Unit 2

Concepts of Lose-Coupling revisited. Generating JAR files, Concept of Class loading, J2EE Design Patterns, Markup Languages, XML, What is XML? Document type Definitions (DTDs), XML namespaces, XML schema, XPath, XSL transformation, APIs, J2EE Best Practices and Frameworks.

Coupling in Java

A situation where an object can be used by another object is termed as coupling. It is the process of collaborating together and working for each other. It simply means that one object requires another object to complete its assigned task. It is basically the usage of an object by another object, thereby reducing the dependency between the modules. It is called as collaboration if one class calls the logic of another class.

Types of Coupling

Couling in Java is further divided into two types, namely:

- Tight coupling
- Loose coupling

Let's understand each one of them.

Tight Coupling: It is when a group of classes are highly dependent on one another. This scenario arises when a class assumes too many responsibilities, or when one concern is spread over many classes rather than having its own class. The situation where an object creates another object for its usage, is termed as Tight Coupling. The parent object will be knowing more about the child object hence the two objects are called as tightly coupled. The dependency factor and the fact that the object cannot be changed by anybody else helps it to achieve the term, tightly coupled.

Example: Suppose you have made two classes. First class is a class called Volume, and the other class evaluates the volume of the box. Any changes that would be made in the Volume class, would be reflecting in the Box class. Hence, both the classes are interdependent on each other. This situation particularly is called as tight coupling.

Below shown code will help you in understanding the implementation process of tight coupling.

Example 1:

```
package tightcoupling;

class Volume {
    public static void main(String args[]) {
        Box b = new Box(15, 15, 15);
        System.out.println(b.volume);
    }
}

class Box {
    public int volume;
    Box(int length, int width, int height) {
        this.volume = length * width * height;
    }
}
```

Output:

```
3375
```

In the above example, you can see how the two classes are bound together and work as a team. This was a simple example of tight coupling in Java. Another example depicting the process!

Example 2:

```
package tightcoupling;

public class ABC{
    public static void main(String args[]) {
        A a = new A();
        a.display();
    }
}

class A {
    B b;
    public A() {
         b = new B();
    }
    public void display() {
         System.out.println("A");
         b.display();
    }
}
```

```
class B {
      public B() {
      }
      public void display() {
            System.out.println("B");
       }
}
```

Output:

```
A
B
```


Loose Coupling: When an object gets the object to be used from external sources, we call it loose coupling. In other words, the loose coupling means that the objects are independent. A loosely coupled code reduces maintenance and efforts. This was the disadvantage of tightly coupled code that was removed by the loosely coupled code. Let's take a look at some of the examples of loose coupling in Java.

Example 1:

```
package lc;

class Volume {
     public static void main(String args[]) {
          Box b = new Box(25, 25, 25);
          System.out.println(b.getVolume());
     }
}

final class Box {
      private int volume;
      Box(int length, int width, int height) {
            this.volume = length * width * height;
      }
      public int getVolume() {
            return volume;
      }
}
```

Output:

```
15625
```

Example 2:

```
package losecoupling;

import java.io.IOException;

public class ABC {
    public static void main(String args[]) throws IOException {
        Show b = new B();
        Show c = new C();
        A a = new A(b);
        a.display();
        A a1 = new A(c);
        a1.display();
    }
}

interface Show {
    public void display();
}

class A {
    Show s;
    public A(Show s) {
        this.s = s;
    }
    public void display() {
        System.out.println("A");
        s.display();
     }
}

class B implements Show {
     public B() {
     }
     public void display() {
         System.out.println("B");
     }
}

class C implements Show {
     public C() {
     }
     public void display() {
         System.out.println("C");
     }
```

}

Output:

A
B
A
C

Difference between Tight Coupling and Loose Coupling

Tight Coupling	Loose Coupling
More interdependency	Less Dependency, better test-ability
Follows GOF principles of the program to interface	Does not provide the concept of interface
Synchronous Communication	Asynchronous Communication
More coordination, swapping a piece of code/objects between two objects are easy	Less coordination, not easy

Generating JAR files

Creating an Executable Jar File

In Java, it is common to combine several classes in one .jar ("java archive") file. Library classes are stored that way. Larger projects use jar files. You can create your own jar file combining several classes, too.

jar files are created using the jar.exe utility program from the JDK. You can make your jar file runnable by telling jar.exe which class has main. To do that, you need to create a manifest file. A manifest is a one-line text file with a "Main-Class" directive. For example:

Main-Class: Craps

This line must end with a newline.

A jar file created with a main class manifest can be used both as a library and a runnable jar. If you use it as a library, you can edit and compile any of the classes included in the jar, and add it to your project. Then it will override the one in the jar file.

You can create a manifest file in any text editor, or even by using the *MS-DOS* echo command. You can give your manifest file any name, but it's better to use something standard, such as manifest.txt.

Once you have a manifest and all your classes have been compiled, you need to run JDK's jar.exe utility. It is located in the JDK's bin folder, the same place where javac.exe and java.exe are. jar.exe takes command-line arguments; if you run it without any arguments, it will display the usage information and examples. You need

```
C\mywork> jar cvfm MyJarName.jar manifest.txt *.class
```

cvfm means "create a jar; show verbose output; specify the output jar file name; specify the manifest file name." This is followed by the name you wish to give to your jar file, the name of your manifest file, and the list of .class files that you want included in the jar. *.class means all class files in the current directory.

Actually, if your manifest contains only the Main-Class directive, you can specify the main class directly on the jar.exe's command line, using the e switch, instead of m. Then you do not need a separate manifest file; jar will add the required manifest to your jar file for you. For example:

```
C\mywork> jar cvfe MyJarName.jar MyMainClass *.class
```

Below is a reference for creating a jar file in *Eclipse* and the detailed steps for doing this in *Command Prompt* and in *JCreator*.

Creating a jar File in *Eclipse*

In *Eclipse* Help contents, expand "Java development user guide" ==> "Tasks" ==> "Creating JAR files." Follow the instructions for "Creating a new JAR file" or "Creating a new runnable JAR file."

The JAR File and Runnable JAR File commands are for some reason located under the File menu: click on Export... and expand the Java node.

Creating a jar File in *JCreator*

You can configure a "tool" that will automate the jar creation process. You only need to do it once.

1. Click on Configure/Options.
2. Click on Tools in the left column.
3. Click New, and choose Create Jar file.
4. Click on the newly created entry Create Jar File in the left column under Tools.
5. Edit the middle line labeled Arguments: it should have

   ```
   cvfm $[PrjName].jar manifest.txt *.class
   ```

6. Click OK.

Now set up a project for your program, create a manifest file manifest.txt or copy and edit an existing one. Place manifest.txt in the same folder where the .class files go. Under View/Toolbars check the Tools toolbar. Click on the corresponding tool button or press Ctrl-1 (or Ctrl-*n* if this is the *n*-th tool) to run the Create Jar File tool.

With *Windows Explorer*, go to the jar file that you just created and double click on it to run.

Creating a jar File in *Command Prompt*

1. Start *Command Prompt*.
2. Navigate to the folder that holds your class files:

   ```
   C:\>cd \mywork
   ```

3. Set path to include JDK's bin. For example:

   ```
   C:\mywork> path c:\Program Files\Java\jdk1.7.0_25\bin;%path%
   ```

4. Compile your class(es):

   ```
   C:\mywork> javac *.java
   ```

5. Create a manifest file and your jar file:

   ```
   C:\mywork> echo Main-Class: Craps >manifest.txt
   C:\mywork> jar cvfm Craps.jar manifest.txt *.class
   ```

   or

   ```
   C:\mywork> jar cvfe Craps.jar Craps *.class
   ```

6. Test your jar:

   ```
   C:\mywork> Craps.jar
   ```

or

C:\mywork> java -jar Craps.jar

ClassLoader in Java

The Java ClassLoader is a part of the Java Runtime Environment that dynamically loads Java classes into the Java Virtual Machine. The Java run time system does not need to know about files and file systems because of classloaders.

Java classes aren't loaded into memory all at once, but when required by an application. At this point, the Java ClassLoader is called by the JRE and these ClassLoaders load classes into memory dynamically.

Types of ClassLoaders in Java

Not all classes are loaded by a single ClassLoader. Depending on the type of class and the path of class, the ClassLoader that loads that particular class is decided. To know the ClassLoader that loads a class the *getClassLoader()* method is used. All classes are loaded based on their names and if any of these classes are not found then it returns a NoClassDefFoundError or ClassNotFoundException.

A Java Classloader is of three types:

1. BootStrap ClassLoader: A Bootstrap Classloader is a Machine code which kickstarts the operation when the JVM calls it. It is not a java class. Its job is to load the first pure Java ClassLoader. Bootstrap ClassLoader loads classes from the location *rt.jar*. Bootstrap ClassLoader doesn't have any parent ClassLoaders. It is also called as the Primodial ClassLoader.
2. Extension ClassLoader: The Extension ClassLoader is a child of Bootstrap ClassLoader and loads the extensions of core java classes from the respective JDK Extension library. It loads files from *jre/lib/ext* directory or any other directory pointed by the system property *java.ext.dirs*.
3. System ClassLoader: An Application ClassLoader is also known as a System ClassLoader. It loads the Application type classes found in the environment variable *CLASSPATH, -classpath or -cp command line option*. The Application ClassLoader is a child class of Extension ClassLoader.

Note: The ClassLoader Delegation Hierarchy Model always functions in the order Application ClassLoader->Extension ClassLoader->Bootstrap ClassLoader. The Bootstrap ClassLoader is always given the higher priority, next is Extension ClassLoader and then Application ClassLoader.

Principles of functionality of a Java ClassLoader

Principles of functionality are the set of rules or features on which a Java ClassLoader works. There are three principles of functionality, they are:

1. Delegation Model: The Java Virtual Machine and the Java ClassLoader use an algorithm called the Delegation Hierarchy Algorithm to Load the classes into the Java file.

   The ClassLoader works based on a set of operations given by the delegation model. They are:

   - ClassLoader always follows the Delegation Hierarchy Principle.
   - Whenever JVM comes across a class, it checks whether that class is already loaded or not.
   - If the Class is already loaded in the method area then the JVM proceeds with execution.
   - If the class is not present in the method area then the JVM asks the Java ClassLoader Sub-System to load that particular class, then ClassLoader sub-system hands over the control to Application ClassLoader.
   - Application ClassLoader then delegates the request to Extension ClassLoader and the Extension ClassLoader in turn delegates the request to Bootstrap ClassLoader.
   - Bootstrap ClassLoader will search in the Bootstrap classpath(JDK/JRE/LIB). If the class is available then it is loaded, if not the request is delegated to Extension ClassLoader.
   - Extension ClassLoader searches for the class in the Extension Classpath(JDK/JRE/LIB/EXT). If the class is available then it is loaded, if not the request is delegated to the Application ClassLoader.
   - Application ClassLoader searches for the class in the Application Classpath. If the class is available then it is loaded, if not then a ClassNotFoundException exception is generated.

2. Visibility Principle: The Visibility Principle states that a class loaded by a parent ClassLoader is visible to the child ClassLoaders but a class loaded by a child ClassLoader is not visible to the parent ClassLoaders. Suppose a class GEEKS.class has been loaded by the Extension ClassLoader, then that class is only visible to the Extension ClassLoader and Application ClassLoader but not to the Bootstrap ClassLoader. If that class is again tried to load using Bootstrap ClassLoader it gives an exception *java.lang.ClassNotFoundException*.
3. Uniqueness Property: The Uniquesness Property ensures that the classes are unique and there is no repetition of classes. This also ensures that the classes loaded by parent classloaders are not loaded by the child classloaders. If the parent class loader isn't able to find the class, only then the current instance would attempt to do so itself.

Methods of Java.lang.ClassLoader

After the JVM requests for the class, a few steps are to be followed in order to load a class. The Classes are loaded as per the delegation model but there are a few important Methods or Functions that play a vital role in loading a Class.

1. loadClass(String name, boolean resolve): This method is used to load the classes which are referenced by the JVM. It takes the name of the class as a parameter. This is of type loadClass(String, boolean).
2. defineClass(): The defineClass() method is a *final* method and cannot be overriden. This method is used to define a array of bytes as an instance of class. If the class is invalid then it throws ClassFormatError.
3. findClass(String name): This method is used to find a specified class. This method only finds but doesn't load the class.
4. findLoadedClass(String name): This method is used to verify whether the Class referenced by the JVM was previously loaded or not.
5. Class.forName(String name, boolean initialize, ClassLoader loader): This method is used to load the class as well as initialize the class. This method also gives the option to

choose any one of the ClassLoaders. If the ClassLoader parameter is NULL then Bootstrap ClassLoader is used.

6. getName()
7. getPackage()

Example: The following code is executed before a class is loaded:

filter_none

brightness_4

```
protected synchronized Class<?>
loadClass(String name, boolean resolve)
   throws ClassNotFoundException
{
   Class c = findLoadedClass(name);
   try {
      if (c == NULL) {
         if (parent != NULL) {
            c = parent.loadClass(name, false);
         }
         else {
            c = findBootstrapClass0(name);
         }
      }
      catch (ClassNotFoundException e)
      {
         System.out.println(e);
      }
   }
}
```

Note: If a class has already been loaded, it returns it. Otherwise, it delegates the search for the new class to the parent class loader. If the parent class loader doesn't find the class, loadClass() calls the method findClass() to find and load the class. The finalClass() method searches for the class in the current ClassLoader if the class wasn't found by the parent ClassLoader.

How ClassLoader works in Java

When JVM request for a class, it invokes a loadClass() method of the java.lang.ClassLoader class by passing the fully classified name of the class. The loadClass() method calls for

findLoadedClass() method to check that the class has been already loaded or not. It is required to avoid loading the class multiple times.

If the class is already loaded, it delegates the request to parent ClassLoader to load the class. If the ClassLoader is not finding the class, it invokes the findClass() method to look for the classes in the file system. The following diagram shows how ClassLoader loads class in Java using delegation.

According to the uniqueness principle, a class loaded by the parent should not be loaded by Child ClassLoader again. So, it is possible to write class loader which violates delegation and uniqueness principles and loads class by itself.

In short, class loader follows the following rule:

- It checks if the class is already loaded.
- If the class is not loaded, ask parent class loader to load the class.
- If parent class loader cannot load class, attempt to load it in this class loader.

Consider the following Example:

```
public class Demo
{
public static void main(String args[])
{
System.out.println("How are you?");
}
}
```

Compile and run the above code by using the following command:

1. javac Demo.java
2. java -verbose:class Demo

-verbose:class: It is used to display the information about classes being loaded by JVM. It is useful when using class loader for loading classes dynamically. The following figure shows the output.

We can observe that runtime classes required by the application class (Demo) are loaded first.

When classes are loaded

There are only two cases:

- When the new byte code is executed.
- When the byte code makes a static reference to a class. For example, System.out.

Static vs. Dynamic Class Loading

Classes are statically loaded with "new" operator. Dynamic class loading invokes the functions of a class loader at run time by using Class.forName() method.

Difference between loadClass() and Class.forName()

The loadClass() method loads only the class but does not initialize the object. While Class.forName() method initialize the object after loading it. For example, if you are using ClassLoader.loadClass() to load the JDBC driver, class loader does not allow to load JDBC driver.

The java.lang.Class.forName() method returns the Class Object coupled with the class or interfaces with the given string name. It throws ClassNotFoundException if the class is not found.

Example

In this example, java.lang.String class is loaded. It prints the class name, package name, and the names of all available methods of String class. We are using Class.forName() in the following example.

Class<?>: Represents a Class object which can be of any type (? is a wildcard). The Class type contains meta-information about a class. For example, type of String.class is Class<String>. Use Class<?> if the class being modeled is unknown.

getDeclaredMethod(): Returns an array containing Method objects reflecting all the declared methods of the class or interface represented by this Class object, including public, protected, default (package) access, and private methods, but excluding inherited methods.

getName(): It returns the method name represented by this Method object, as a String.

```
import java.lang.reflect.Method;
public class ClassForNameExample
{
public static void main(String[] args)
```

```
{
try
{
Class<?> cls = Class.forName("java.lang.String");
System.out.println("Class Name: " + cls.getName());
System.out.println("Package Name: " + cls.getPackage());
Method[] methods = cls.getDeclaredMethods();
System.out.println("-----Methods of String class ----------");
for (Method method : methods)
{
System.out.println(method.getName());
}
}
catch (ClassNotFoundException e)
{
e.printStackTrace();
}
}
}
```

Output

```
Class Name: java.lang.String
Package Name: package java.lang
-----Methods of String class ----------
value
coder
equals
length
toString
hashCode
getChars
------
------
------
intern
isLatin1
checkOffset
checkBoundsOffCount
checkBoundsBeginEnd
access$100
access$200
```

JEE or J2EE Design Patterns

J2EE design patterns are built for the developing the Enterprise Web-based Applications.

In J2EE , there are mainly three types of design patterns, which are further divided into their sub-parts:

1. Presentation Layer Design Pattern

1. Intercepting Filter Pattern
2. Front Controller Pattern
3. View Helper Pattern
4. Composite View Pattern

2. Business Layer Design Pattern

1. Business Delegate Pattern
2. Service Locator Pattern
3. Session Facade Pattern
4. Transfer Object Pattern

3. Integration Layer Design Pattern

1. Data Access Object Pattern
2. Web Service Broker Pattern

Context Object Strategies

Request Context Strategies

- Request Context Map Strategy
- Request Context POJO Strategy
- Request Context Validation Strategy

Configuration Context Strategies

- JSTL Configuration Strategy

Security Context Strategies

General Context Object Strategies

- Context Object Factory Strategy

  Context Object Auto

  population Strategy

Web Service Broker: Strategies

- Custom XML Messaging Strategy
- Java Binding Strategy
- JAX-RPC Strategy

Markup Language

A markup language is a computer language that uses tags to define elements within a document. It is human-readable, meaning markup files contain standard words, rather than typical programming syntax. While several markup languages exist, the two most popular are HTML and XML.

HTML is a markup language used for creating webpages. The contents of each webpage are defined by HTML tags. Basic page tags, such as <head>, <body>, and define sections of the page, while tags such as <table>, <form>, <image>, and <a> define elements within the page. Most elements require a beginning and end tag, with the content placed between the tags. For example, a link to the TechTerms.com home page may use the following HTML code:

TechTerms.com

XML is used for storing structured data, rather than formatting information on a page. While HTML documents use predefined tags (like the examples above), XML files use custom tags to define elements. For example, an XML file that stores information about computer models may include the following section:

```
<computer>
  <manufacturer>Dell</manufacturer>
  <model>XPS 17</model>
  <components>
    <processor>2.00 GHz Intel Core i7</processor>
    <ram>6GB</ram>
    <storage>1TB</storage>
  </components>
</computer>
```

XML is called the "Extensible Markup Language" since custom tags can be used to support a wide range of elements. Each XML file is saved in a standard text format, which makes it easy for software programs to parse or read the data. Therefore, XML is a common choice for exporting structured data and for sharing data between multiple programs.

NOTE: Since both HTML and XML files are saved in a plain text format, they can be viewed in a standard text editor. You can also view the HTML source of an open webpage by selecting the "View Source" option. This feature is found in the View menu of most Web browsers.

XML - eXtensible Markup Language (XML):

What is XML?

- XML stands for eXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to store and transport data
- XML was designed to be self-descriptive
- XML is a W3C Recommendation

The Difference Between XML and HTML

XML and HTML were designed with different goals:

- XML was designed to carry data - with focus on what data is
- HTML was designed to display data - with focus on how data looks
- XML tags are not predefined like HTML tags are

XML is Extensible

Most XML applications will work as expected even if new data is added (or removed).

Imagine an application designed to display the original version of note.xml (<to> <from> <heading> <body>).

Then imagine a newer version of note.xml with added <date> and <hour> elements, and a removed <heading>.

```
<note>
  <date>2015-09-01</date>
  <hour>08:30</hour>
  <to>Tove</to>
  <from>Jani</from>
  <body>Don't forget me this weekend!</body>
</note>
```

XML Simplifies Things

- It simplifies data sharing
- It simplifies data transport
- It simplifies platform changes
- It simplifies data availability

Many computer systems contain data in incompatible formats. Exchanging data between incompatible systems (or upgraded systems) is a time-consuming task for web developers. Large amounts of data must be converted, and incompatible data is often lost.

XML stores data in plain text format. This provides a software- and hardware-independent way of storing, transporting, and sharing data.

XML also makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

With XML, data can be available to all kinds of "reading machines" like people, computers, voice machines, news feeds, etc.

XML is a W3C Recommendation

XML became a W3C Recommendation as early as in February 1998.

XML Separates Data from Presentation

XML does not carry any information about how to be displayed.

The same XML data can be used in many different presentation scenarios.

Because of this, with XML, there is a full separation between data and presentation.

XML is Often a Complement to HTML

In many HTML applications, XML is used to store or transport data, while HTML is used to format and display the same data.

XML Separates Data from HTML

When displaying data in HTML, you should not have to edit the HTML file when the data changes.

With XML, the data can be stored in separate XML files.

With a few lines of JavaScript code, you can read an XML file and update the data content of any HTML page.

Books.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>

  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
```

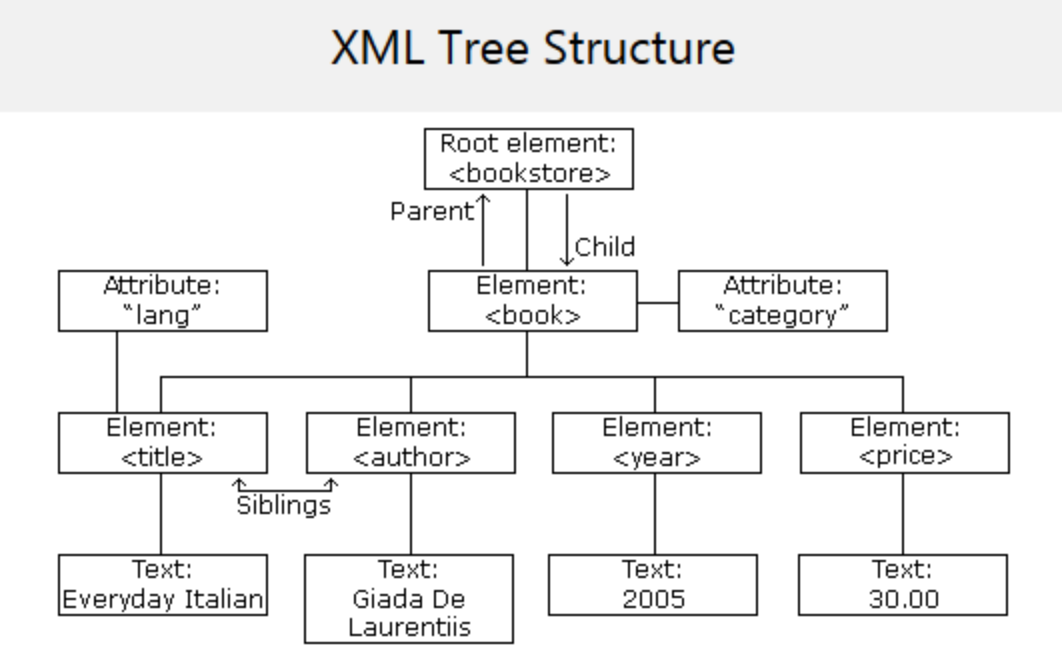
```
  <year>2005</year>
  <price>30.00</price>
 </book>

 <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">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</price>
 </book>

 <book category="web" cover="paperback">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
 </book>

</bookstore>
```



Empty XML Elements

An element with no content is said to be empty.

In XML, you can indicate an empty element like this:

<element></element>

You can also use a so called self-closing tag:

<element />

XML Attributes Must be Quoted

Attribute values must always be quoted. Either single or double quotes can be used.

For a person's gender, the <person> element can be written like this:

<person gender="female">

or like this:

<person gender='female'>

If the attribute value itself contains double quotes you can use single quotes, like in this example:

<gangster name='George "Shotgun" Ziegler'>

or you can use character entities:

<gangster name="George "Shotgun" Ziegler">

XML Namespaces

XML Namespaces provide a method to avoid element name conflicts.

Name Conflicts

In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications.

This XML carries HTML table information:

```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

This XML carries information about a table (a piece of furniture):

```
<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

If these XML fragments were added together, there would be a name conflict. Both contain a <table> element, but the elements have different content and meaning.

A user or an XML application will not know how to handle these differences.

<student>
<stud name=”t1.txt”>
<description> This is a text file
</description>

</stud>

<stud name=”t1.jpg”>
<description> This is a image File
</description>

</stud>
<stud name=”t1.doc”>
<description> This is a document file
</description>

</stud>
</student>

Document Type Definition (DTD)

What is a DTD?

A DTD is a Document Type Definition.

A DTD defines the structure and the legal elements and attributes of an XML document.

Why Use a DTD?

With a DTD, independent groups of people can agree on a standard DTD for interchanging data.

An application can use a DTD to verify that XML data is valid.

An Internal DTD Declaration

If the DTD is declared inside the XML file, it must be wrapped inside the <!DOCTYPE> definition:

XML document with an internal DTD

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend</body>
</note>
```

The DTD above is interpreted like this:

- !DOCTYPE note defines that the root element of this document is note
- !ELEMENT note defines that the note element must contain four elements: "to,from,heading,body"
- !ELEMENT to defines the to element to be of type "#PCDATA"
- !ELEMENT from defines the from element to be of type "#PCDATA"
- !ELEMENT heading defines the heading element to be of type "#PCDATA"
- !ELEMENT body defines the body element to be of type "#PCDATA"

An External DTD Declaration

If the DTD is declared in an external file, the <!DOCTYPE> definition must contain a reference to the DTD file:

XML document with a reference to an external DTD

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
```

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

And here is the file "note.dtd", which contains the DTD:

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

The Building Blocks of XML Documents

Seen from a DTD point of view, all XML documents are made up by the following building blocks:

- Elements
- Attributes
- Entities
- PCDATA
- CDATA

Elements

Elements are the main building blocks of both XML and HTML documents.

Examples of HTML elements are "body" and "table". Examples of XML elements could be "note" and "message". Elements can contain text, other elements, or be empty. Examples of empty HTML elements are "hr", "br" and "img".

Examples:

<body>some text</body>

<message>some text</message>

Attributes

Attributes provide extra information about elements.

Attributes are always placed inside the opening tag of an element. Attributes always come in name/value pairs. The following "img" element has additional information about a source file:

The name of the element is "img". The name of the attribute is "src". The value of the attribute is "computer.gif". Since the element itself is empty it is closed by a " /".

Entities

Some characters have a special meaning in XML, like the less than sign (<) that defines the start of an XML tag.

Most of you know the HTML entity: " ". This "no-breaking-space" entity is used in HTML to insert an extra space in a document. Entities are expanded when a document is parsed by an XML parser.

The following entities are predefined in XML:

Entity References	Character
<	<
>	>
&	&
"	"
'	'

PCDATA

PCDATA means parsed character data.

Think of character data as the text found between the start tag and the end tag of an XML element.

PCDATA is text that WILL be parsed by a parser. The text will be examined by the parser for entities and markup.

Tags inside the text will be treated as markup and entities will be expanded.

However, parsed character data should not contain any &, <, or > characters; these need to be represented by the & < and > entities, respectively.

CDATA

CDATA means character data.

CDATA is text that will NOT be parsed by a parser. Tags inside the text will NOT be treated as markup and entities will not be expanded.

Empty Elements

Empty elements are declared with the category keyword EMPTY:

<!ELEMENT element-name EMPTY>

Example:

<!ELEMENT br EMPTY>

XML example:

Elements with Parsed Character Data

Elements with only parsed character data are declared with #PCDATA inside parentheses:

<!ELEMENT element-name (#PCDATA)>

Example:

<!ELEMENT from (#PCDATA)>

Elements with any Contents

Elements declared with the category keyword ANY, can contain any combination of parsable data:

<!ELEMENT element-name ANY>

Example:

<!ELEMENT note ANY>

Elements with Children (sequences)

Elements with one or more children are declared with the name of the children elements inside parentheses:

<!ELEMENT element-name (child1)>
or
<!ELEMENT element-name (child1,child2,...)>

Example:

<!ELEMENT note (to,from,heading,body)>

An Internal Entity Declaration

Syntax

<!ENTITY entity-name "entity-value">

Example

DTD Example:

```
<!ENTITY writer "Donald Duck.">
<!ENTITY copyright "Copyright W3Schools.">
```

XML example:

```
<author>&writer;&copyright;</author>
```

Note: An entity has three parts: an ampersand (&), an entity name, and a semicolon (;).

An External Entity Declaration

Syntax

```
<!ENTITY entity-name SYSTEM "URI/URL">
```

Example

DTD Example:

```
<!ENTITY writer SYSTEM "https://www.w3schools.com/entities.dtd">
<!ENTITY copyright SYSTEM "https://www.w3schools.com/entities.dtd">
```

XML example:

```
<author>&writer;&copyright;</author>
```

XML Schema

What is an XML Schema?

An XML Schema describes the structure of an XML document.

The XML Schema language is also referred to as XML Schema Definition (XSD).

XSD Example

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<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>

</xs:schema>
```

The purpose of an XML Schema is to define the legal building blocks of an XML document:

- the elements and attributes that can appear in a document
- the number of (and order of) child elements
- data types for elements and attributes
- default and fixed values for elements and attributes

Why Learn XML Schema?

In the XML world, hundreds of standardized XML formats are in daily use.

Many of these XML standards are defined by XML Schemas.

XML Schema is an XML-based (and more powerful) alternative to DTD.

XML Schemas Support Data Types

One of the greatest strength of XML Schemas is the support for data types.

- It is easier to describe allowable document content

- It is easier to validate the correctness of data
- It is easier to define data facets (restrictions on data)
- It is easier to define data patterns (data formats)
- It is easier to convert data between different data types

XML Schemas use XML Syntax

Another great strength about XML Schemas is that they are written in XML.

- You don't have to learn a new language
- You can use your XML editor to edit your Schema files
- You can use your XML parser to parse your Schema files
- You can manipulate your Schema with the XML DOM
- You can transform your Schema with XSLT

XML Schemas are extensible, because they are written in XML.

With an extensible Schema definition you can:

- Reuse your Schema in other Schemas
- Create your own data types derived from the standard types
- Reference multiple schemas in the same document

XML Schemas Secure Data Communication

When sending data from a sender to a receiver, it is essential that both parts have the same "expectations" about the content.

With XML Schemas, the sender can describe the data in a way that the receiver will understand.

A date like: "03-11-2004" will, in some countries, be interpreted as 3.November and in other countries as 11.March.

However, an XML element with a data type like this:

<date type="date">2004-03-11</date>

ensures a mutual understanding of the content, because the XML data type "date" requires the format "YYYY-MM-DD".

Well-Formed is Not Enough

A well-formed XML document is a document that conforms to the XML syntax rules, like:

- it must begin with the XML declaration
- it must have one unique root element
- start-tags must have matching end-tags
- elements are case sensitive
- all elements must be closed
- all elements must be properly nested
- all attribute values must be quoted
- entities must be used for special characters

An XML Schema

The following example is an XML Schema file called "note.xsd" that defines the elements of the XML document above ("note.xml"):

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="https://www.w3schools.com"
xmlns="https://www.w3schools.com"
elementFormDefault="qualified">

<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>
```

</xs:schema>

The <schema> Element

The <schema> element is the root element of every XML Schema:

```
<?xml version="1.0"?>

<xs:schema>
...
...
</xs:schema>
```

Referencing a Schema in an XML Document

```
<?xml version="1.0"?>

<note xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="note.xsd">

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

What is a Simple Element?

A simple element is an XML element that can contain only text. It cannot contain any other elements or attributes.

However, the "only text" restriction is quite misleading. The text can be of many different types. It can be one of the types included in the XML Schema definition (boolean, string, date, etc.), or it can be a custom type that you can define yourself.

You can also add restrictions (facets) to a data type in order to limit its content, or you can require the data to match a specific pattern.

Defining a Simple Element

The syntax for defining a simple element is:

<xs:element name="xxx" type="yyy"/>

where xxx is the name of the element and yyy is the data type of the element.

XML Schema has a lot of built-in data types. The most common types are:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

Example

Here are some XML elements:

<lastname>Refsnes</lastname>
<age>36</age>
<dateborn>1970-03-27</dateborn>

And here are the corresponding simple element definitions:

<xs:element name="lastname" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="dateborn" type="xs:date"/>

Default and Fixed Values for Simple Elements

Simple elements may have a default value OR a fixed value specified.

A default value is automatically assigned to the element when no other value is specified.

In the following example the default value is "red":

<xs:element name="color" type="xs:string" default="red"/>

A fixed value is also automatically assigned to the element, and you cannot specify another value.

In the following example the fixed value is "red":

<xs:element name="color" type="xs:string" fixed="red"/>

XSLT Elements

The XSLT elements from the W3C Recommendation (XSLT Version 1.0)

The Extensible Stylesheet Language Family (XSL)

XSL is a family of recommendations for defining XML document transformation and presentation. It consists of three parts:

XSL Transformations (XSLT)
: a language for transforming XML;

The XML Path Language (XPath)
: an expression language used by XSLT (and many other languages) to access or refer to parts of an XML document;

XSL Formatting Objects (XSL-FO)
: an XML vocabulary for specifying formatting semantics.

An XSLT stylesheet specifies the presentation of a class of XML documents by describing how an instance of the class is transformed into an XML document that uses a formatting vocabulary, such as (X)HTML or XSL-FO. For a more detailed explanation of how XSL works, see the What Is XSL page.

XSLT is developed by the W3C XSLT Working Group (members only) whose charter is to develop the next version of XSLT. XSLT is part of W3C's XML Activity, whose work is described in the XML Activity Statement.

XPath is developed jointly by the XQuery and XSLT Working Groups.

The XSL-FO work at W3C was taken over by the XML Print and Page Layout Working Group which has now been closed.

```
<?xml version="1.0"?>

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

<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/cd">
        <tr>
          <td><xsl:value-of select="title"/></td>
          <td><xsl:value-of select="artist"/></td>
        </tr>
      </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="cdcatalog.xsl"?><bookstore>
<book>
  <title>Harry Potter</title>
  <artist>dsfgag</artist>
</book>

<book>
  <title>Learning XML</title>
```

```
  <artist>dsfgagaaa</artist>
</book>

</bookstore>
```