Systems for Design and Implementation

2015-2016

Course 10

Contents

- WebSockets
- XPath
- XSLT
- API for C# and Java

WebSockets

Limitations of HTTP:

- Stateless:
  - The browser opens up a socket on port to 80.
  - It sends an HTTP header request to the server.
  - The server application decides what to do with the request, fetches data, generates HTML and sends it back to the server.
  - The web server adds the appropriate HTTP headers to the body, sends it back to the browser and closes the connection.
- The websites need to maintain information about users (e.g. cookies).
  - The information (cookies) is passed back and fourth for every request made. The additional information carries overhead and is open to security vulnerabilities (if not properly secured).
- All communication is client initiated and each stateless request/response is isolated.

WebSockets

- WebSockets are a bi-directional, full-duplex, persistent connection from a web browser to a server.
- Once a WebSocket connection is established the connection stays open until the client or server decides to close this connection.
- With this open connection, the client or server can send a message at any given time to the other.
- This makes web programming entirely event driven, not (just) user initiated. It is stateful.
- A single running server application is aware of all connections, allowing communication with any number of open connections at any given time.

WebSocket Protocol

- In 2011, the IETF standardized the WebSocket protocol as RFC 6455.
- Since then, the majority of the Web browsers are implementing client APIs that support the WebSocket protocol.
- A number of libraries (Java, .NET, Ruby, Objective C) have been developed that implement the WebSocket protocol.

- Browsers: natively in Chrome, Firefox, Opera and Safari (including mobile Safari), Internet Explorer.
- Any browser that does not support WebSockets can use a Flash polyfill.

WebSocket Protocol Handshake

To establish a WebSocket connection, the client sends a HTTP WebSocket handshake request:


```
GET /echo HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Version: 13
Origin: http://example.com
```

WebSocket Protocol Handshake

- If the server accepts the request to upgrade the application-layer protocol, it returns a `HTTP 101 Switching Protocols` response:


```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: echo
```

- After the server returns its 101 response, the application-layer protocol switches from HTTP to WebSockets which uses the previously established TCP connection. Messages can now be sent or received by either endpoint at any time

WebSocket - URI

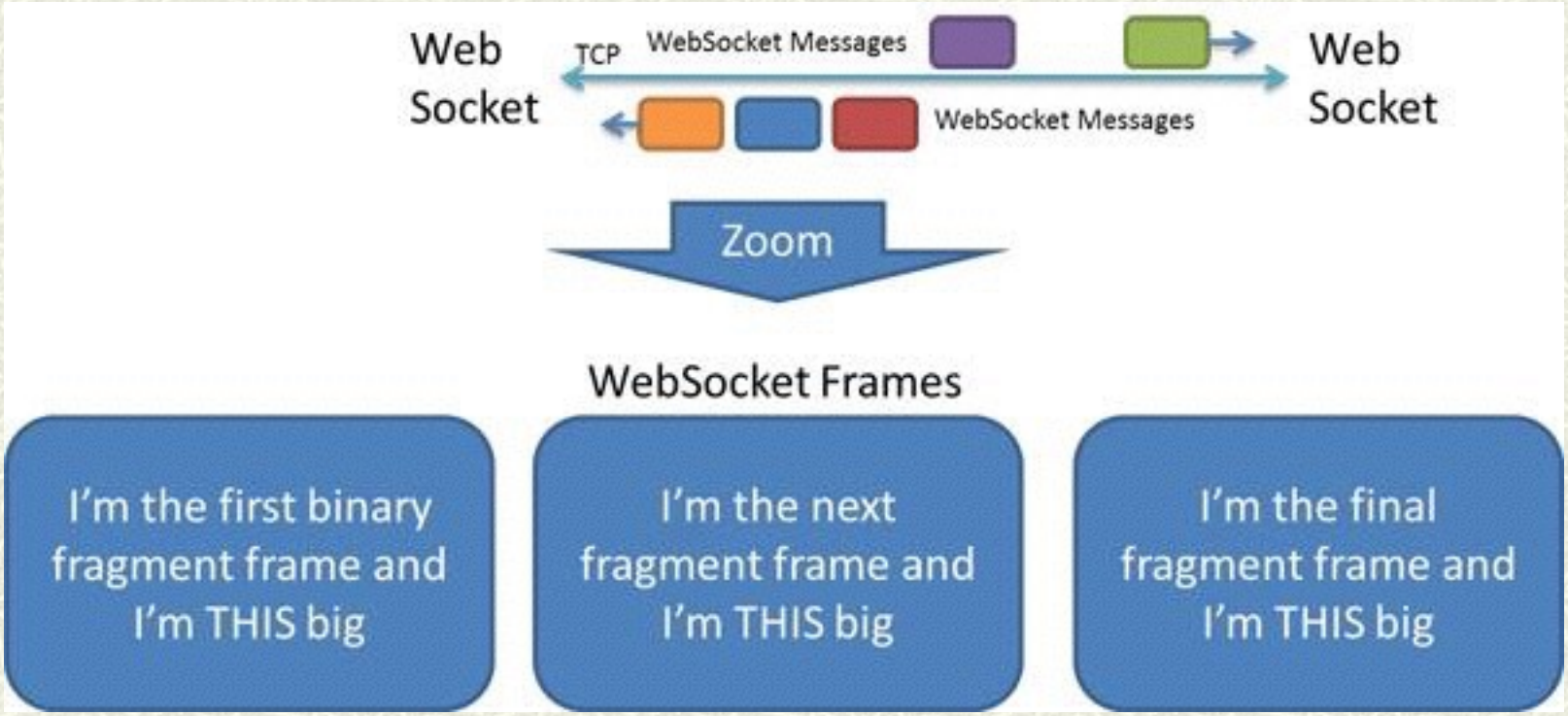
The WebSocket protocol defines two new URI schemes which are similar to the HTTP schemes:

- `"ws:" "//" host [ ":" port ] path [ "?" query ]` is modeled on the “http:” scheme.
  - Its default port is 80.
  - It is used for unsecure (unencrypted) connections.
- `"wss:" "//" host [ ":" port ] path [ "?" query ]` is modeled on the “https:” scheme.
  - Its default port is 443.
  - It is used for secure connections tunneled over Transport Layer Security (TLS).

```
ws://example.com
ws://example.com:8080/echo
```

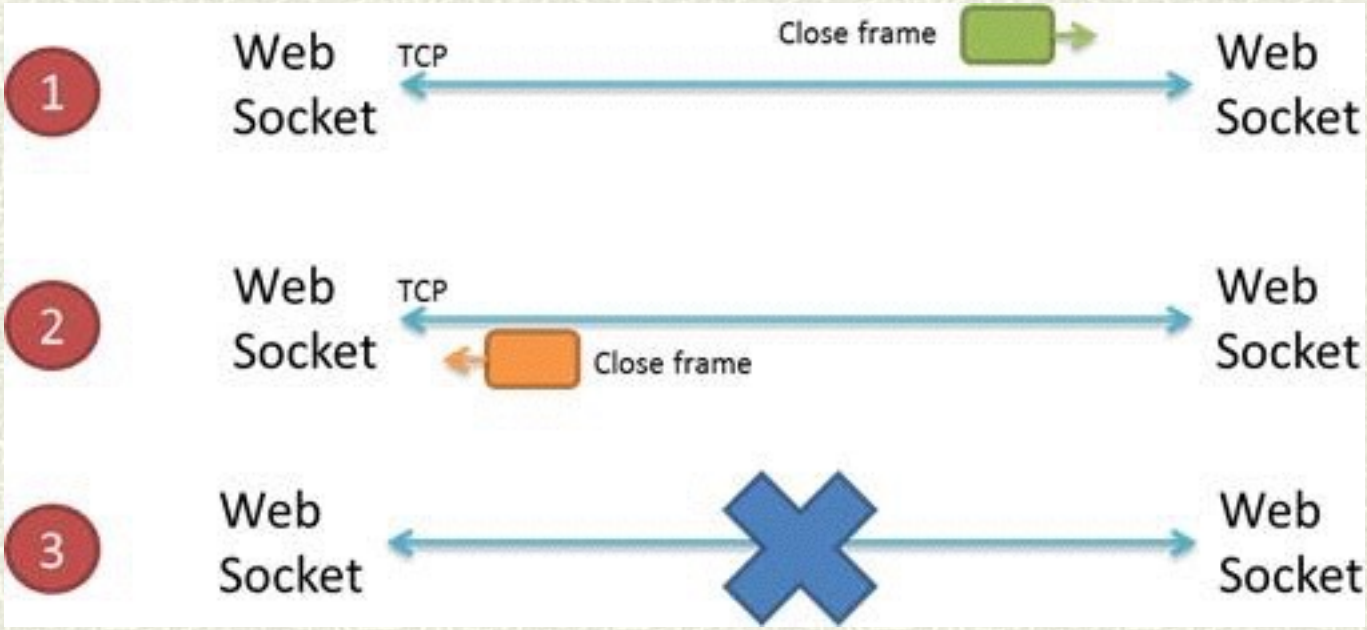
WebSocket - Messages

- After a successful handshake, the application and the Websocket server may exchange WebSocket messages.
- A message is composed as a sequence of one or more message fragments or data "frames." Each frame includes information such as:
  - Frame length
  - Type of message (binary or text) in the first frame in the message
  - A flag (FIN) indicating whether this is the last frame in the message



Closing a WebSocket

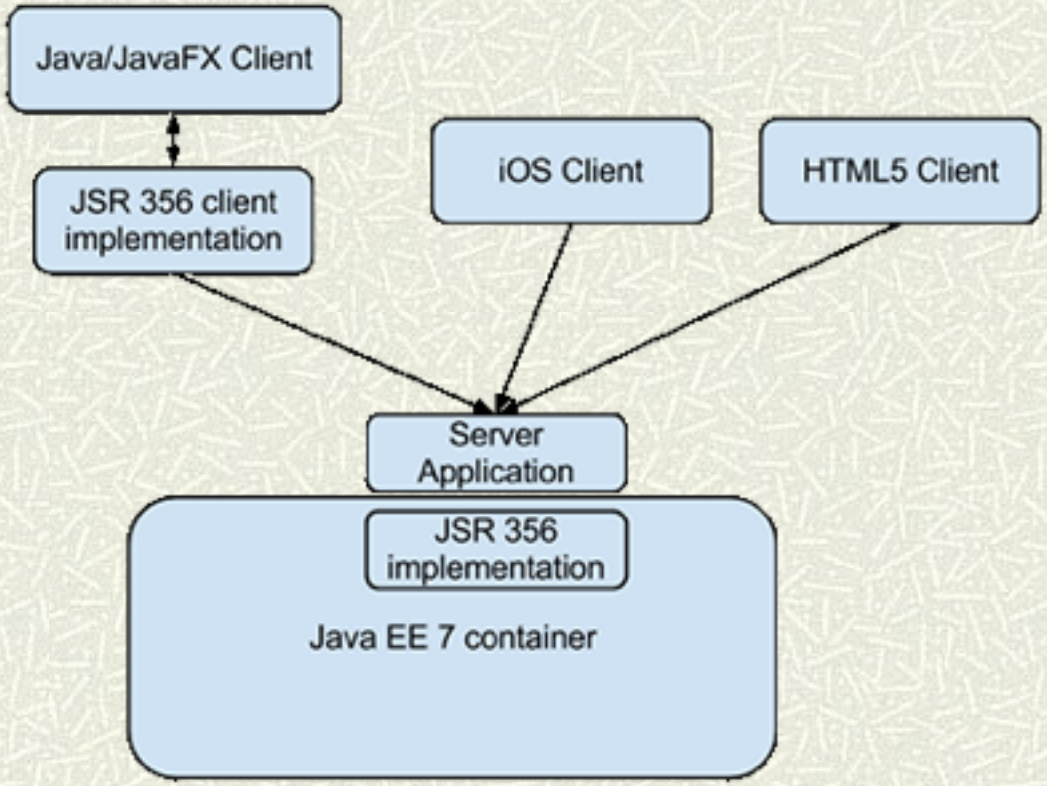
- Either endpoint (the application or the server) can initiate a closing handshake.
- A special kind of frame – a *close* frame – is sent to the other endpoint. The close frame may contain an optional status code and reason for closing.
- The protocol defines a set of appropriate values for the status code.
- The sender of the close frame must not send further application data after the close frame.
- When the other endpoint receives the close frame, it responds with its own close frame in response. It may send pending messages prior to responding with the close frame.



WebSockets API - Java

- JSR 356, Java API for WebSocket, specifies the API that Java developers can use when they want to integrate WebSockets into their applications—both on the server side as well as on the Java client side.
- Every implementation of the WebSocket protocol that claims to be compliant with JSR 356 must implement this API.
- Developers can write their WebSocket-based applications independent of the underlying WebSocket implementation.
- JSR 356 is a part of the Java EE 7 standard.
- All Java EE 7–compliant application servers will have an implementation of the WebSocket protocol that adheres to the JSR 356 standard.

WebSockets API - Java



- A JavaFX client can rely on any JSR 356–compliant client implementation for handling the WebSocket-specific protocol issues. Other clients (for example, an iOS client and an HTML5 client) can use other (non-Java) implementations that are compliant with RFC 6455 in order to communicate with the server application.

WebSockets API - Java

- JSR 356 leverages annotations and injection.
- Two different programming models are supported:
  - *Annotation-driven.* Using annotated POJOs, developers can interact with the WebSocket lifecycle events.
  - *Interface-driven.* Developers can implement the Endpoint interface and the methods that interact with the lifecycle events.
- *Lifecycle Events*: the typical lifecycle events of a WebSocket interaction:
  - One peer (a client) initiates the connection by sending an HTTP handshake request.
  - The other peer (the server) replies with a handshake response.
  - The connection is established. From now on, the connection is completely symmetrical.
  - Both peers send and receive messages.
  - One of the peers closes the connection.
- Most of the WebSocket lifecycle events can be mapped to Java methods, both in the annotation-driven and interface-driven approaches.

Java WebSockets API - Annotations

- An endpoint that is accepting incoming WebSocket requests can be a POJO annotated with the `@ServerEndpoint` annotation.
- This annotation tells the container that the given class should be considered to be a WebSocket endpoint. The required value element specifies the path of the WebSocket endpoint.

```
@ServerEndpoint("/hello")
public class MyEndpoint { }

@ServerEndpoint("/hello/{userid}")
public class MyEndpoint { }
```

where the value of `{userid}` can be obtained in lifecycle method calls using the `@PathParam` annotation.

Java WebSockets API - Annotations

- An endpoint that should initiate a WebSocket connection can be a POJO annotated with the `@ClientEndpoint` annotation.
- `ClientEndpoint` does not accept a path value element, because it is not listening to incoming requests.

```
@ClientEndpoint
public class MyClientEndpoint {}
```

- Initiating a WebSocket connection in Java leveraging the annotation-driven POJO approach can be done as follows:

```
javax.websocket.WebSocketContainer container =
javax.websocket.ContainerProvider.getWebSocketContainer();

container.conntectToServer(MyClientEndpoint.class,
new URI("ws://localhost:8080/tictactoeserver/endpoint"));
```

- Classes annotated with `@ServerEndpoint` or `@ClientEndpoint` will be called annotated endpoints.

Java WebSockets API - Annotations

- After a WebSocket connection has been established, a `Session` is created and the method annotated with `@OnOpen` on the annotated endpoint will be called.
- This method can contain a number of parameters:
  - A `javax.websocket.Session` parameter, specifying the created Session
  - An `EndpointConfig` instance containing information about the endpoint configuration
  - Zero or more string parameters annotated with `@PathParam`, referring to path parameters on the endpoint path

```
@OnOpen
public void myOnOpen (Session session) {
   System.out.println ("WebSocket opened: "+session.getId());
}
```

Java WebSockets API - Annotations

- A `Session` instance is valid as long as the WebSocket is not closed.
- The `Session` class contains methods that allow developers to obtain more information about the connection.
- It also contains a hook to application-specific data, by means of the `getUserProperties()` method returning a `Map<String, Object>`.
- This allows developers to populate Session instances with session- and application-specific information that should be shared among method invocations.

Java WebSockets API - Annotations

- When the WebSocket endpoint receives a message, the method annotated with `@OnMessage` will be called.
- A method annotated with `@OnMessage` can contain the following parameters:
  - The `javax.websocket.Session` parameter.
  - Zero or more string parameters annotated with `@PathParam`, referring to path parameters on the endpoint path.
  - The message itself (text, binary, or pong message).
- For each different type of message, one `@OnMessage` annotated method is allowed. The allowed parameters for specifying the message content in the annotated methods are dependent on the type of the message.

```
@OnMessage
public void myOnMessage (String txt) {
   System.out.println ("WebSocket received message: "+txt);
}
```

Java WebSockets API - Annotations

- If the return type of the method annotated with @OnMessage is not void, the WebSocket implementation will send the return value to the other peer.

```
@OnMessage
public String myOnMessage (String txt) {
   return txt.toUpperCase();
}
```

- Alternative way of sending messages over a WebSocket connection:

```
RemoteEndpoint.Basic other = session.getBasicRemote();
other.sendText ("Hello, world");
```

- The `getBasicRemote()` method on the `Session` instance returns a representation of the other part of the WebSocket, the `RemoteEndpoint`. That `RemoteEndpoint` instance can be used for sending text or other types of messages.
- The `Session` object can be obtained from the lifecycle callback methods (e.g., the method annotated with `@OnOpen`)

Java WebSockets API - Annotations

- When the WebSocket connection is closing, the method annotated with `@OnClose` is called. The method can take the following parameters:
  - The `javax.websocket.Session` parameter. This parameter cannot be used once the WebSocket is really closed, which happens after the `@OnClose` annotated method returns.
  - A `javax.websocket.CloseReason` parameter describing the reason for closing the WebSocket (e.g., normal closure, protocol error, overloaded service, etc.).
  - Zero or more string parameters annotated with `@PathParam`, referring to path parameters on the endpoint path.

```
@OnClose
public void myOnClose (CloseReason reason) {
        System.out.prinlnt ("Closing a WebSocket due to
    "+reason.getReasonPhrase());
}
```

- In case an error is received, the method annotated with `@OnError` will be called.

Java WebSockets API - Messages

- Any Java object can be sent or received as a WebSocket message.
- Three different types of messages:
  - *Text-based* messages
  - *Binary* messages
  - *Pong* messages, which are about the WebSocket connection itself.
- Allowed message parameters for text messages:
  - `String` to receive the whole message
  - Java primitive or class equivalent to receive the whole message converted to that type
  - `String` and boolean pair to receive the message in parts
  - `Reader` to receive the whole message as a blocking stream
  - any object parameter for which the endpoint has a text decoder (`Decoder.Text` or `Decoder.TextStream`).

Java WebSockets API - Messages

- Allowed message parameters for binary messages:
  - `byte[]` or `ByteBuffer` to receive the whole message
  - `byte[]` and `boolean` pair, or `ByteBuffer` and `boolean` pair to receive the message in parts
  - `InputStream` to receive the whole message as a blocking stream
  - any object parameter for which the endpoint has a binary decoder (`Decoder.Binary` or `Decoder.BinaryStream`).
- Allowed message parameters for pong messages:
  - `PongMessage` for handling pong messages.
- Any Java object can be encoded into a text-based or binary message using an encoder.
- The text-based or binary message is transmitted to the other peer, where it can be decoded into a Java object again.
- XML or JSON are often used for the transmission of WebSocket messages, and the encoding/decoding then comes down to marshaling a Java object into XML or JSON and back.

Java WebSockets API - Messages

- An encoder is defined as an implementation of the `javax.websocket.Encoder` interface, and a decoder is an implementation of the `javax.websocket.Decoder` interface.
- The endpoint instances need to know what the possible encoders and decoders are.
- Using the annotation-driven approach, a list of encoders and decoders is passed via the encoder and decoder elements in the `@ClientEndpoint` and `@ServerEndpoint` annotations.

```
@ServerEndpoint(value="/endpoint", encoders = MessageEncoder.class,
decoders= MessageDecoder.class)
public class MyEndpoint {
...
}
```

Java WebSockets API - Messages

```
class MessageEncoder implements Encoder.Text<MyJavaObject> {
   @override
   public String encode(MyJavaObject obj) throws EncodingException {
      ...
   }
}

class MessageDecoder implements Decoder.Text<MyJavaObject> {
   @override
   public MyJavaObject decode (String src) throws DecodeException {
      ...
   }

   @override
   public boolean willDecode (String src) {
          // return true if we want to decode this String into a
//MyJavaObject instance
   }
}
```

Java WebSockets API - Messages

- The `Encoder` interface has a number of subinterfaces:
  - `Encoder.Text` for converting Java objects into text messages
  - `Encoder.TextStream` for adding Java objects to a character stream
  - `Encoder.Binary` for converting Java objects into binary messages
  - `Encoder.BinaryStream` for adding Java objects to a binary stream

- The `Decoder` interface has four subinterfaces:
  - `Decoder.Text` for converting a text message into a Java object
  - `Decoder.TextStream` for reading a Java object from a character stream
  - `Decoder.Binary` for converting a binary message into a Java object
  - `Decoder.BinaryStream` for reading a Java object from a binary stream

Java WebSockets API - Interface

- Using the interface-driven approach, a developer extends `javax.websocket.Endpoint` and overrides the `onOpen`, `onClose`, and `onError` methods:

```
public class myOwnEndpoint extends javax.websocket.Endpoint {
   public void onOpen(Session session, EndpointConfig config) {...}
   public void onClose(Session session, CloseReason closeReason)
{...}
   public void onError (Session session, Throwable throwable) {...}
}
```

In order to intercept messages, a `javax.websocket.MessageHandler` needs to be registered in the `onOpen` implementation:

```
public void onOpen (Session session, EndpointConfig config) {
   session.addMessageHandler (new MessageHandler() {...});
}
```

Java WebSockets API - Interface

- `MessageHandler` is an interface with two subinterfaces:
  - `MessageHandler.Partial`. It should be used when the developer wants to be notified about partial deliveries of messages.
  - `MessageHandler.Whole`. It should be used for notification about the arrival of a complete message.

```
public void onOpen (Session session, EndpointConfig config) {
   final RemoteEndpoint.Basic remote = session.getBasicRemote();
   session.addMessageHandler (new MessageHandler.Whole<String>() {
      public void onMessage(String text) {
                  try {
                      remote.sendString(text.toUpperCase());
                  } catch (IOException ioe) {
                      // handle send failure here
                  }
              }
   });
}
```

WebSockets

- Examples:
  - Transformer
  - Whiteboard

- References
  - RFC 6455

https://tools.ietf.org/html/rfc6455

  - Johan Vos, *JSR 356, Java API for WebSocket*,

http://www.oracle.com/technetwork/articles/java/jsr356-1937161.html

  - Brian Raymor, *WebSockets: Stable and Ready for Developers*,

https://msdn.microsoft.com/en-us/hh969243.aspx

XPath

- XPath, the XML Path Language, is a query language for selecting nodes from an XML document.
- XPath may also be used to compute values (e.g., strings, numbers, or Boolean values) from the content of an XML document.
- XPath was defined by the World Wide Web Consortium (W3C).
- The XPath language is based on a tree representation of the XML document, and provides the ability to navigate around the tree, selecting nodes by a variety of criteria.

XPath

- XML documents are treated as trees of nodes. The root of the tree is called the document node (or root node).
- In XPath, there are seven kinds of nodes: element, attribute, text, namespace, processing-instruction, comment, and document (root) nodes.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore> <!-- document node -->
  <book>
     <title lang="ro">Poezii</title>
     <author>Mihai Eminescu</author>  <!- element node -->
     <year>2005</year>
     <price>15</price>
   </book>
</bookstore>

<!- lang="ro" attribute node -->
```

XPath

- Atomic values are nodes with no children or parent.

Eg. Poezii, Mihai Eminescu, "ro"

- Items are atomic values or nodes.
- Relationships between nodes:
  - **Parent:** Each element and attribute has one parent (the **book** element is the parent of the **title**, **author, year,** and **price**).
  - **Children:** Element nodes may have zero, one or more children (the **title**, **author**, **year**, and **price** elements are all children of the **book** element).
  - **Siblings:** Nodes that have the same parent (the **title**, **author**, **year**, and **price** elements are all siblings).
  - **Ancestors:** A node's parent, parent's parent, etc. (the ancestors of the **title** element are the **book** element and the **bookstore** element).
  - **Descendants:** A node's children, children's children, etc. (descendants of the **bookstore** element are the **book**, **title**, **author**, **year**, and **price** elements).

XPath

- XPath uses path expressions to select nodes or node-sets in an XML document. The node is selected by following a path or steps.

Expression	Description
nodename	Selects all child nodes of the named node
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are.
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

XPath

- Selecting nodes examples:
  - `bookstore` Selects all the child nodes of the bookstore element.
  - `/bookstore` Selects the root element bookstore.
  - `bookstore/book` Selects all book elements that are children of bookstore.
  - `//book` Selects all book elements no matter where they are in the document.
  - `bookstore//book` Selects all book elements that are descendant of the `bookstore` element, no matter where they are under the `bookstore` element
  - `//@lang` Selects all attributes that are named `lang`.

Remark:

If the path starts with a slash (/) it always represents an absolute path to an element!

XPath - Predicates

- Predicates are used to find a specific node or a node that contains a specific value.
- Predicates are always embedded in square brackets.

`/bookstore/book[1]` Selects the first `book` element that is the child of the `bookstore` element.

`/bookstore/book[last()]` Selects the last `book` element that is the child of the `bookstore` element.

`/bookstore/book[last()-1]` Selects the last but one `book` element that is the child of the `bookstore` element.

`/bookstore/book[position()<3]` Selects the first two `book` elements that are children of the `bookstore` element.

`//title[@lang]` Selects all the `title` elements that have an attribute named `lang`.

`//title[@lang='eng']` Selects all the `title` elements that have an attribute named `lang` with a value of `'eng'`.

`/bookstore/book[price>35.00]` Selects all the `book` elements of the `bookstore` element that have a `price` element with a value greater than 35.00.

`/bookstore/book[price>35.00]/title` Selects all the `title` elements of the `book` elements of the `bookstore` element that have a `price` element with a value greater than 35.00.

XPath - Wildcards

- XPath wildcards can be used to select unknown XML elements:
  - `*` Matches any element node
  - `@*` Matches any attribute node
  - `node()` Matches any node of any kind
- Examples:

`/bookstore/*`	Selects all the child nodes of the `bookstore` element
`//*`	Selects all elements in the document
`//title[@*]`	Selects all `title` elements which have any attribute

XPath - | operator

- Several paths can be selected by using the | operator in an XPath expression.

- Examples:

  `//book/title | //book/price`

  Selects all the `title` AND `price` elements of all `book` elements

  `//title | //price`

  Selects all the `title` AND `price` elements in the document

  `/bookstore/book/title | //price`

  Selects all the `title` elements of the `book` element of the `bookstore` element AND all the `price` elements in the document

XPath - Axes

An axis defines a node-set relative to the current node:

- **`ancestor`** Selects all ancestors (parent, grandparent, etc.) of the current node.
- **`ancestor-or-self`** Selects all ancestors (parent, grandparent, etc.) of the current node and the current node itself.
- **`attribute`** Selects all attributes of the current node.
- **`child`** Selects all children of the current node.
- **`descendant`** Selects all descendants (children, grandchildren, etc.) of the current node.
- **`descendant-or-self`** Selects all descendants (children, grandchildren, etc.) of the current node and the current node itself.
- **`following`** Selects everything in the document after the closing tag of the current node.
- **`following-sibling`** Selects all siblings after the current node.
- **`namespace`** Selects all namespace nodes of the current node.
- **`parent`** Selects the parent of the current node.
- **`preceding`** Selects everything in the document that is before the start tag of the current node.
- **`preceding-sibling`** Selects all siblings before the current node.
- **`self`** Selects the current node.

XPath – Location Path

- A location path can be absolute or relative.
  - An absolute location path starts with a slash (/) and a relative location path does not. In both cases the location path consists of one or more steps, each separated by a slash:
  - An absolute location path: `/step/step/...`
  - A relative location path: `step/step/...`
- Examples:

`child::book`	Selects all `book` nodes that are children of the current node.
`attribute::lang`	Selects the `lang` attribute of the current node.
`child::*`	Selects all children of the current node.
`attribute::*`	Selects all attributes of the current node.
`child::text()`	Selects all text child nodes of the current node.
`child::node()`	Selects all child nodes of the current node.
`descendant::book`	Selects all `book` descendants of the current node.
`ancestor::book`	Selects all `book` ancestors of the current node.

XPath – Operators

- An XPath expression returns either a node-set, a string, a Boolean, or a number.
- Operators:
  - `|`
  - `+, - , * , div`
  - `= ,  != , <, <=, >, >=`
  - `or, and`
  - `mod`
- Examples:

```
//book | //cd
price=9.80 or price=9.70
price>9.00 and price<9.90
5 mod 2
```

XPath – Examples

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
    <book category="Science">
      <title lang="en">Basic Algebra</title>
      <author>John Smith</author>
      <year>2005</year>
      <price>30.00</price>
    </book>
    <book category="Children">
      <title lang="en">Harry Potter</title>
      <author>J K. Rowling</author>
      <year>2005</year>
      <price>29.99</price>
    </book>
    <book category="Web">
      <title lang="en">Learning XML</title>
      <author>William Jones</author>
      <year>2003</year>
      <price>39.00</price>
    </book>
</bookstore>
```

XPath –Examples

```
/bookstore/book/title  <!-- select all title nodes -->
<bookstore>
    <book category="Science">
      <title lang="en">Basic Algebra</title>
      <author>John Smith</author>
      <year>2005</year>
      <price>30.00</price>
    </book>
    <book category="Children">
      <title lang="en">Harry Potter</title>
      <author>J K. Rowling</author>
      <year>2005</year>
      <price>29.99</price>
    </book>
    <book category="Web">
      <title lang="en">Learning XML</title>
      <author>William Jones</author>
      <year>2003</year>
      <price>39.00</price>
    </book>
</bookstore>
```

XPath – Examples

```
/bookstore/book[1]/title <!- select the title of the first book -->
<bookstore>
    <book category="Science">
      <title lang="en">Basic Algebra</title>
      <author>John Smith</author>
      <year>2005</year>
      <price>30.00</price>
    </book>
    <book category="Children">
      <title lang="en">Harry Potter</title>
      <author>J K. Rowling</author>
      <year>2005</year>
      <price>29.99</price>
    </book>
    <book category="Web">
      <title lang="en">Learning XML</title>
      <author>William Jones</author>
      <year>2003</year>
      <price>39.00</price>
    </book>
</bookstore>
```

XPath – Examples

```
/bookstore/book/price/text() <!- select all prices -->
<bookstore>
    <book category="Science">
      <title lang="en">Basic Algebra</title>
      <author>John Smith</author>
      <year>2005</year>
      <price>30.00</price>
    </book>
    <book category="Children">
      <title lang="en">Harry Potter</title>
      <author>J K. Rowling</author>
      <year>2005</year>
      <price>29.99</price>
    </book>
    <book category="Web">
      <title lang="en">Learning XML</title>
      <author>William Jones</author>
      <year>2003</year>
      <price>39.00</price>
    </book>
</bookstore>
```

XPath – Examples

```
/bookstore/book[price>35]/title <!- Select title nodes with
   price>35-->
<bookstore>
    <book category="Science">
      <title lang="en">Basic Algebra</title>
      <author>John Smith</author>
      <year>2005</year>
      <price>30.00</price>
    </book>
    <book category="Children">
      <title lang="en">Harry Potter</title>
      <author>J K. Rowling</author>
      <year>2005</year>
      <price>29.99</price>
    </book>
    <book category="Web">
      <title lang="en">Learning XML</title>
      <author>William Jones</author>
      <year>2003</year>
      <price>39.00</price>
    </book>
</bookstore>
```

XPath – Standard functions

- `id(...)`: Returns the node with the specified ID.
- `last()`: Returns the index of the last element.
- `position()`: Returns the index position.
- `count(...)`: Returns the count of elements.
- String functions: `concat`, `starts-with`, `contains`, `substring`, `string-length`, `normalize-space`, etc.
- Boolean functions: `not(...)`, `true()`, `false()`
- Numeric functions: `sum`, `floor`, `ceiling`, `round`
- Conversion functions:
  - `string(...)`: Returns the string value of a number, Boolean, or node-set.
  - `boolean(...)`: Returns a Boolean value for a number, string, or node-set (a non-zero number, a nonempty node-set, and a nonempty string are all true).
  - `number(...)`: Returns the numeric value of a Boolean, string, or node-set (true is 1, false is 0, a string containing a number becomes that number, the string-value of a node-set is converted to a number).

XSLT

- Extensible Stylesheet Language Transformations (XSLT) is an XML-based language used for the transformation of XML documents into other XML or "human-readable" documents.
- The original document is not changed; rather, a new document is created based on the content of an existing one.
- The new document may be serialized (output) by the processor in standard XML syntax or in another format, such as HTML or plain text.
- XSLT is most often used to convert data between different XML formats or to convert XML data into HTML or XHTML documents for web pages.

XSLT

- The XSLT processing model involves:
  - one or more XML *source* documents;
  - one or more XSLT *stylesheet* modules;
  - the XSLT template processing engine (the *processor*);
  - one or more *result* documents.
- The XSLT processor usually takes two input documents - an XML source document, and an XSLT stylesheet—and produces an output document.
- The XSLT stylesheet contains the XSLT program text and is itself an XML document. It describes a collection of *template rules* - *instructions* and other directives that guide the processor in producing the output document.
- XSLT relies upon the XPath language for identifying subsets of the source document tree, as well as for performing calculations.
- XSLT uses XPath to define parts of the source document that should match one or more predefined templates. When a match is found, XSLT will transform the matching part of the source document into the result document.

XSLT

- The root element that declares the document to be an XSL style sheet is `<xsl:stylesheet>` or `<xsl:transform>`.
- `<xsl:stylesheet>` and `<xsl:transform>` are completely synonymous and either can be used.

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

or

```
<xsl:transform version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

XSLT

- An XSL style sheet consists of one or more set of rules that are called templates.
- A template contains the rules to apply when a specified node is matched.
- The `<xsl:template>` element is used to build templates.
- The `match` attribute is used to associate a template with an XML element.
- The `match` attribute can also be used to define a template for the entire XML document.
- The value of the `match` attribute is an XPath expression (i.e. `match="/"` defines the whole document).
- The content inside the `<xsl:template>` element defines how to produce the output document (either XML or XHTML).
- The `<xsl:value-of>` element is used to extract the value of a selected node.
- The `<xsl:value-of>` element can be used to extract the value of an XML element and add it to the output stream of the transformation.

XSLT

- The XSL `<xsl:for-each>` element can be used to select every XML element of a specified node-set.

```
<xsl:for-each select="bookstore/book">
….
</xsl:for-each>
```

- The value of the `select` attribute is an XPath expression.

```
<xsl:for-each select="bookstore/book[title='Poezii']">
…
</xsl:for-each>
```

XSLT

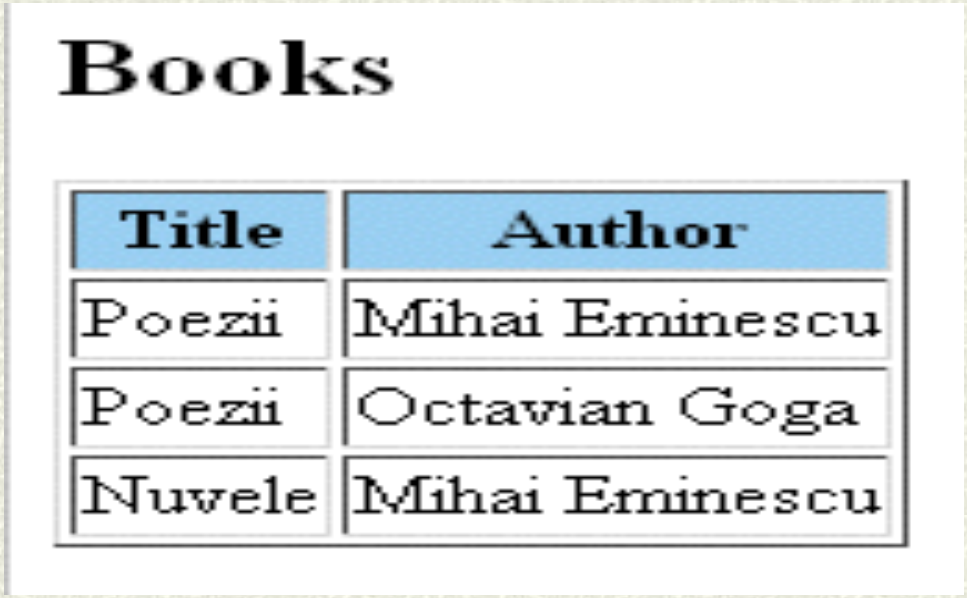
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
  <book>
    <title>Poezii</title>
    <author>Mihai Eminescu</author>
     <price>15.00</price>
    <year>1995</year>
  </book>
  <book>
    <title>Poezii</title>
    <author>Octavian Goga</author>
     <price>19.00</price>
    <year>1999</year>
  </book>
<book>
    <title>Nuvele</title>
    <author>Mihai Eminescu</author>
     <price>22.00</price>
    <year>2006</year>
  </book>
</bookstore>
```

XSLT

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html> <body>
  <h2>Books</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>   <th>Author</th>
    </tr>
    <xsl:for-each select="bookstore/book">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="author"/></td>
    </tr>
    </xsl:for-each>
  </table>
  </body>   </html>
</xsl:template>
</xsl:stylesheet>
```

XSLT

Result



Books

Title	Author
Poezii	Mihai Eminescu
Poezii	Octavian Goga
Nuvele	Mihai Eminescu

XSLT

- The `<xsl:sort>` element is used to sort the output.
- In order to sort the output, just add an `<xsl:sort>` element inside the `<xsl:for-each>` element in the XSL file:

```
<xsl:for-each select="bookstore/book">
    <xsl:sort select="author"/>
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="author"/></td>
    </tr>
</xsl:for-each>
```

- The `select` attribute indicates what XML element to sort on.

XSLT

- The `<xsl:if>` element is used to put a conditional test against the content of the XML file. The value of the required `test` attribute contains the expression to be evaluated.

```
<xsl:if test="expression">
     ...some output if the expression is true...
</xsl:if>
```

- The `<xsl:if>` element must be placed inside the `<xsl:for-each>` element in the XSL file.

```
<xsl:for-each select="bookstore/book">
      <xsl:if test="price &gt; 10">
        <tr>
          <td><xsl:value-of select="title"/></td>
          <td><xsl:value-of select="author"/></td>
        </tr>
      </xsl:if>
 </xsl:for-each>
```

XSLT

- The `<xsl:choose>` element is used in conjunction with `<xsl:when>` and `<xsl:otherwise>` to express multiple conditional tests.

```
<xsl:for-each select="bookstore/book">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <xsl:choose>
        <xsl:when test="price &gt; 10">
          <td bgcolor="#ff00ff">
          <xsl:value-of select="author"/></td>
        </xsl:when>
        <xsl:when test="price &gt; 9">
          <td bgcolor="#cccccc">
          <xsl:value-of select="author"/></td>
        </xsl:when>
        <xsl:otherwise>
          <td><xsl:value-of select="author"/></td>
        </xsl:otherwise>
      </xsl:choose>
    </tr>
</xsl:for-each>
```

XSLT

- The `<xsl:apply-templates>` element applies a template to the current element or to the current element's child nodes.
- If a `select` attribute is added to the `<xsl:apply-templates>` element it will process only the child element that matches the value of the attribute.
- The `select` attribute can be used to specify the order in which the child nodes are processed.

XSLT

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html> <body> <h2>Books</h2>
  <xsl:apply-templates/>
  </body>  </html>
</xsl:template>
<xsl:template match="book">
  <p> <xsl:apply-templates select="title"/>
  <xsl:apply-templates select="author"/>   </p>
</xsl:template>
<xsl:template match="title">
  Title: <span style="color:#ff0000">
  <xsl:value-of select="."/></span>   <br />
</xsl:template>
<xsl:template match="author">
  Author: <span style="color:#00ff00">
  <xsl:value-of select="."/></span>   <br />
</xsl:template>
</xsl:stylesheet>
```

XSLT

- Other xsl elements:
  - `<xsl:attribute>` Creates an attribute node and attaches it to an output element.
  - `<xsl:copy>` Copies the current node from the source to the output.
  - `<xsl:comment>` Generates a comment in the output.
  - `<xsl:text>` Generates text node from a style sheet. White-space-only nodes are preserved in the output.
  - `<xsl:element>` Creates an output element with the specified name.

API for C#

- Namespaces:
  - `System.Xml.XPath`: Infrastructure and API for Xpath
  - `System.Xml.Xsl`: Infrastructure and API for performing XSLT transformations of XML
- `System.Xml.XPath`
  - `XPathDocument` Provides a fast, read-only, in-memory representation of an XML document by using the XPath data model.
  - `XPathException` Provides the exception thrown when an error occurs while processing an XPath expression.
  - `XPathExpression` Provides a typed class that represents a compiled XPath expression.
  - `XPathNavigator` Provides a cursor model for navigating and editing XML data.
  - `XPathNodeIterator` Provides an iterator over a selected set of nodes.
- `System.Xml.Xsl`
  - `XslCompiledTransform` Transforms XML data using an XSLT style sheet.

C# API

XPath queries can be executed within code in the following ways:

- Call one of the `SelectXYZ` methods on an `XmlDocument` or `XmlNode`.
- Spawn an `XPathNavigator` from either:
  - An `XmlDocument`
  - An `XPathDocument`

C# API

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<customers>
  <customer id="123">
    <firstname>Jim</firstname>
    <lastname>Bo</lastname>
  </customer>
  <customer id="234">
    <firstname>Thomas</firstname>
    <lastname>Jones</lastname>
  </customer>
</customers>

XmlDocument doc = new XmlDocument(  );
doc.Load ("customers.xml");
XmlNode n = doc.SelectSingleNode ("customers/customer[firstname='Jim']");
Console.WriteLine (n.InnerText);  // JimBo

XmlNodeList nodes = doc.SelectNodes ("//lastname");
```

XPathNavigator

- `XPathNavigator` is a cursor over the XPath representation of an XML document.
- It is loaded with primitive methods that move the cursor around the tree (e.g., move to parent, move to first child, etc.).
- The `XPathNavigator's Select*` methods take an XPath string to express more complex navigations or queries that return multiple nodes.

```
XPathNavigator nav = doc.CreateNavigator(  );
XPathNavigator jim = nav.SelectSingleNode("customers/
   customer[firstname='Jim']");
Console.WriteLine (jim.Value);                    // JimBo
```

The `SelectSingleNode` method returns a single `XPathNavigator`.

The `Select` method returns an `XPathNodeIterator`, which simply iterates over multiple `XPathNavigators`.

XPathNavigator

```
XPathNavigator nav = doc.CreateNavigator(  );
string xPath = "customers/customer/firstname/text(  )";
foreach (XPathNavigator navC in nav.Select (xPath))
  Console.WriteLine (navC.Value);

OUTPUT:
Jim
Thomas
```

- To perform faster queries, you can compile an XPath query into an XPathExpression.
- You then pass the compiled expression to a Select* method, instead of a string.

```
XPathNavigator nav = doc.CreateNavigator(  );
XPathExpression expr = nav.Compile ("customers/customer/firstname");
foreach (XPathNavigator a in nav.Select (expr))
  Console.WriteLine (a.Value);

OUTPUT:
Jim
Thomas
```

XPathDocument

- `XPathDocument` is used for read-only XML documents that conform to the XPath Model. An `XPathNavigator` backed by an `XPathDocument` is faster than an `XmlDocument`, but it cannot make changes to the underlying document:

```
XPathDocument doc = new XPathDocument ("customers.xml");
XPathNavigator nav = doc.CreateNavigator(  );
foreach (XPathNavigator a in nav.Select ("customers/customer/
   firstname"))
  Console.WriteLine (a.Value);
```

```
OUTPUT:
Jim
Thomas
```

XslCompiledTransform

- The **`System.Xml.Xsl.XslCompiledTransform`** transform class efficiently performs XLST transforms.

```
XslCompiledTransform transform = new XslCompiledTransform();
transform.Load ("test.xsl");
transform.Transform ("input.xml", "output.xml");
```

- Generally, it's more useful to use the overload of **`Transform`** that accepts an **`XmlWriter`** rather than an output file, so you can control the formatting.

```
XmlWriterSettings settings = new XmlWriterSettings();
settings.Indent = true;
settings.IndentChars = "\t";
XmlWriter writer = XmlWriter.Create("output.xml", settings);

xslt.Transform("input.xml", writer);
writer.Close();
```

Java API

- **`javax.xml.xpath`**: This package provides an object-model neutral API for the evaluation of XPath expressions.
- **`javax.xml.transform`**: This package defines the factory class you use to get a **`Transformer`** object. The transformer must be configured with input (source) and output (result) objects, and invoke its **`transform()`** method to make the transformation.
- **`javax.xml.transform.dom`**: Defines the **`DOMSource`** and **`DOMResult`** classes, which let you use a DOM as an input to or output from a transformation.
- **`javax.xml.transform.sax`**: Defines the **`SAXSource`** and **`SAXResult`** classes, which let you use a SAX event generator as input to a transformation, or deliver SAX events as output to a SAX event processor.
- **`javax.xml.transform.stream`**: Defines the **`StreamSource`** and **`StreamResult`** classes, which let you use an I/O stream as an input to or output from a transformation.

Java API

- Package `javax.xml.xpath`:
  - `XPath` provides access to the XPath evaluation environment and expressions.
  - `XPathExpression` provides access to compiled XPath expressions.
  - `XPathFunction` provides access to XPath functions.
  - `XPathConstants` XPath constants.
  - `XPathFactory` An XPathFactory instance is used to create XPath objects.
  - `XPathException` represents a generic XPath exception

```
XPath xpath = XPathFactory.newInstance().newXPath();
String expression = "/customers/customer";
InputSource inputSource = new InputSource("customers.xml");
NodeList nodes = (NodeList) xpath.evaluate(expression, inputSource,
   XPathConstants.NODESET);
```

Java API

```
// parse the XML as a W3C Document
DocumentBuilder builder =
   DocumentBuilderFactory.newInstance().newDocumentBuilder();
Document document = builder.parse(new File("/customers.xml"));

XPath xpath = XPathFactory.newInstance().newXPath();
String expression = "/customers/customer[firstname='Jim']";
Node custNode = (Node) xpath.evaluate(expression, document,
   XPathConstants.NODE);

XPath xpath = XPathFactory.newInstance().newXPath();
String expression = “firstname";
Node firstnameNode = (Node) xpath.evaluate(expression, custNode,
   XPathConstants.NODE);
```

javax.xml.transform

- It contains generic APIs for processing transformation instructions, and performing a transformation from source to result.
- `Result` The interface for holding the information needed to build a transformation result tree.
- `Source` The interface for holding the information needed to act as source input (XML source or transformation instructions).
- `Transformer` An instance of this abstract class can transform a source tree into a result tree.
- `TransformerFactory` A TransformerFactory instance can be used to create `Transformer` objects.
- `TransformerConfigurationException` Indicates a serious configuration error.
- `TransformerException` This class specifies an exceptional condition that occured during the transformation process.

javax.xml.transform

```
import java.io.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
class XSLTransformer {
   public static void main(String[] args) {
      try {
         File sourcefile = new File(args[0]);
         File resultfile = new File(args[1]);
         File templatefile = new File(args[2]);
          TransformerFactory fac = TransformerFactory.newInstance();
  Transformer t = fac.newTransformer(new StreamSource(templatefile));
          Source source = new StreamSource(sourcefile);
          Result result = new StreamResult(resultfile);
          t.transform(source, result);
      } catch (TransformerConfigurationException e) { …
      } catch (TransformerException e) { …
      }
  }
}
```