de um documento ou *stream* de dados através do objeto *parser*

- Enquanto processa os dados XML, o *parser*
  - detecta partes significativas tais como *start-tag*, *end-tag*, erros, etc,
  - envia esta informação para o aplicativo utilizando um mecanismo de *call-backs*

Exemplo: *parsing* SAX

```
<pedido>
  <cliente>
    <razao_social>ABC</razao_social>
    <cgc>00.000.000/0001-00</cgc>
  </cliente>
  <itens_pedido>
    <item>
       <produto>caneta azul</produto>
       <quantidade>100</quantidade>
       <preco_unit>2</preco_unit>
    </item>
  </itens_pedido>
</pedido>
```

Exemplo: *parsing* SAX

```
start document
start element: pedido
start element: cliente
start element: razao_social
characters: ABC
end element: razao_social
start element: cgc
characters: 00.000.000/0001-00
...
end element: cliente
start element: itens_pedido
start element: item
start element: produto
characters: caneta azul
end element: produto
...
end document
```

Mecanismo de *call-backs*

1. O aplicativo cria um ou vários objetos cujos métodos são utilizados pelo *parser* para informar a ocorrência de eventos
2. O aplicativo passa referências a estes objetos para o *parser*
3. O *parser* aceita os objetos e os utiliza no processamento dos dados

Interfaces

- Os objetos que o *parser* aceita devem pertencer a uma classe que define uma ou várias interfaces SAX (`org.xml.sax.*`)

- Deste modo, o *parser*
  - sabe que os métodos necessários estão presentes
  - executa chamadas aos métodos apropriados de acordo com a ocorrência dos eventos.

Interfaces SAX 2.0

Interface XMLReader

Implementada pelo parser propriamente dito.

Interface ContentHandler

Manipula eventos básicos de marcação - início de documento, início de elemento, etc.

Interface Attributes

Manipula atributos

Interface ErrorHandler

Manipula erros

Interface EntityResolver

manipula entidades

Interfaces SAX 2.0 (cont.)

Interface Locator

- localização de cada erro - linha/coluna

Interface DTDHandler

- Eventos relacionados a DTDs

Interface XMLFilter

- Filtro XML

Classe DefaultHandler (SAX 2.0)

- Implementa as interfaces

  EntityResolver
  DTDHandler
  ContentHandler
  ErrorHandler

  com um comportamento default.

Outras classes da API SAX 2.0

Classe SAXNotSupportedException

Classe SAXNotRecognizedException

Classe SAXException

Encapsula erros e warnings gerais de SAX

Classe SAXParseException

Encapsula erros e warnings de parsing

Hierarquia de Interfaces

interface org.xml.sax.Attributes

interface org.xml.sax.ContentHandler

interface org.xml.sax.DTDHandler

interface org.xml.sax.EntityResolver

interface org.xml.sax.ErrorHandler

interface org.xml.sax.Locator

interface org.xml.sax.XMLReader

- interface org.xml.sax.XMLFilter

Hierarquia de Classes

class java.lang.Object

- class org.xml.sax.helpers.AttributesImpl (implements org.xml.sax.Attributes)
- class org.xml.sax.helpers.DefaultHandler (implements org.xml.sax.ContentHandler, org.xml.sax.DTDHandler, org.xml.sax.EntityResolver, org.xml.sax.ErrorHandler)
- class org.xml.sax.helpers.LocatorImpl (implements org.xml.sax.Locator)
- class org.xml.sax.helpers.NamespaceSupport
- class org.xml.sax.helpers.ParserAdapter (implements org.xml.sax.DocumentHandler, org.xml.sax.XMLReader)
- class org.xml.sax.helpers.XMLFilterImpl (implements org.xml.sax.ContentHandler, org.xml.sax.DTDHandler, org.xml.sax.EntityResolver, org.xml.sax.ErrorHandler, org.xml.sax.XMLFilter)
- class org.xml.sax.helpers.XMLReaderAdapter (implements org.xml.sax.ContentHandler, org.xml.sax.Parser)
- class org.xml.sax.helpers.XMLReaderFactory

Hierarquia de Classes (p/ Exceções)

class java.lang.Object

- class java.lang.Throwable (implements java.io.Serializable)
  - class java.lang.Exception
    - class org.xml.sax.SAXException
      - class org.xml.sax.SAXNotRecognizedException
      - class org.xml.sax.SAXNotSupportedException
      - class org.xml.sax.SAXParseException

Atividade não contemplada pelo SAX

- Escrever uma estrutura em formato XML
- Essa característica, quando implementada, será diferente em cada *parser*

Construindo uma aplicação JAVA

- API XML para Java: JAXP
  - http://java.sun.com/xml/downloads/jaxp.html
- Usaremos a classe ***DefaultHandler***
  - Implementação com características default
  - Criar uma classe que **ESTENDE** a classe **DafaultHandler**…
  - Além disso, precisamos de alguma implementação do PARSER propriamente dito (aquele que lê o documento e dispara os eventos para a nossa aplicação)
    - Em Java, o parser já vem no JAXP, e é a classe **SAXParser**
    - Uma instância da classe SAXParser pode ser obtida através da Factory **SAXParserFactory**
  - Em outras linguagens, a implementação do parser provavelmente estará disponível em alguma classe que implementa a interface XMLReader

Construindo uma aplicação JAVA

```
import java.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;

public class MySAXApp extends DefaultHandler {

public MySAXApp () {
   super();
}
```

Acompanhem o exemplo no arquivo MySAXApp.java, no diretório **exercicio1**

Documentação JAVA:
http://docs.oracle.com/javase/7/docs/api/index.html

Classe main

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

  if (argv.length != 1) {
    System.err.println("Modo de usar: MySAXApp <arquivo XML>");
    System.exit(1);
  }
  // Informa quem é o gerenciador de eventos SAX
  DefaultHandler handler = new MySAXApp();

  // Usa o parsing default (sem validação)
  SAXParserFactory factory = SAXParserFactory.newInstance();

  try {
    // Faz o parsing
    SAXParser saxParser = factory.newSAXParser();
    saxParser.parse( new File(argv[0]), handler );
  } catch (Throwable t) {
    t.printStackTrace();
  }
  System.exit(0);
}
```

Classe main

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

  if (argv.length != 1) {
    System.err.println("Modo de usar: MySAXApp <arquivo XML>");
    System.exit(1);
  }
  // Informa quem é o gerenciador de eventos SAX
  DefaultHandler handler = new MySAXApp();

  // Usa o parsing default (sem validação)
  SAXParserFactory factory = SAXParserFactory.newInstance();

  try {
    // Faz o parsing
    SAXParser saxParser = factory.newSAXP     r();
    saxParser.parse( new File(argv[0]), ha      );
  } catch (Throwable t) {
    t.printStackTrace();
  }
  System.exit(0);
}
```

O handler é uma instância da aplicação que estamos construindo!

Exercício 1

- Compilar a classe MySAXApp (dentro de exercicio1) e executá-la usando o arquivo pedido1.xml como parâmetro
- Inserir um erro no arquivo pedido1.xml e executar a aplicação novamente

- Para compilar: javac MySAXApp.java
- Para executar: java MySAXApp pedido1.xml

- Se o javac não estiver funcionando, adicione C:\Arquivos de programas\Java\jdk1.6.0_21\bin no PATH (Painel de Controle, Sistemas, Avançado, Variáveis de Ambiente)

Interface XMLReader

- Seus métodos dividem-se em três grupos
- Métodos do tipo `set`
  - usados pela aplicação para registrar objetos no *parser* que correspondem a outras interfaces
- Métodos do tipo `parse`
  - métodos deste grupo são utilizados pelo aplicativo para executar o *parsing*
- Métodos do tipo `get`

Interface XMLReader (parse)

void parse(InputSource input)

Faz o parsing de um documento XML.

void parse(String systemId)

Faz o parsing de um documento XML a partir de um identificador de sistema (URI).

Importantes para quem vai usar outra linguagem de programação que não Java

Interface XMLReader (set)

void setContentHandler(ContentHandler handler)
Permite que uma aplicação registre um ContentHandler

void setDTDHandler(DTDHandler handler)
Permite que uma aplicação registre um ContentHandler

void setEntityResolver(EntityResolver resolver)
Permite que uma aplicação registre um EntityResolver

void setErrorHandler(ErrorHandler handler)
Permite que uma aplicação registre um ErrorHandler

void setFeature(String name, boolean value)
Seta o valor de uma característica

void setProperty(String name, Object value)
Seta o valor de uma propriedade

Interface XMLReader (get)

ContentHandler getContentHandler()
Retorna o ContentHandler

DTDHandler getDTDHandler()
Returna o DTDHandler

EntityResolver getEntityResolver()
Retorna o EntityResolver

ErrorHandler getErrorHandler()
Retorna o ErrorHandler

Boolean getFeature(String name)
Retorna o valor de uma característica

Object getProperty(String name)
Retorna o valor de uma propriedade

Funcionamento

- Depois de registrar um ou mais objetos no *parser*, a aplicação chama um dos métodos `parse`
- O *parser* começa a ler os dados XML
- Quando um objeto significativo é encontrado, o *parser* pára e a informação é enviada para o aplicativo chamando o método apropriado através de um dos objetos registrados
- O *parser* espera o método retornar para continuar o processamento.

Para manipular o conteúdo do documento XML...

- O programador deve implementar os métodos da interface ContentHandler
- Através desses métodos, o aplicativo trata os eventos disparados pelo parser

Interface ContentHandler

void characters(char[] ch, int start, int length)
Recebe notificação do recebimento de caracteres (conteúdo de um elemento)

void endDocument()
Recebe notificação de final de documento

void endElement(String uri, String localName, String qName)
Recebe notificação de final de elemento

void processingInstruction(String target, String data)
Recebe notificação de instrução de processamento

void startDocument()
Recebe notificação de início de documento

void startElement(String uri, String localName, String qName, Attributes atts)
Recebe notificação de início de elemento

Interface ContentHandler

void endPrefixMapping(String prefix)
Termina o escopo de um mapeamento de prefixo

void ignorableWhitespace(char[] ch, int start, int length)
Recebe notificação de espaço em branco ignorável

void setDocumentLocator(Locator locator)
Recebe um objeto para localizar a origem dos eventos SAX

void skippedEntity(String name)
Recebe notificação de uma entidade ignorada

void startPrefixMapping(String prefix, String uri)
Inicia o escopo de um mapeamento de prefixo de um namespace

Exemplo

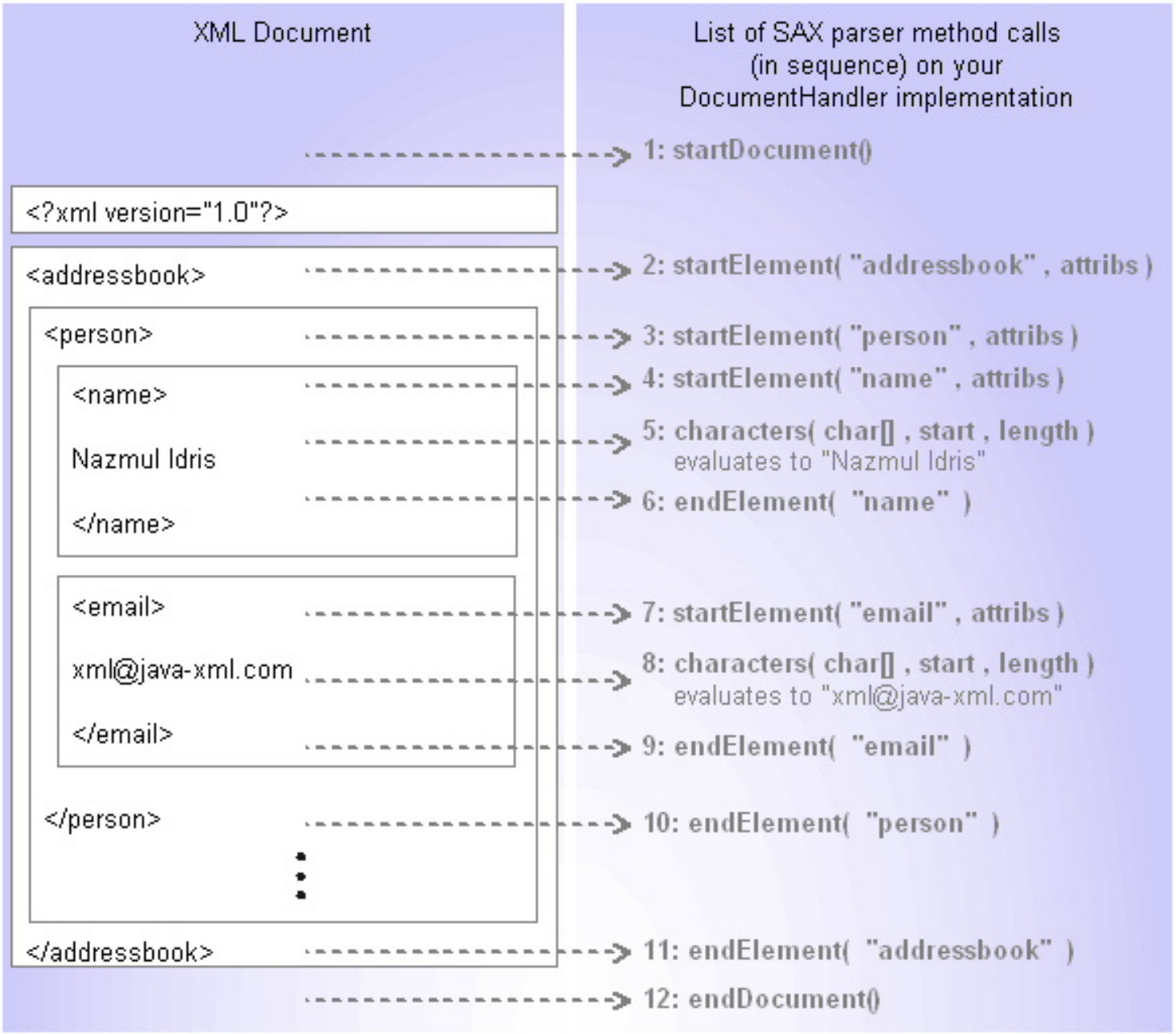


Figure 1 : SAX DocumentHandler interface methods and their sequence

startDocument e endDocument

- O primeiro método chamado é startDocument e o último é endDocument
- Métodos úteis para inicializar variáveis, abrir e fechar arquivos, etc

```
public void startDocument() {
  total_pedido=0;
}
```

Exercício 2

- Compilar a classe MySAXApp (dentro de exercicio2) e executá-la usando o arquivo pedido1.xml como parâmetro
- Modificar a classe para adicionar o método endDocument que imprime "final do documento" na saída

startElement e endElement

- O método startElement é chamado quando uma start-tag é encontrada no fluxo de dados
- O método endElement é chamado quando uma end-tag é encontrada no fluxo de dados.

XML que está sendo processado...

```
<pedido numero="1000">
 <cliente>
   <razao_social>ABC</razao_social>
   <cgc>00.000.000/0001-00</cgc>
 </cliente>
 <itens_pedido>
   <item>
     <produto>caneta azul</produto>
     <quantidade>100</quantidade>
     <preco_unit>2</preco_unit>
   </item>
   <item>
     <produto>caneta preta</produto>
     <quantidade>200</quantidade>
     <preco_unit>1</preco_unit>
   </item>
 </itens_pedido>
</pedido>
```

Exemplo

```
public void startElement (String uri, String localName, String
   qName, Attributes atts) {
        pilha.push(qName);
   }
```

Exemplo

URI do Namespace, ou string vazia se não houver namespace, ou se os namespaces não estiverem sendo processados

```
public void startElement (String uri, String localName, String
    qName, Attributes atts) {
        pilha.push(qName);
    }
```

Nome local (sem prefixo de namespace), ou string vazia se os namespaces não estiverem sendo processados

Nome qualificado (com prefixo do namespace), ou string vazia se nome qualificado não estiver disponível

Exemplo

```
public void startElement (String uri, String localName, String
   qName, Attributes atts) {
        pilha.push(qName);
   }

public void endElement (String uri, String localName, String
   qName) throws SAXException {
        pilha.pop();
   }
```

(pilha é um objeto do tipo Stack)

characters

- O método characters é chamado quando um string de texto é encontrado. O exemplo abaixo converte o array de entrada para um string simples e armazena o nome do cliente.

```
public void characters (char[] ch, int start, int length) throws
   SAXException {
   if (pilha.peek().equals("razao_social"))
         String cliente = new String(ch,start,length);
```

Exercício 3

- Utilizar a classe MySAXApp (dentro de exercicio3) como base, e modificá-la para que ela processe o pedido pedido1.xml mostrando a seguinte saída:

  CLIENTE: ABC
  CGC: 00.000.000/0001-00
  PRODUTO: caneta azul
  PRODUTO: caneta preta

- Atenção!! Todos os métodos necessários já estão declarados! Basta colocar a implementação deles!!!

ignorableWhitespace

- Este método é chamado quando um string de caracteres que podem ser ignorados é encontrado
  - Espaços em branco que aparecem em elementos que somente podem conter elementos são considerados caracteres ignoráveis
  - Quando um esquema não é utilizado este método não é chamado, pois o parser não é capaz de distinguir entre elementos que podem conter texto e elementos que só podem conter outros elementos

processingInstruction

- processingInstruction
  - este método é chamado quando uma instrução de processamento é encontrada
  - o método recebe como parâmetros
    - nome da aplicação de destino
    - instrução de processamento

Exemplo

```
public void processingInstruction(String target, String data)
{
  if ( target.equals("ACME") )
  {
    // o aplicativo é o processador ACME
    if ( data.equals("new_page") )
    {
      // quebra de página aqui...
    } ...
   }
}
```

<?ACME new_page?>

Exercício 4

- Utilizar a classe MySAXApp (dentro de exercicio4) como base e modificá-la para encontrar as instruções de processamento do arquivo view1.xml e as exibir na tela
- Atenção!! Exibir somente as instruções de processamento que contêm instruções SQL!!

Atributos

- Quando o parser informa para a aplicação que uma star-tag foi encontrada, ele chama o método startElement
- Uma start-tag pode conter um ou mais atributos

Atributos

- Problema:
  - não existe limite no número de atributos que um elemento pode conter
  - portanto, esta informação não pode ser passada como parâmetro
- Solução:
  - criar um objeto para encapsular todos os detalhes dos atributos
  - este objeto deve implementar a interface *Attributes*

Interface Attributes

int getIndex(java.lang.String qName)
Retorna o índice de um atributo (sua posição no elemento) de nome "qName"

int getIndex(java.lang.String uri, java.lang.String localName)
Retorna o índice de um atributo (sua posição no elemento) de nome "uri":"localName"

int getLength()
Retorna o número de atributos da lista

java.lang.String getLocalName(int index)
Retorna o nome do atributo na posição indicada pelo parâmetro "index"

java.lang.String getQName(int index)
Retorna o nome do atributo na posição indicada pelo parâmetro "index"

java.lang.String getType(int index)
Retorna o tipo do atributo na posição indicada pelo parâmetro "index"

Interface Attributes

java.lang.String getType(java.lang.String qName)
Retorna o tipo do atributo de nome "qName"

java.lang.String getType(java.lang.String uri,
java.lang.String localName)
Retorna o tipo do atributo de nome "uri":"localName"

java.lang.String getURI(int index)
Retorna o namespace do atributo na posição "index"

java.lang.String getValue(int index)
Retorna o valor do atributo na posição "index"

java.lang.String getValue(java.lang.String qName)
Retorna o valor do atributo de nome "qName"

java.lang.String getValue(java.lang.String uri,
java.lang.String localName)
Retorna o valor do atributo de nome "uri":"localName"

Métodos getLength e getQName

- getLength
  - retorna um valor inteiro que representa o número de atributos que o elemento possui
  - o valor zero indica que não existem atributos
  - cada atributo é identificado por um valor de índice, começando em zero.

- getQName
  - recebe um valor de índice e retorna o nome do atributo correspondente.

Exemplo

- Recupera o nome do último atributo

```
String lastAttribute =  null;
int totalAtts = atts.getLength();
if ( totalAtts > 0 )
    lastAttribute = atts.getQName(totalAtts - 1);
```

Método getValue

- getValue
  - retorna o valor de um atributo
  - é possível utilizar tanto um valor de índice quanto um nome de atributo como parâmetro.

```
lastAttValue = atts.getValue(totalAtts - 1);
```

Método getType

- getType
  - retorna informações sobre o tipo do atributo
  - quando uma DTD é utilizada, cada atributo possui um tipo de dados (CDATA, ID ou NMTOKEN)
  - quando o parser não possui acesso à DTD, o tipo default (CDATA) é utilizado

Exemplo

- Reconstrói a lista de atributos original (declaração da DTD)

```
public void startElement (String uri, String localName, String qName,
    Attributes atts)
     {
  System.out.print( "<!ATTLIST " + qName + " " );
  for( int i =0; i < atts.getLength(); i++ )
  {
    System.out.print( atts.getQName(i) + " " +
                      atts.getType(i) + " " +
                      "#IMPLIED \"" +
                      atts.getValue(i) + "\" \n" );
  }
  System.out.print( "> \n" );
}
```

Método getValue

- getValue
  - recupera o valor de um atributo nomeado
  - o nome do atributo deve ser passado como parâmetro
  - se o atributo não existe, um valor nulo é retornado

Exemplo

- Recupera o valor do atributo numero do elemento pedido

```
public void startElement (String uri, String localName,
   String qName, Attributes atts)
{
    if (qName.equals("pedido"))
          num_pedido = atts.getValue("numero");
}
```

Exercício 5

- Utilizar a classe MySAXApp (dentro de exercicio5) como base e modificá-la para processar o pedido pedido1.xml, exibindo as seguintes informações:

****

PEDIDO NUMERO 1000 PROCESSADO!

CLIENTE: ABC

VALOR TOTAL: 400

****

Interface ErrorHandler

- Um aplicativo implementa a interface ErrorHandler para ser informado sobre erros e warnings
- Métodos:

void error(SAXParseException exception)
Recebe notificação sobre um erro recuperável

void fatalError(SAXParseException exception)
Recebe notificação sobre um erro irrecuperável

void warning(SAXParseException exception)
Recebe notificação sobre um warning

Exemplos de tipo de erro

- ERROR: erros de validação
- FATAL ERROR: erro de má-formação do XML
- WARNING: falha ao carregar alguma entidade externa

- Atenção: Um erro fatal causa uma excessão no método parse por default, e o evento fatalError não é disparado. Para forçar o disparo do evento, adicionar a linha factory.setFeature("http://apache.org/xml/features/continue-after-fatal-error", true)

Exemplo

- Mostra mensagem cada vez que um warning é disparado

```
public void warning (SAXParseException exception)
{
  System.out.print( "WARNING: " + exception);
}
```

Suporte a namespaces

- Para saber se o processamento de namespaces está setado:

```
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser saxParser = factory.newSAXParser();
saxParser.isNamespaceAware();
```

- Para setar o processamento de namespaces:

```
SAXParserFactory factory = SAXParserFactory.newInstance();
factory.setNamespaceAware(true);
```

Veja classe exemplo no diretório namespaces

Parser com validação – na JAXP

- Para fazer um parser com validação

```
// Usa o parsing com validação
SAXParserFactory factory = SAXParserFactory.newInstance();
factory.setValidating(true);
```

- Implementar o método error para exibir os erros de validação:

```
public void error(SAXParseException err)
{
  System.out.println( "ERROR (" + err + ")");
}
```

Validação com XML Schema - JAXP

- Declaração de constantes (só para facilitar, não é necessário)

```
static final String JAXP_SCHEMA_LANGUAGE = "http://java.sun.com/xml/
    jaxp/properties/schemaLanguage";
static final String W3C_XML_SCHEMA = "http://www.w3.org/2001/
    XMLSchema";
```

- No método main da classe:

```
SAXParserFactory factory = SAXParserFactory.newInstance();
factory.setNamespaceAware(true);
factory.setValidating(true);
SAXParser saxParser = factory.newSAXParser();
saxParser.setProperty(JAXP_SCHEMA_LANGUAGE, W3C_XML_SCHEMA);
```

Validação – outras linguagens

- A interface XMLReader tem o método setFeature que é usado, entre outras coisas, para setar a validação
  - Setar a "feature" validation para TRUE

```
xr.setFeature("http://xml.org/sax/features/validation", true);
```

  - (xr é a referência para a classe que implementa a interface XMLReader)

- Implementar o método error para exibir os erros de validação

```
public void error(SAXParseException err)
{
  System.out.println( "ERROR (" + err + ")");
}
```

Exercício 6

- Utilizar a classe MySAXApp (dentro de exercicio6) como base e modificá-la para processar o pedido pedido2.xml com validação
- Inserir erros no documento (em relação à DTD) e executar o parsing

- Vejam o que acontece quando vocês não implementam o método error()

Tratamento de Erros

- Os erros podem ser tratados de forma mais especializada usando os métodos: error, fatalError e warning…

Outras interfaces

- Mais específicas, usadas por quem precisa implementar um parser mais complexo
- Interface Locator
  - Identifica o número de linha e o número do caractere que originou um erro no parsing
- Interface DTDHandler
  - Informa sobre entidades binárias e declaração de notações
- Interface EntityResolver
  - Intercepta referências a entidades

- Detalhes sobre elas podem ser encontrados em http://www.saxproject.org/apidoc/org/xml/sax/package-summary.html

Exercício 7

- Modificar a classe MySAXApp (dentro de exercicio7) para exibir a nota fiscal referente ao pedido.
- Processe com o documento pedido3.xml, que possui dados do endereço do cliente.

```
ABC                              00.000.000/0001-00
Rua das Flores, 75               Porto Alegre   RS
-------------------------------------------------------
Produto                          Quant  P.Unit.  P.Total
-------------------------------------------------------
caneta azul                      100          2      200
papel                            100          8      800
-------------------------------------------------------
                                                    1000
```

Exercício 8

- Imprimir na tela o documento XML que está sendo lido. O arquivo *MySAXApp* (dentro de exercicio8) já contém a declaração dos métodos necessários
- Testar com *pedido3.xml* e *view1.xml*

Fontes de Referência:

- http://www.saxproject.org/apidoc/org/xml/sax/package-summary.html (Documentação de todas as interfaces (em Java))
- Excelente tutorial da SUN sobre JAXP: http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JAXPSAX.html