DOM – Document Object Model

Vanessa Braganholo

DOM

- API padrão para processamento de dados XML baseado em um modelo de árvore
  - o parser constrói na memória um objeto representando a árvore XML (objeto DOM)
- Padrão desenvolvido pela W3C
- Define uma interface para a construção e tratamento de instâncias de documentos

DOM

- DOM foi projetado orientado a objetos
  - assume o uso de linguagens com suporte a programação orientada a objetos (como Java ou C++)
- O padrão é composto por um conjunto de interfaces
  - em Java as interfaces encontram-se definidas no pacote org.w3c.dom
- Cada processador particular implementa estas interfaces
  - o desenvolvedor de aplicações importa o pacote e usa as classes

Exemplo: implementação Java (JAXP)

- JavaDoc:
  - http://download.oracle.com/javase/6/docs/api/index.html

Exemplo: uso do *parser* da SUN

```
try
  {
  //Instancia o parser
    DocumentBuilderFactory b =
    DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = b.newDocumentBuilder();



    //Faz o parsing do documento
    Document myDoc = builder.parse("meuDoc.xml");
}
catch( Exception e ) {...}
```

Exemplo: uso do *parser* da SUN

- A classe para executar o *parsing* é DocumentBuilder
- O método parse
  - executa o *parsing*
  - constrói o objeto DOM na memória
  - retorna uma referência ao nodo documento

Nodos

- DOM define interfaces para manipular nodos
- Nodos
  - Representam elementos, texto, comentários, instruções de processamento, seções CDATA, referências a entidades, declarações de entidades, declarações de notações e documentos inteiros
  - São usados também para representar atributos de um elemento

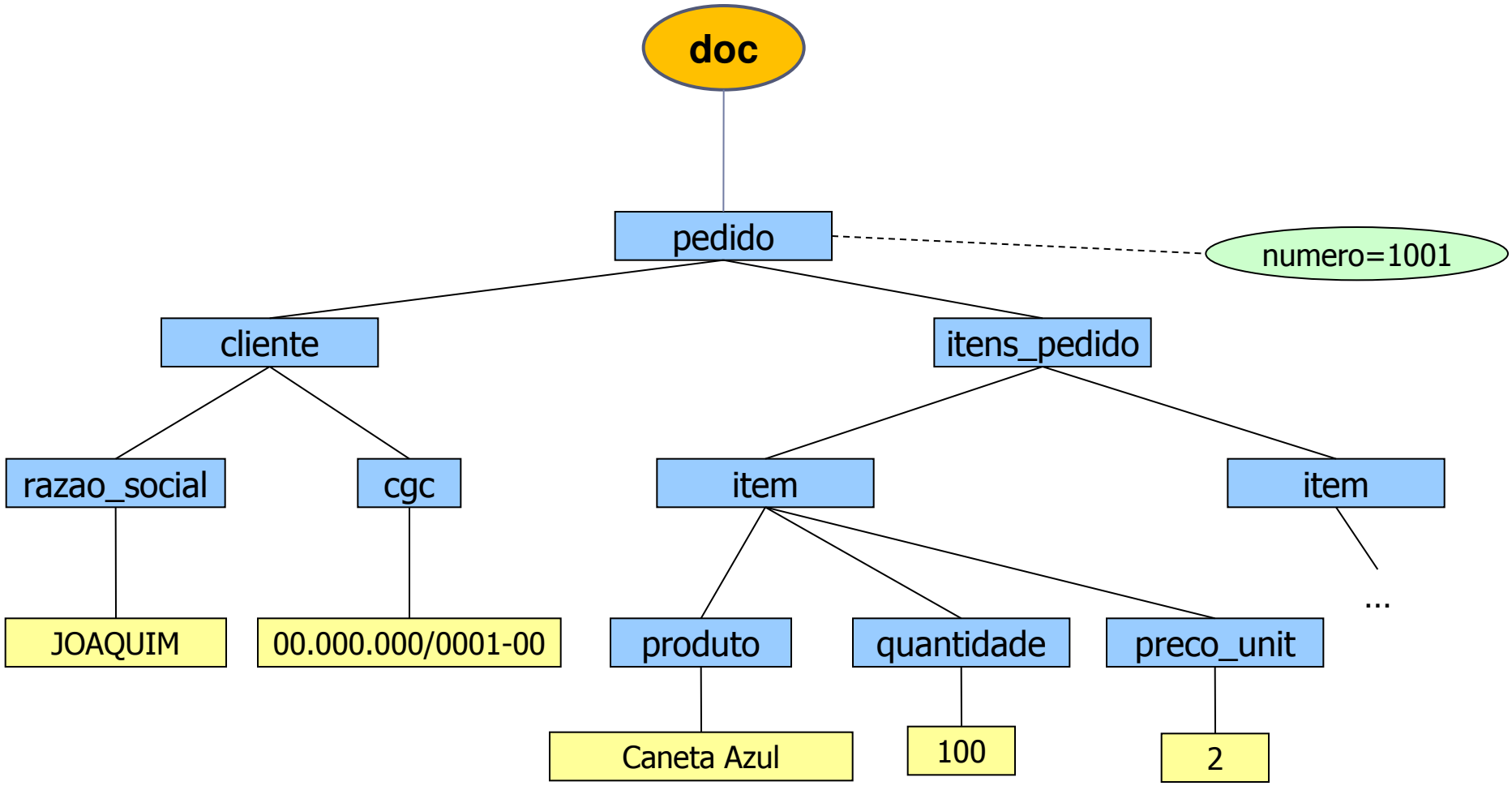
Nodos

- Existem tipos de dados adicionais para facilitar a manipulação de tipos de nodos
  - Dois tipos de listas de nodos
  - Um tipo para transferência de nodos entre diferentes partes de um documento

Exemplo de Documento XML

```
<!DOCTYPE pedido SYSTEM “pedido.dtd”>
<pedido numero="1001">
 <cliente>
  <razao_social>JOAQUIM</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>3</preco_unit>
  </item>
 </itens_pedido>
</pedido>
```

Árvore DOM correspondente



Árvore DOM correspondente

- Execução de ProcessaDocumentoDOM.java em exemplo1

  java ProcessaDocumentoDOM teste1.xml

- Para cada nodo do documento, ele gera:

  System.out.print(n.getNodeName() + "=>" + n.getNodeValue() + " ");

```
<raiz>
    <primeiro_filho>
        <primeiro_neto>texto primeiro neto</primeiro_neto>
         <segundo_neto>texto segundo neto</segundo_neto>
    </primeiro_filho>
    <segundo_filho>
         <terceiro_neto>texto terceiro neto</terceiro_neto>
         <quarto_neto>
             <primeiro_bisneto>texto primeiro bisneto</primeiro_bisneto>
         </quarto_neto>
    </segundo_filho>
</raiz>
```

```
<raiz>
    <primeiro_filho>
        <primeiro_neto>texto primeiro neto</primeiro
         <segundo_neto>texto segundo neto</segun
    </primeiro_filho>
    <segundo_filho>
        <terceiro_neto>texto terceiro neto</terceiro_
        <quarto_neto>
            <primeiro_bisneto>texto primeiro bisneto<
        </quarto_neto>
    </segundo_filho>
</raiz>
```

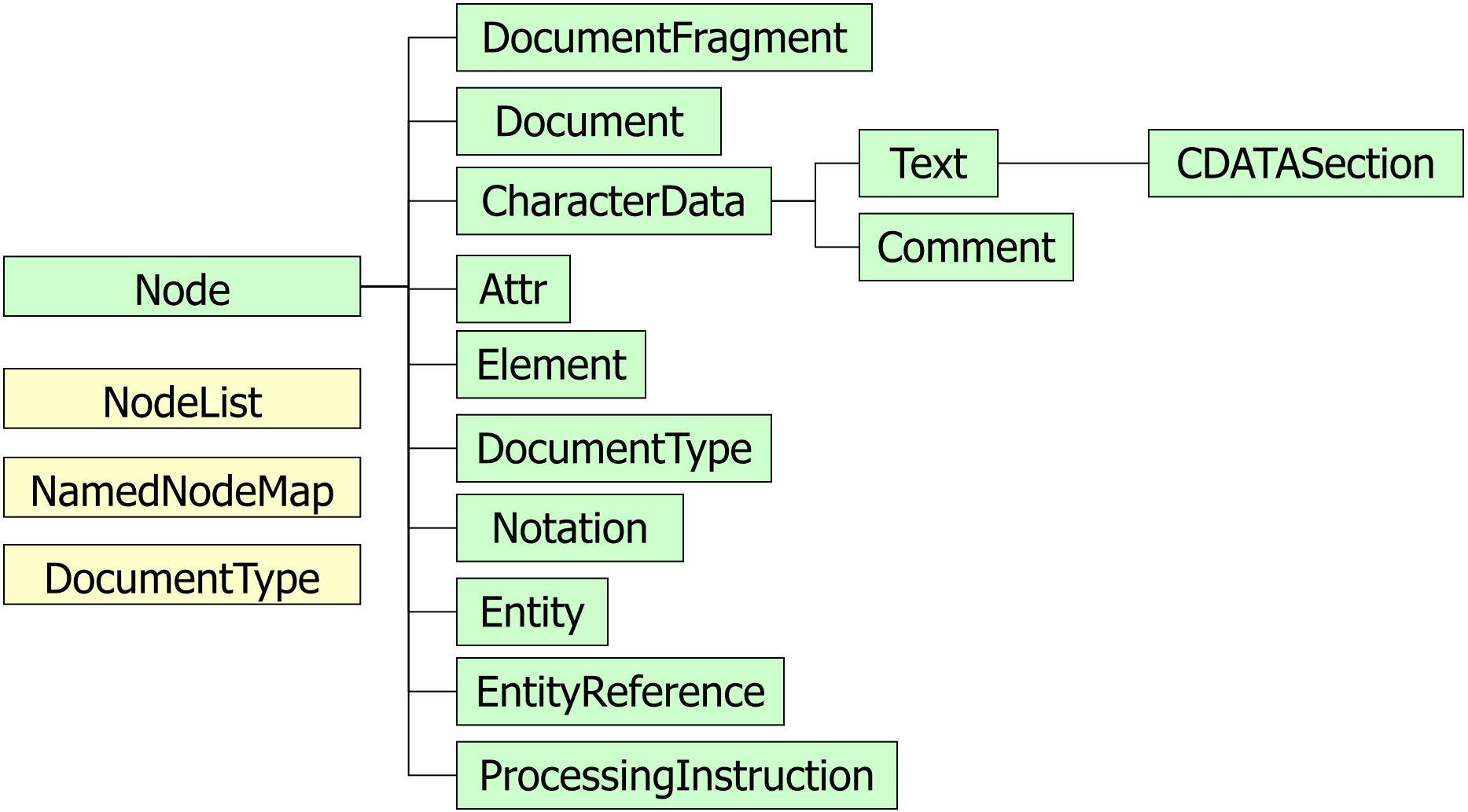
raiz=>null
- **#text=>**
- **primeiro_filho**=>null
  - **#text=>**
  - **primeiro_neto**=>null
    - #text=>texto primeiro neto
    - **#text=>**
  - **segundo_neto**=>null
    - #text=>texto segundo neto
    - **#text=>**
  - **#text=>**
- **segundo_filho**=>null
  - **#text=>**
  - **terceiro_neto**=>null
    - #text=>texto terceiro neto
    - **#text=>**
  - **quarto_neto**=>null
    - **#text=>**
    - **primeiro_bisneto**=>null
      - #text=>texto primeiro bisneto
      - **#text=>**
    - **#text=>**
- **#text=>**

Notem os espaços em branco que não puderam ser ignorados devido à falta da DTD!!

Nodos

- Cada nodo possui:
  - Características: tipo, nome, valor
  - Localização contextual na árvore do documento
  - Acesso a seus parentes: pai, irmãos, filhos
  - Capacidade de alterar seu conteúdo
    - os nodos que representam seus filhos

Interfaces DOM



Interfaces DOM

- Interface Node
  - representa o ponto de partida no esquema DOM
- Interface NodeList
  - Fornece uma abstração de uma coleção ordenada de nodos
- Interface NamedNodeMap
  - Representa coleções de nodos que podem ser acessados pelo nome (basicamente, atributos)
- O restante das interfaces (sub-interfaces de Node) provêem funcionalidades específicas ao tipo de objeto que representam

Interfaces DOM

- Interface Node
  - representa o ponto de partida no esquema DOM
- Interface NodeList
  - Fornece uma abstração de uma coleção ordenada de nodos
- Interface NamedNodeMap
  - Representa coleções de nodos que podem ser acessados pelo nome (basicamente, atributos)
- O restante das interfaces (sub-interfaces de Node) provêem funcionalidades específicas ao tipo de objeto que representam

Interface Node

- Métodos relacionados:
  - características de um nodo (nome, tipo, valor, etc.)
  - Navegação na árvore (pai, filhos, irmãos, etc.)
  - Manipulação de nodos (inserir, atualizar, remover, etc.)

Interface Node – Características dos Nodos

- Cada nodo possui um tipo
  - Elemento
  - Atributo
  - Texto
  - Comentário
  - Instrução de Processamento
  - etc.

Interface Node – Características dos Nodos

- Cada nodo possui um nome
  - Se o nodo é do tipo elemento → nome do nodo é o nome do elemento
  - Se nodo é do tipo comentário → nome é um valor constante (#comment)
  - Se nodo é do tipo texto → nome é um valor constante (#text)
  - Se nodo é do tipo atributo → nome é o nome do atributo

Interface Node – Características dos Nodos

- O nome de um nodo NÃO representa um identificador único
  - um nodo individual pode ser identificado de forma única somente por sua localização na árvore do documento

Interface Node – Características dos Nodos

- Muitos tipos de nodos não podem ter filhos
  - exemplo:
    - Comentário
    - Instrução de Processamento
    - Atributo
- Os nodos que podem ter filhos:
  - elementos
  - documentos
  - fragmentos de documentos

Interface Node – Características dos Nodos

NamedNodeMap **getAttributes**()

Retorna a lista de atributos do nodo (se ele for um elemento) ou null caso contrário.

NodeList **getChildNodes**()

Retorna uma lista de todos os filhos do nodo.

String **getLocalName**()

Retorna o nome (sem o prefixo do namespace) do nodo.

String **getNamespaceURI**()

Retorna a URI do namespace do nodo, ou null se o namespace não tiver sido especificado.

String **getNodeName**()

Retorna o nome do nodo, dependendo do tipo do nodo (veja tabela em http://java.sun.com/j2se/1.5.0/docs/api/org/w3c/dom/Node.html).

short **getNodeType**()

Retorna um código que representa o tipo do nodo.

Interface Node – Características dos Nodos

String **getNodeValue**()

Retorna o valor do nodo, dependendo do seu tipo. (veja tabela em http://java.sun.com/j2se/1.5.0/docs/api/org/w3c/dom/Node.html).

Document **getOwnerDocument**()

Retorna o documento ao qual o nodo está associado.

String **getPrefix**()

Retorna o prefixo do namespace do nodo, ou null, caso não esteja especificado.

String **getTextContent**()

Retorna o conteúdo texto deste nodo e de seus descendentes. (Não implementado pelo Java 1.4.2)

boolean **hasAttributes**()

Retorna true se o nodo possui atributos.

boolean **hasChildNodes**()

Retorna true se o nodo possui filhos (lembre-se que texto é considerado um filho do elemento ao qual ele pertence).

Interface Node – Características dos Nodos

boolean **isDefaultNamespace**(String namespaceURI)
Checa se a URI passada como parâmetro é o namespace default.

boolean **isEqualNode**(Node arg)
Testa se dois nodos são iguais.

boolean **isSameNode**(Node other)
Testa se dois nodos são os mesmos.

String **lookupNamespaceURI**(String prefix)
Recupera o URI do namespace associado a um prefixo, iniciando no nodo atual.

String **lookupPrefix**(String namespaceURI)
Recupera o prefixo associado a um dado namespace, iniciando no nodo atual.

void **normalize**()
Reorganiza o conteúdo texto do nodo (e de sua subárvore) de modo que apenas estrutura (isto é, elementos, comentários, etc.) separam nodos Texto. Isso implica que ao final da execução deste método, não existem dois nodos texto adjacentes, e nem nodos texto vazios.

Interface Node – Características dos Nodos

void **setNodeValue**(String nodeValue)
Seta o valor do nodo, dependendo do seu tipo.

void **setPrefix**(String prefix)
Seta o prefixo do namespace do nodo.

void **setTextContent**(String textContent)
Seta o conteúdo texto deste nodo.

Object **setUserData**(String key, Object data, UserDataHandler handler)
Associa um objeto a uma chave neste nodo.

String **getBaseURI**()
Retorna a URI base deste nodo, ou null caso não seja possível obtê-la.

Método getNodeType()

- Um nodo pode ser qualquer objeto XML
  - é preciso determinar o que o nodo representa antes de processá-lo
- O método getNodeType() é utilizado para determinar o tipo de nodo
  - um valor de 1 a 12 é retornado

Método getNodeType()

Valores retornados pelo método getNodeType():

ELEMENT_NODE = 1
ATTRIBUTE_NODE = 2
TEXT_NODE = 3
CDATA_SECTIOM_NODE = 4
ENTITY_REFERENCE_NODE = 5
ENTITY_NODE = 6
PROCESSING_INSTRUCTION_NODE = 7
COMMENT_NODE = 8
DOCUMENT_NODE = 9
DOCUMENT_TYPE_NODE = 10
DOCUMENT_FRAGMENT_NODE = 11
NOTATION_NODE = 12

Método getNodeType()

```
if (myNode.getNodeType() == Node.ELEMENT_NODE)
{
  // processar elemento
}
```

Interface Node – Características dos Nodos

- O método getNodeName() retorna o nome do nodo
- O método getNodeValue() retorna o valor do nodo
- O valor de um nodo pode ser atualizado utilizando o método setNodeValue()
  - um string é passado como parâmetro

Método hasChildNodes()

- O método hasChildNodes() determina se um nodo possui filhos

```
if ( myNode.hasChildNodes() )
{
  // processar os filhos de myNode
}
```

Exercício 1

- Modifique a classe ProcessaDocumentoDOM (em exemplo1), e imprima, para cada nodo:

  nome do nodo => valor do nodo => se tem filhos

Nodos #text

- Como mencionado anteriormente, os nodos #text gerados pelos caracteres de fim de linha (ENTER) entre as tags não são gerados quando existe uma DTD associada ao documento

- No entanto, só isso não basta: é necessário chamar o método **setIgnoringElementContentWhitespace**

- Note que este método não surte efeito quando o documento não possui DTD associada (pois o parser neste caso não tem como saber quais espaços em branco podem ser ignorados)

Exemplo

```
try
  {
  //Instancia o parser
    DocumentBuilderFactory b =
    DocumentBuilderFactory.newInstance();
      b.setIgnoringElementContentWhitespace(true);
    DocumentBuilder builder = b.newDocumentBuilder();



    //Faz o parsing do documento
    Document myDoc = builder.parse("meuDoc.xml");
}
catch( Exception e ) {...}
```

Exercício 2

- Repita o exercício anterior, agora usando o método **setIgnoringElementContentWhitespace**
- Compare os resultados gerados.

Método getAttributes()

- Os atributos de um nodo podem ser acessados através do método getAttributes()
  - retorna um objeto de tipo NamedNodeMap

```
NamedNodeMap myNodeMap = myNode.getAttributes();
```

- Na prática, só nodos de tipo elemento podem ter atributos
  - a interface Element possui métodos alternativos para processar atributos

Interface Node – Navegação nos Nodos

Node **getFirstChild**()
Retorna o primeiro filho do nodo.

Node **getLastChild**()
Retorna o último filho do nodo.

Node **getNextSibling**()
Retorna o nodo imediatamente após o nodo atual.

Node **getPreviousSibling**()
Retorna o nodo imediatamente anterior ao nodo atual.

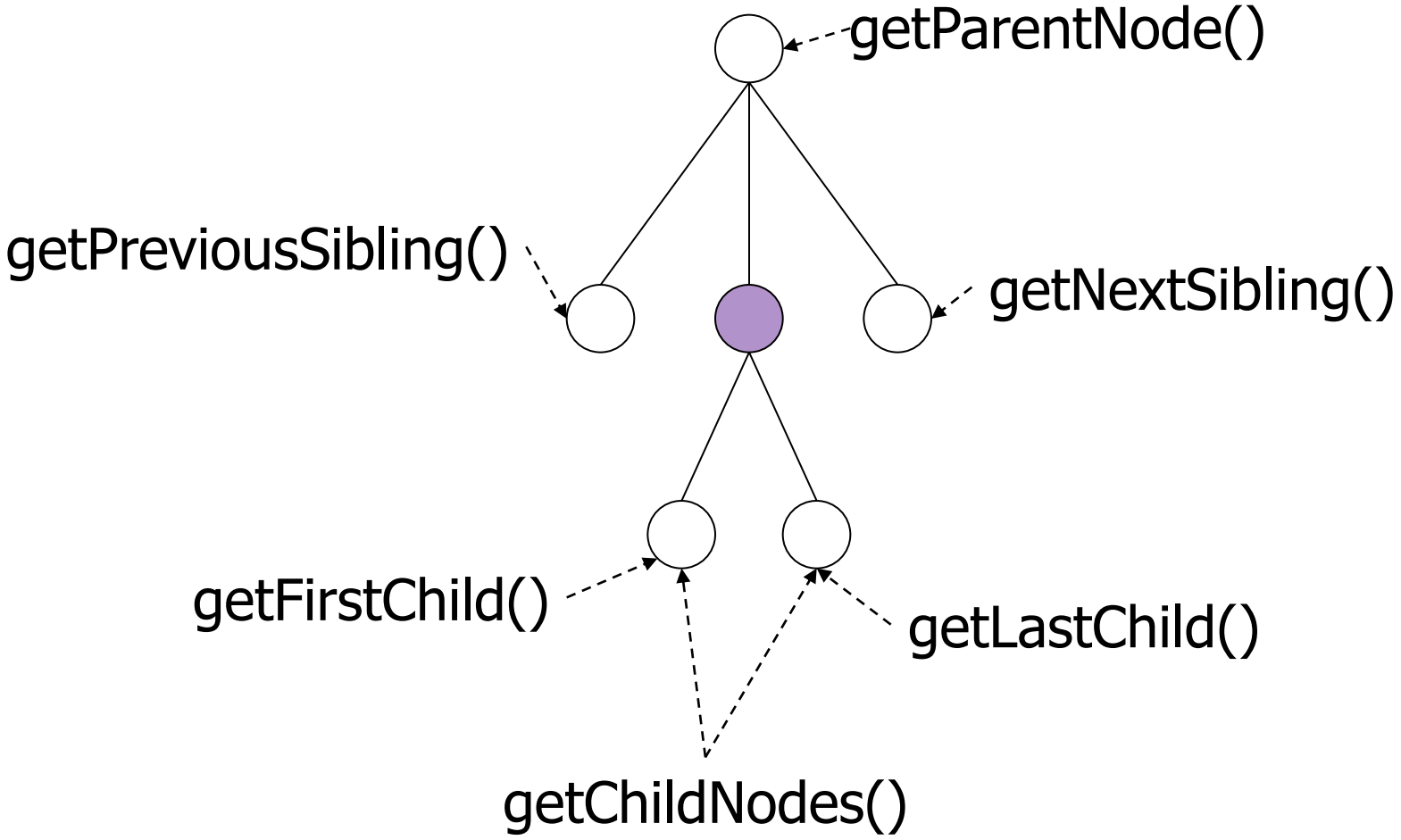
Node **getParentNode**()
Retorna o pai do nodo atual.

NodeList **getChildNodes**()
Retorna os filhos do do nodo atual.

Interface Node – Navegação nos Nodos

- Utilizando os métodos getFirstChild() e getNextSibling() é possível percorrer a árvore completa
- O restante dos métodos provêem funcionalidades adicionais. . .
  - getFirstChild(): voltar ao primeiro filho
  - getParentNode(): subir na hierarquia
  - getPreviousSibling(): voltar na lista de irmãos
  - getChildNodes(): iterar nos filhos

Interface Node – Navegação nos Nodos



Exemplo

- Se o nodo possui filhos, as referências ao primeiro e ao segundo filho são obtidas. A segunda referência será nula se myNode possui só um filho.

```
if ( myNode.hasChildNodes() )
{
  Node firstChild = myNode.getFirstChild();
  Node secondChild = firstChild.getNextSibling();
}
```

Interface Node – Manipulação de Nodos

- Voltaremos à interface Node mais tarde, para estudar os métodos relativos a manipulação de nodos
- Mas antes...

Interfaces DOM

- Interface Node
  - representa o ponto de partida no esquema DOM
- Interface NodeList
  - Fornece uma abstração de uma coleção ordenada de nodos
- Interface NamedNodeMap
  - Representa coleções de nodos que podem ser acessados pelo nome (basicamente, atributos)
- O restante das interfaces (sub-interfaces de Node) provêem funcionalidades específicas ao tipo de objeto que representam

Interface NodeList

- Lista de nodos ordenada
- Utilizada para percorrer os elementos de uma lista de nodos
  - Manipula os itens da lista
  - Os itens no NodeList são acessíveis via um índice, iniciando em 0
- getElementsByTagName() devolve um objeto deste tipo

Interface NodeList

- Apenas dois métodos:

int **getLength**()
Retorna o número de nodos na lista.

Node **item**(int index)
Retorna o i-ésimo item na lista.

Exemplo

```
NodeList nl = myDoc.getElementsByTagName("*");
for (int i = 0; i < nl.getLength(); i++)
{
  Node item = nl.item(i);
  // processar...
}
```

Exemplo 2

- Classe para processar pedidos
- ProcessaPedidoDOM.java (em exemplo2)

Exercício 3

- Modificar a classe ProcessaPedidoDOM.java (em exercicio3) para exibir os itens do pedido, e o valor de cada item
- Resultado esperado:

```
**********************************
--> Cliente: ABC
--->Produto: caneta azul
---->quantidade: 100
---->preco: 2
--->Produto: papel
---->quantidade: 100
---->preco: 8
--> Valor Total: 1000
**********************************
```

Interfaces DOM

- Interface Node
  - representa o ponto de partida no esquema DOM
- Interface NodeList
  - Fornece uma abstração de uma coleção ordenada de nodos
- Interface NamedNodeMap
  - Representa coleções de nodos que podem ser acessados pelo nome (basicamente, atributos)
- O restante das interfaces (sub-interfaces de Node) provêem funcionalidades específicas ao tipo de objeto que representam

Interface NamedNodeMap

- Mapa de nodos nomeados
- Nodos sem ordenação
  - Os nodos podem ser acessados por um índice, mas não existe ordem
- Utilizada para manipulação de atributos de um nodo

```
NamedNodeMap nnm = myElement.getAttributes();
...
Node atributoId = nnm.getNamedItem("id");
```

Interface NamedNodeMap

int **getLength**()
Retorna o número de nodos no “mapa”.

Node **getNamedItem**(String name)
Recupera um nodo especificado por um nome

Node **getNamedItemNS**(String namespaceURI, String localName)
Recupera o nodo especificado por um nome e um namespace.

Node **item**(int index)
Retorna o i-ésimo item do “mapa”.

Node **removeNamedItem**(String name)
Remove o nodo que tem o nome especificado no parâmetro.

Node **removeNamedItemNS**(String namespaceURI, String localName)
Remove o nodo que tem nome e namespace igual aos passados como parâmetro.

Node **setNamedItem**(Node arg)
Adiciona um nodo chamado arg.

Node **setNamedItemNS**(Node arg)
Adiciona um nodo usando o namespace e nome passados por parâmetro.

Exercício 4

- Modificar a classe ProcessaPedidoDOM (do exercício anterior) para exibir o número do pedido.
- Resultado esperado:

```
**********************************
--> Pedido numero: 1000
--> Cliente: ABC
--->Produto: caneta azul
---->quantidade: 100
---->preco: 2
--->Produto: papel
---->quantidade: 100
---->preco: 8
--> Valor Total: 1000
**********************************
```

Interfaces DOM

- Interface Node
  - representa o ponto de partida no esquema DOM
- Interface NodeList
  - Fornece uma abstração de uma coleção ordenada de nodos
- Interface NamedNodeMap
  - Representa coleções de nodos que podem ser acessados pelo nome (basicamente, atributos)
- O restante das interfaces (sub-interfaces de Node) provêem funcionalidades específicas ao tipo de objeto que representam

Interface Node – Manipulação de Nodos

- Voltando à interface Node...

- Existem métodos para realizar diferentes operações com nodos, tais como
  - remover
  - adicionar
  - substituir
  - clonar nodos

Interface Node – Manipulação de Nodos

Node **appendChild**(Node newChild)
Adiciona o nodo **newChild** no final da lista de filhos do nodo atual.

Node **cloneNode**(boolean deep)
Returna uma cópia do nodo. O parâmetro indica se os descendentes devem ou não ser clonados também.

Node **insertBefore**(Node newChild, Node refChild)
Insere **newChild** antes do nodo **refChild**. O nodo **refChild** já deve existir no documento.

Node **removeChild**(Node oldChild)
Remove o nodo indicado por **oldChild** e retorna uma referência para ele.

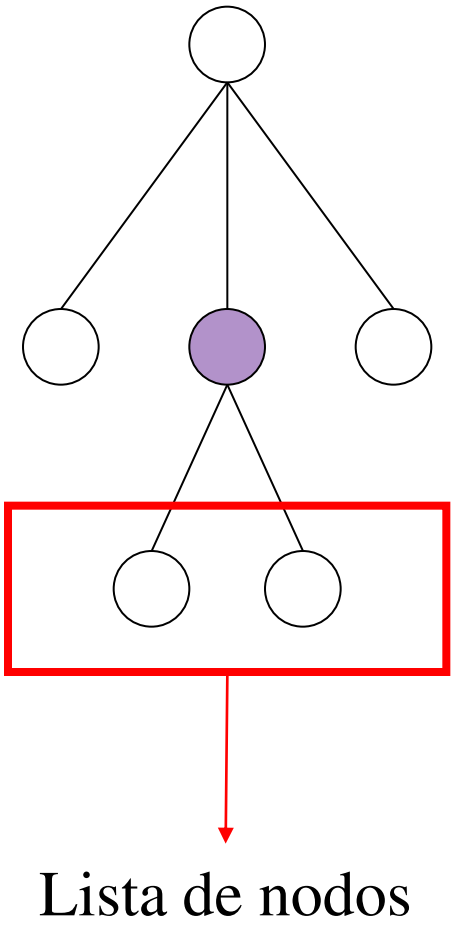
Node **replaceChild**(Node newChild, Node oldChild)
Substitui o nodo **oldChild** por **newChild**, e retorna uma referência para **oldChild**.

Remoção de Nodos - removeChild()

- O método removeChild
  - desassocia um nodo de sua localização original
  - O nodo ainda existe, ou seja, seu conteúdo não é excluído
  - O método retorna uma referência ao nodo removido
- Um nodo desassociado pode voltar a ser associado à arvore DOM

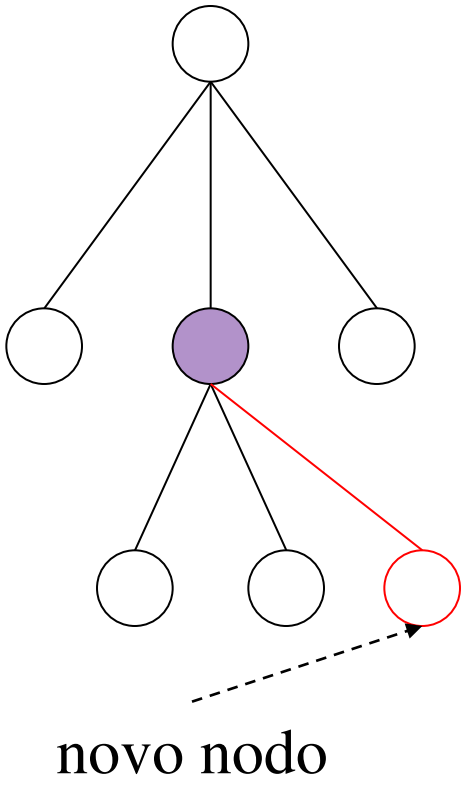
Inclusão de Nodos

- Um nodo pode ser adicionado à lista de nodos de um nodo existente
- Os novos nodos:
  - podem ser nodos já existentes (desassociados do objeto DOM anteriormente)
  - podem ser criados utilizando métodos na interface Node



Inclusão de Nodos – appendChild()

- O método appendChild
  - associa um nodo à lista de nodos de um nodo existente
  - retorna uma referência ao nodo inserido
  - o novo nodo é inserido ao final da lista de nodos



Exemplo

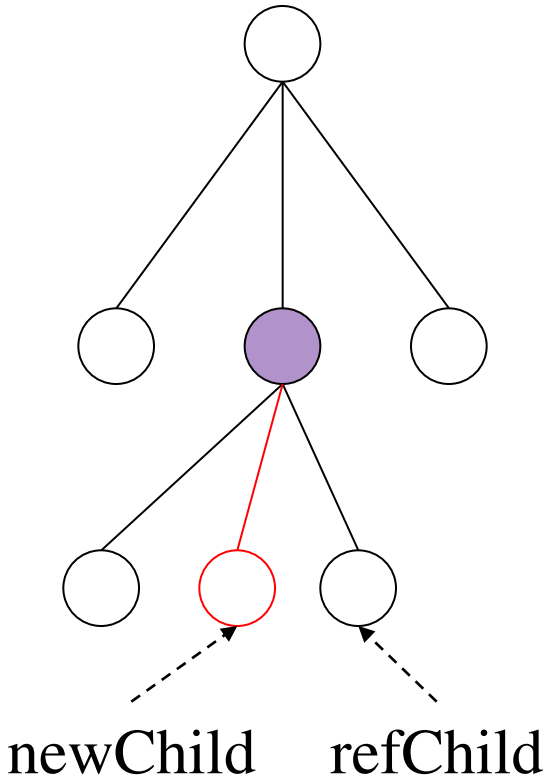
- Utiliza o método createElement para criar um novo nodo, associa o novo nodo como filho do nodo theParent e o elimina

```
Node addedNode = theParent.appendChild(doc.createElement("para"));
theParent.removeChild(addedNode);
```

(createElement() – método da Interface Document)

Inclusão de Nodos – insertBefore()

- O método insertBefore
  - Insere um nodo em uma posição específica
  - Uma referência a um nodo da lista é passada como parâmetro
  - Após a operação, newChild precede refChild



Exemplo

- Adiciona um nodo em uma posição específica
  - Um elemento para é criado (createElement())
  - Este elemento é adicionado à lista de nodos
  - A seguir um elemento note é inserido antes do para

```
Node paraNode =
    theParent.appendChild(doc.createElement("para"));
theParent.insertBefore(doc.createElement("note"), paraNode);
```

Substituição de Nodos – replaceChild()

- O método replaceChild
  - substitui um nodo existente na lista por um novo nodo
  - provê uma forma simples e direta de substituição de nodos.
- Esta funcionalidade pode ser executada através de uma combinação dos métodos mencionados anteriormente

Clonagem de Nodos

- O método cloneNode
  - serve para clonar um nodo
  - retorna um novo nodo que possui mesmo tipo, nome e valor que o nodo clonado
  - utilidade na edição de texto (copiar e colar)
- Formas de clonagem, de acordo com o parâmetro passado
  - true: os filhos e os filhos dos filhos, ..., também devem ser clonados (*deep cloning*)
  - false: nenhum filho deve ser clonado (*shallow cloning*)

Exemplo ECHO
(código completo está em exemploImprime)

```
static void imprime(Node myDoc)
  {
      try {
       // configura o transformador
      TransformerFactory transfac = TransformerFactory.newInstance();
      Transformer trans = transfac.newTransformer();
      trans.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION, "yes");
      trans.setOutputProperty(OutputKeys.INDENT, "yes");
       // cria uma string a partir da árvore XML
      StringWriter sw = new StringWriter();
      StreamResult result = new StreamResult(sw);
      DOMSource source = new DOMSource(myDoc);
      trans.transform(source, result);
      String xmlString = sw.toString();
      // imprime o XML
      System.out.println("Aqui esta o xml:\n\n" + xmlString);
      } catch (Exception e) { System.out.println(e);  }
  }
```

Exercício 5

Modifique a classe ProcessaDocumentoDOM (dentro de exercicio5) da seguinte forma:

- Inserir um novo elemento chamado “NovoElemento” no documento teste1.xml, depois do elemento primeiro_filho
- Imprima a árvore DOM antes e depois da inserção
- Clonar o elemento quarto_neto e inserir o clone antes do elemento segundo_filho
- Excluir o elemento segundo_neto e inseri-lo como filho do elemento terceiro_neto

Exercício 6

- Suponha que a empresa mudou o formato do documento XML que modela o pedido

<produto> → <prod>

<quantidade> → <quant>

  - O que mudaria na implementação?
  - PROBLEMA!!! Como resolver sem mudar a implementação???

Parser com validação

- Para fazer o parser com validação, algumas coisas devem ser feitas:
  - Setar a validação:

```
DocumentBuilderFactory b =
        DocumentBuilderFactory.newInstance();
b.setValidating(true);
```

  - Construir um handler para os erros (SAX!!!)

Parser com validação

```
// Get an instance of the parser
DocumentBuilderFactory b = DocumentBuilderFactory.newInstance();

//Seta validação para true
b.setValidating(true);

DocumentBuilder builder = b.newDocumentBuilder();

//Pega uma instância do manipulador de erros
MeuManipuladorDeErros handler = new MeuManipuladorDeErros();

//Seta o ErrorHandler
builder.setErrorHandler(handler);

//Parse the document
Document myDoc = builder.parse(argv[0]);
```

Parser com validação

- Abra o arquivo MeuManipuladorDeErros (dentro de exemplo3) para ver como foi implementado
- Execute a classe Processa para o pedido4.xml
- Insira erros relativos à DTD no arquivo pedido4.xml e execute a classe novamente

Exercício 7

- Modificar a classe ProcessaPedidoDOM.java (do exercício 4) para que ela gere a nota fiscal do pedido:
- Processe com o documento pedido4.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

- Modifique a classe ProcessaSQLDOM (dentro de exercicio8) 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!!