

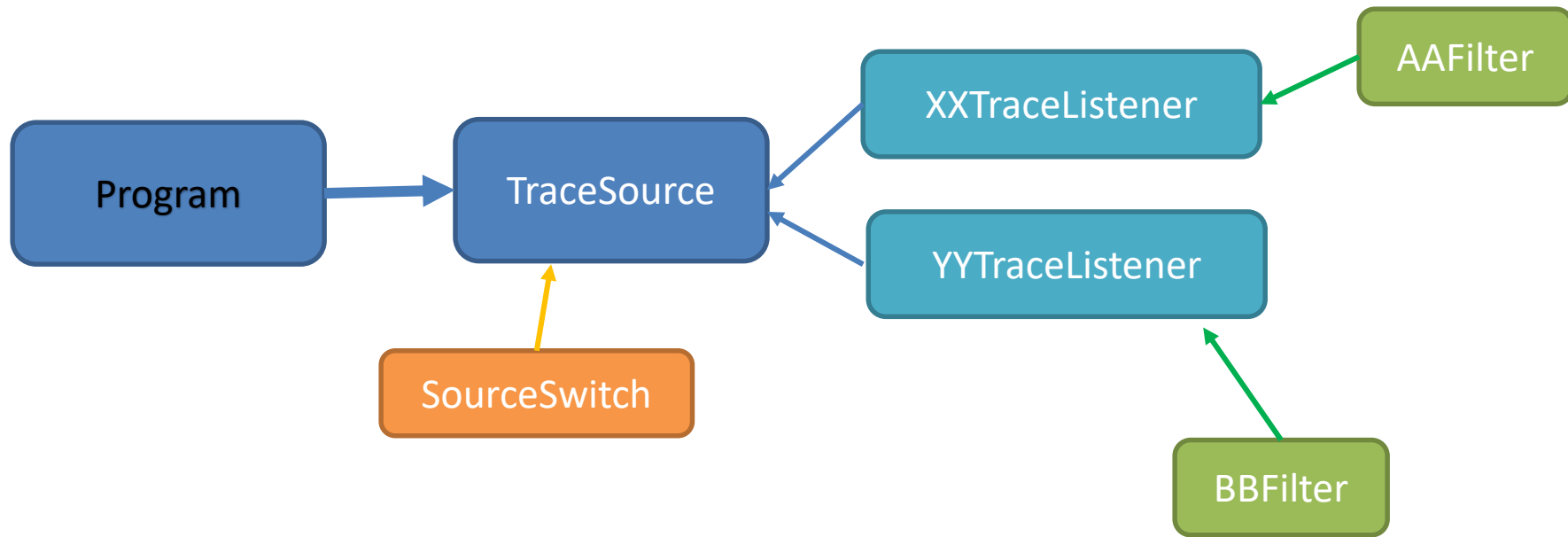
Server Framework2

Logging
Configuration

Tracing / logging information

- Instead of using `Console.WriteLine` use tracing / logging for released Systems.
- You can setup the log to write to:
 - The Console
 - A File, in different formats
 - (Windows Event Log)
- The Tracing can have several output channels
- The Trace level can be changed (actual at runtime)

Overview Tracing / logging



How to Choose Output Chanel

- The **TraceSource** class can write to “TraceListener”
- The “TraceListener” is an abstract class
i.e. you need concrete TraceListener class.
- They work like observers
i.e. you can add them to a TraceSource (ts) object like:
ts.Listeners.Add(objOfTraceListener) ;
- C# have some buildt in classes like:
 - TextWriterTraceListener
 - XmlWriterTraceListener
 - EventLogTraceListener
- Customer Created Listener

Make your own TraceListener class

- You can design and implement you own Listener by Inherits from **TraceListener** and override:
 - **public override void Write(string message)**
 - **public override void WriteLine(string message)**

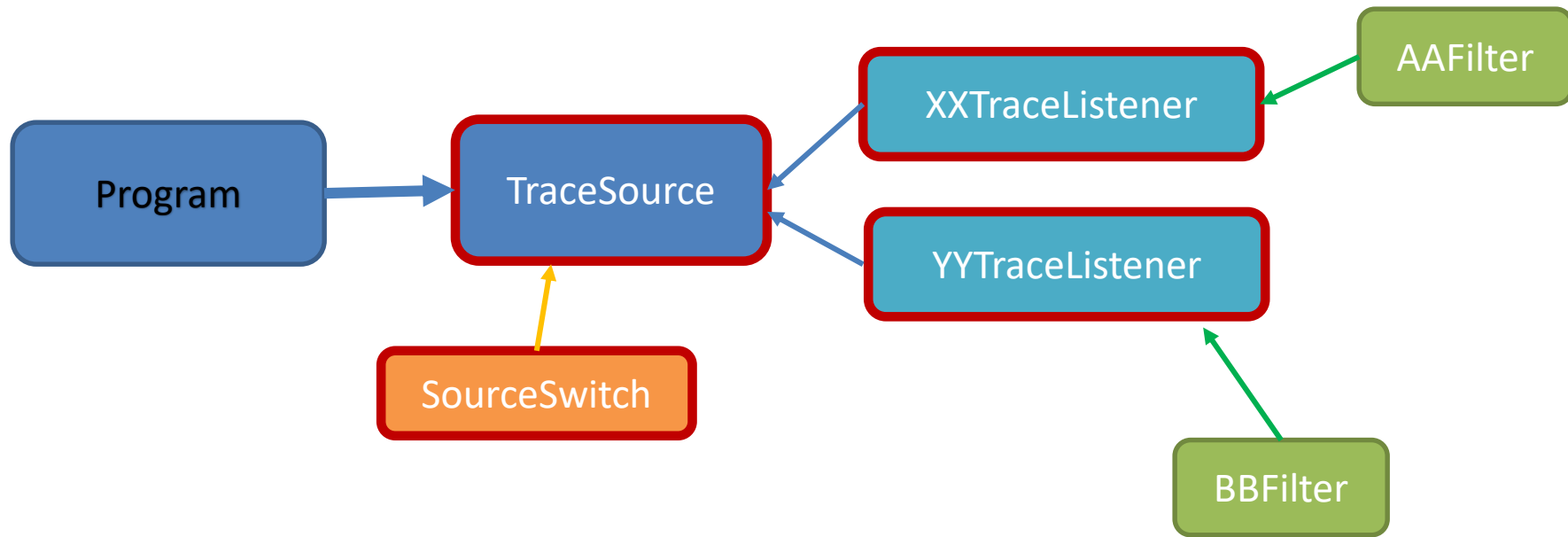
Trace Level

- TraceSource works with diff. Levels of logging
 - Verbose
 - Info
 - Warning
 - Error
 - Critical
- Setting actual levels of logging ex:
`ts.Switch = new SourceSwitch("Peters","All");`
- You specify the level when you logging like:
`ts.TraceEvent(TraceEventType.Error, <<ID>>, <<Object/string to log>>);`

Example:

```
ts.TraceEvent(TraceEventType.Error, 333, "This is an Error");
```

Overview Tracing / logging



Trace Filters

- TraceSource can take filters to configure the individual TraceListener
- Types of filters:
 - SourceFilter (build in) -- for configure which part of the system to log
 - EventTypeFilter (build in) -- for configure level of logging messages to log
 - Customer Created Filters
- Example of filter setting:
`xxListener.Filter = new EventTypeFilter(SourceLevels.All);`

Make your own Filter

- You can design and implement you own Filter by Inherits from **TraceFilter** and override:

- **public override bool ShouldTrace(**

- TraceEventCache cache,

- string source,**

- TraceEventType eventType,**

- int id,**

- string formatOrMessage,**

- object[] args,

- object data1,

- object[] data)

- some metadata

- **where does it come from**

- **the level error,warning...**

- **some id**

- **the text string – can be null**

- additional inf.

- additional inf.

- additional inf.

Special TraceListener - EventLog

- The `EventLogTraceListener` will log to the system Event Log System (use EventViewer to lookup the logging)
- Need to get NuGet Package (`System.Diagnostics.EventLog`)

- **Example:**

```
TraceListener logListener = new EventLogTraceListener("Application");  
ts.Listeners.Add(logListener);
```

Demo

... then exercise.

XML intro

What is XML?

- XML stands for **EX**tensible **M**arkup **L**anguage
- XML is a **markup language** much like HTML
- XML was designed to **carry data**, not to display data
- XML tags are not predefined. You must **define your own tags**
- XML is designed to be **self-descriptive**
- XML is a **W3C Recommendation**

The Difference Between XML and HTML

- **XML is not a replacement for HTML.**
- XML and HTML were designed with different goals:
 - **XML** was designed to **transport** and **store** data, with focus on what data is. (like model)
 - **HTML** was designed to **display** data, with focus on how data looks. (like view)

Therefore - HTML is about displaying information, while XML is about carrying information.

The Difference Between XML and JSON

- **Both for carrying information (share data).**
- **Json is shorter in bytes**
- **XML can be validated**
- **Json often used in REST-services**
- **XML often used in configuration**

XML Example

```
<note>  
  <to>Tove</to>  
  <from>Jani</from>  
  <heading>Reminder</heading>  
  <body>Don't forget me this Weekend!</body>  
</note>
```

```
{"Note": {"To": "Tove", "From": "Jani", "Heading":  
"Reminder", "Body": "Don\u0027t forget me  
this Weekend!"}}
```


XML Simplifies Data Sharing

- XML data is stored in plain text format.
- Meaning it is software- and hardware-independent.
- With XML, data can easily be exchanged between incompatible systems.

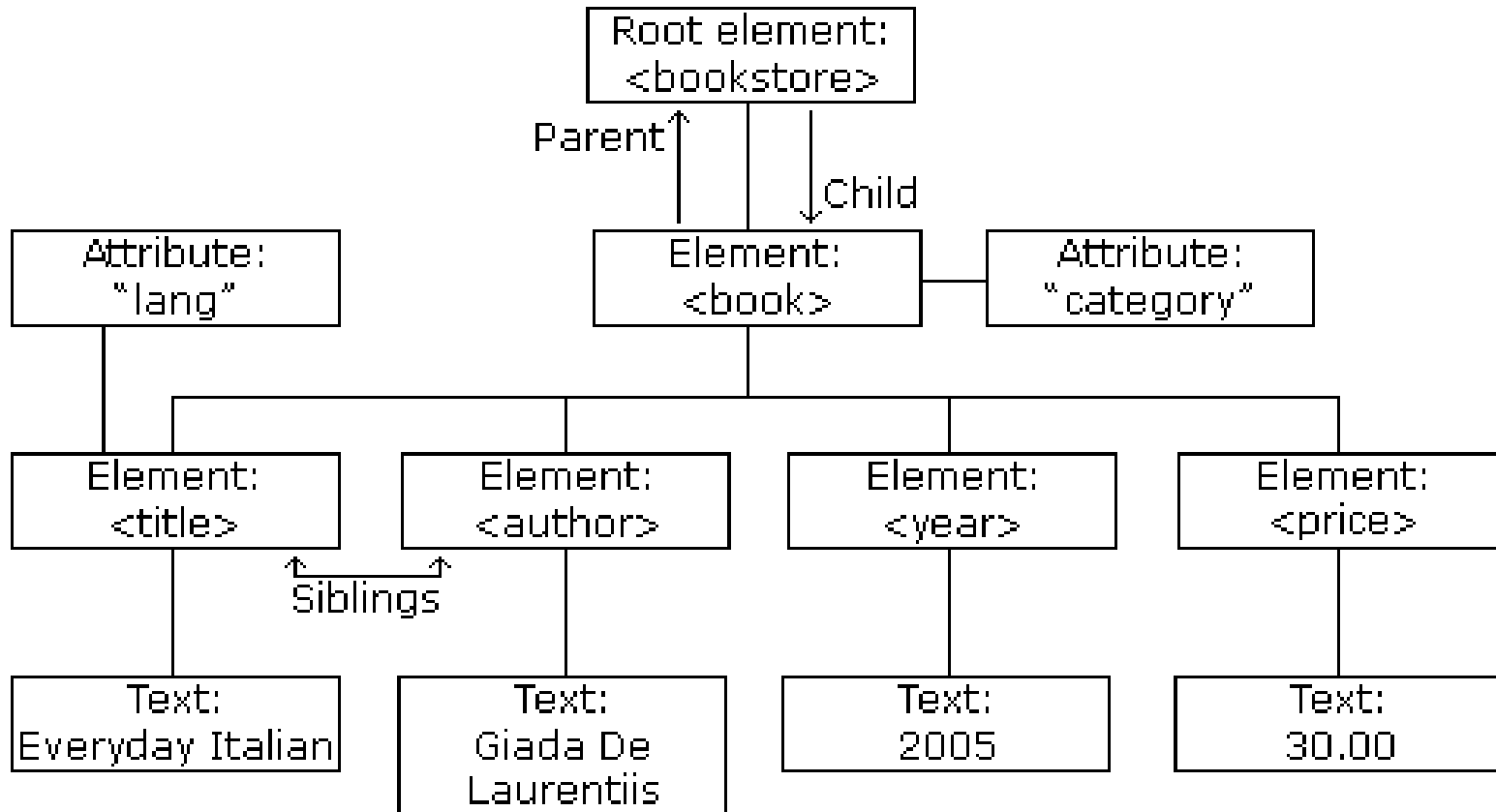
XML Documents Form a Tree Structure

- XML documents must contain a **root element**.
This element is "the parent" of all other elements.
NB! Only one root element are allowed
- The elements in an XML document form a document tree.
- The tree starts at the root.

XML Documents – General structure

- All elements can have sub elements (child elements):
- ```
<root>
 <child>
 <subchild>.....</subchild>
 </child>
 <child> // sibling
 <subchild>.....</subchild>
 </child>
</root>
```
- Parent elements have children. Children on the same level are called siblings (brothers or sisters).

# Example of XML-dom-tree



```
<bookstore>
<book category="COOKING">
 <title lang="en">Everyday Italian</title>
 <author>Giada De Laurentiis</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>Erik T. Ray</author>
 <year>2003</year>
 <price>39.95</price>
</book>
</bookstore>
```

A diagram consisting of two red arrows. One arrow starts at the root element '<bookstore>' at the top left of the XML code block and points to the first '<book>' element. The second arrow starts at the '</bookstore>' element at the bottom left and points back to the root '<bookstore>' element, forming a triangle that encompasses the entire XML document structure.

The <book>  
element itself has  
4 children:

<title>, <author>,  
<year>, <price>.

The **root** element in the example is <bookstore>. All <book> elements in the document are contained within <bookstore>.

# XML Syntax Rules – to be wellformed

- **All XML Elements Must Have a Closing Tag**
- **XML Tags are Case Sensitive**
- **XML Documents must have one Root Element**
- **XML Elements must be Properly Nested**
- **XML Attribute values must be Quoted**
- **Entity References**

# XML Elements vs. Attributes

- Take a look at these two examples:

```
<person sex="female"> // Attribute
<firstname>Anna</firstname> // Sex inf. to 'person'-tag
<lastname>Smith</lastname>
</person>
```

- ```
<person>                          // Element  
<sex>female</sex>                // Sex separate tag  
<firstname>Anna</firstname>  
<lastname>Smith</lastname>  
</person>
```

- Both examples provide the same information.
- There are no rules about when to use attributes and when to use elements. But in general use elements **except** for metadata.

Valid XML Documents

- A "Valid" XML document is
 - "Well Formed" XML document
 - Conforms to a Document Type Definition (DTD): *(or schema)*
- ```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE note SYSTEM "Note.dtd">
<note>
<to>Tove</to><from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```
- The DOCTYPE declaration in the example above, is a reference to an external DTD file.



# XML DTD (ex: note.dtd)

- The purpose of a DTD is to define the structure of an XML document. It defines the structure with a list of legal elements:
- ```
<!DOCTYPE note [  
<!ELEMENT note (to,from,heading,body) >  
<!ELEMENT to (#PCDATA) >  
<!ELEMENT from (#PCDATA) >  
<!ELEMENT heading (#PCDATA) >  
<!ELEMENT body (#PCDATA) >  
>
```
- xxx+ -> 1-many xxx* -> 0-many xxx? -> 0-1
- , -> and | -> or

XML Schema

- W3C supports an XML based alternative to DTD called XML Schema:
- ```
<xs:element name="note">
<xs:complexType>
 <xs:sequence>
 <xs:element name="to" type="xs:string"/>
 <xs:element name="from" type="xs:string"/>
 <xs:element name="heading" type="xs:string"/>
 <xs:element name="body" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
</xs:element>
```

# Reading XML files in C#

## Example:

- To open config-file use:

```
XmlDocument configDoc = new XmlDocument();
configDoc.Load("<< configFileName >> ");
```

- To read a port number:

```
XmlNode xxNode =
configDoc.DocumentElement.SelectSingleNode("<NameOf
fTag>");
if (xxNode != null)
{
 String xxStr = xxNode.InnerText.Trim();
 Int xx = Convert.ToInt32(xxStr);
}
```

# Demo

... then exercise.