

XML

<subtitle>Java and XML 1</subtitle>

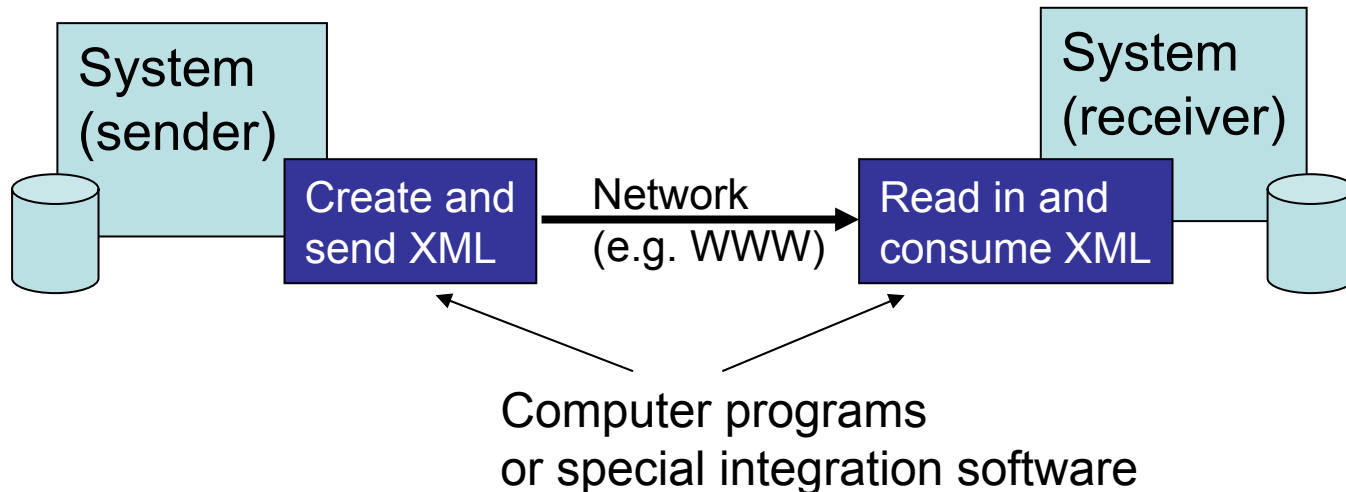
<author>Prof. Dr. Christian Pape</author>

Content

- Overview
- Java XML for Data Binding
- Ant

Overview

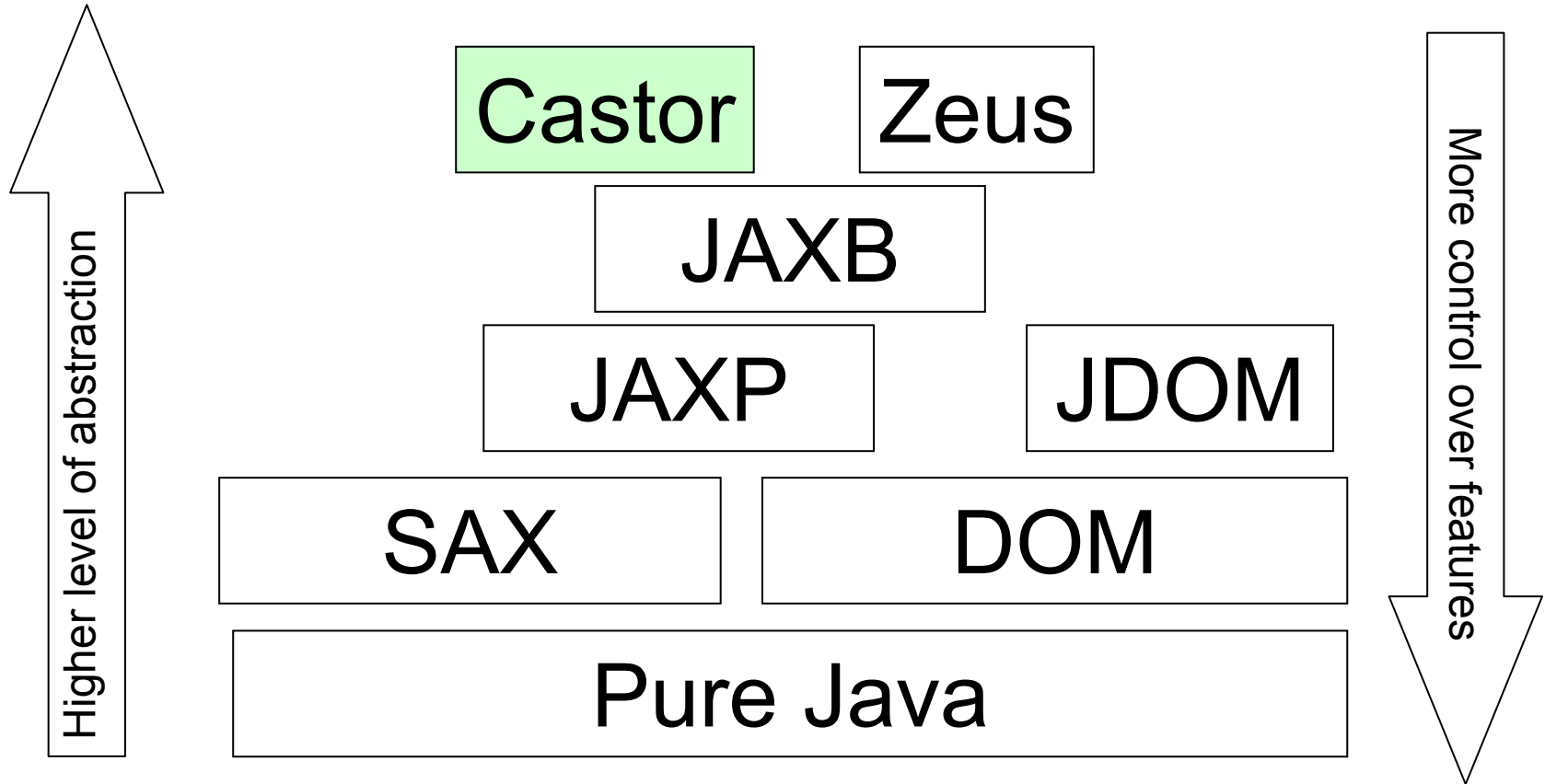
- XML is not for human reading only
- Main purpose
 - Electronic exchange of documents and data
 - XML must be written by one system, send to another system
 - Second system reads in XML and consumes data
- In several situations
 - Programming is needed to extract data, create an XML document, and vice versa.



Overview

- No need to start from scratch
 - There quite a lot XML standards for Java
- Low level APIs
 - Simple API for XML (SAX)
 - Document Object Model (DOM) from W3C
 - JDOM from Sun
 - Java API for XML Processing (JAXP)
- High level APIs
 - Java API for XML Data Binding (JAXB)
 - Frameworks for JAXB: Castor, Zeus, other

Overview / Some Java APIs and a few Products



Java XML for Data Binding

- Recall relation between UML class diagram and XML Schema (or DTD)
- UML covers enough information to
 - specify and generate an XML Schema
 - generate Java classes
- Why not bring both together
 - Associate complex-type with Java classes and simple-types with Java Attributes
 - Associate Java Objects with XML document instances or parts of XML document instances
- Advantages
 - No knowledge needed for XML parsers and low level APIs
 - Java classes can directly be used in computer program to write or consume XML data
- Disadvantages
 - Could be slower than direct manipulation
 - Less control over XML features

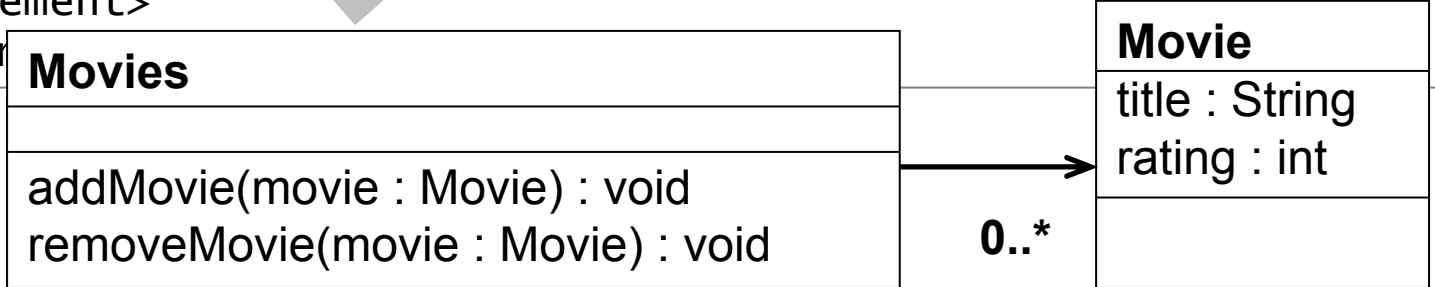
Java XML for Data Binding / Basic Idea

```

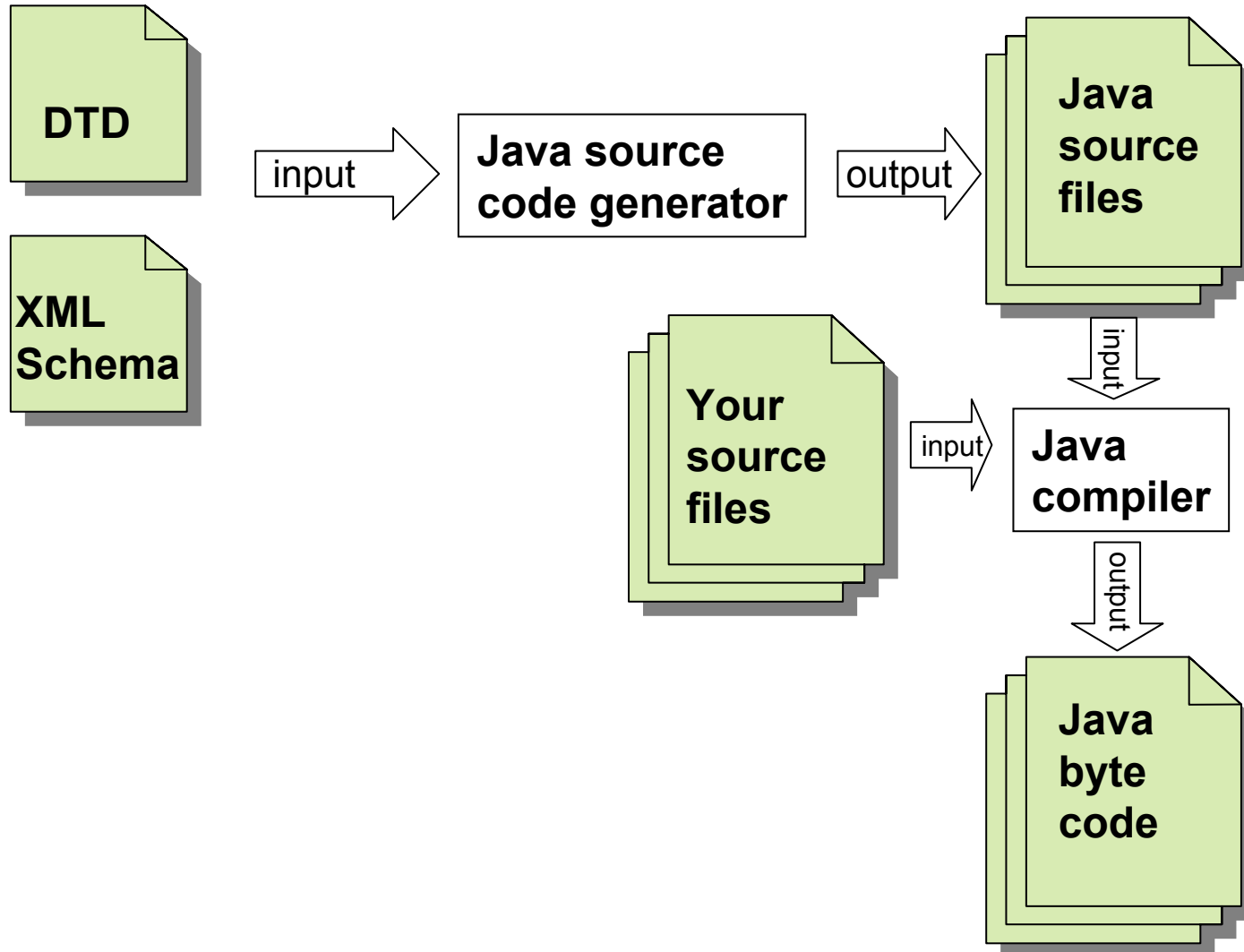
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="movie-type">
    <xsd:sequence>
      <xsd:element name="title" type="xsd:string"/>
      <xsd:element name="rating" type="xsd:int"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="movies">
    <xsd:complexType base="movie-type">
      <xsd:sequence>
        <xsd:element name="movie" type="movie-type"
          minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

```

**Create classes
from XML model**



Java XML for Data Binding / Basic Idea



Java XML for Data Binding / Basic Idea

```

<movies>
  <movie>
    <title>Die Hard</title>
    <rating>2</rating>
  </movie>
</movies>

```

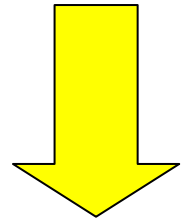
```

Movies movies = ... // read in XML document instance
Movie movie = ... // create new Java instance Movie

movie.setTitle("Time Bandits");
movie.setRating(3);
movies.addMovie(movie);
movies. ... / write back to file with new movie

```

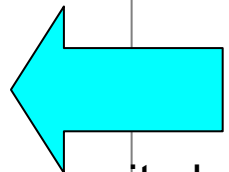
read in
(unmarshal)



movies : Movies

: Movie
title = "Die Hard"
rating = 2

: Movie
title = "Time Bandits"
rating = 3



write back
(marshal)

```

<title>Die Hard</title>
<rating>2</rating>
</movie>
<movie>
  <title>Time
Bandits</title>
  <rating>3</rating>
</movie>
</movies>

```

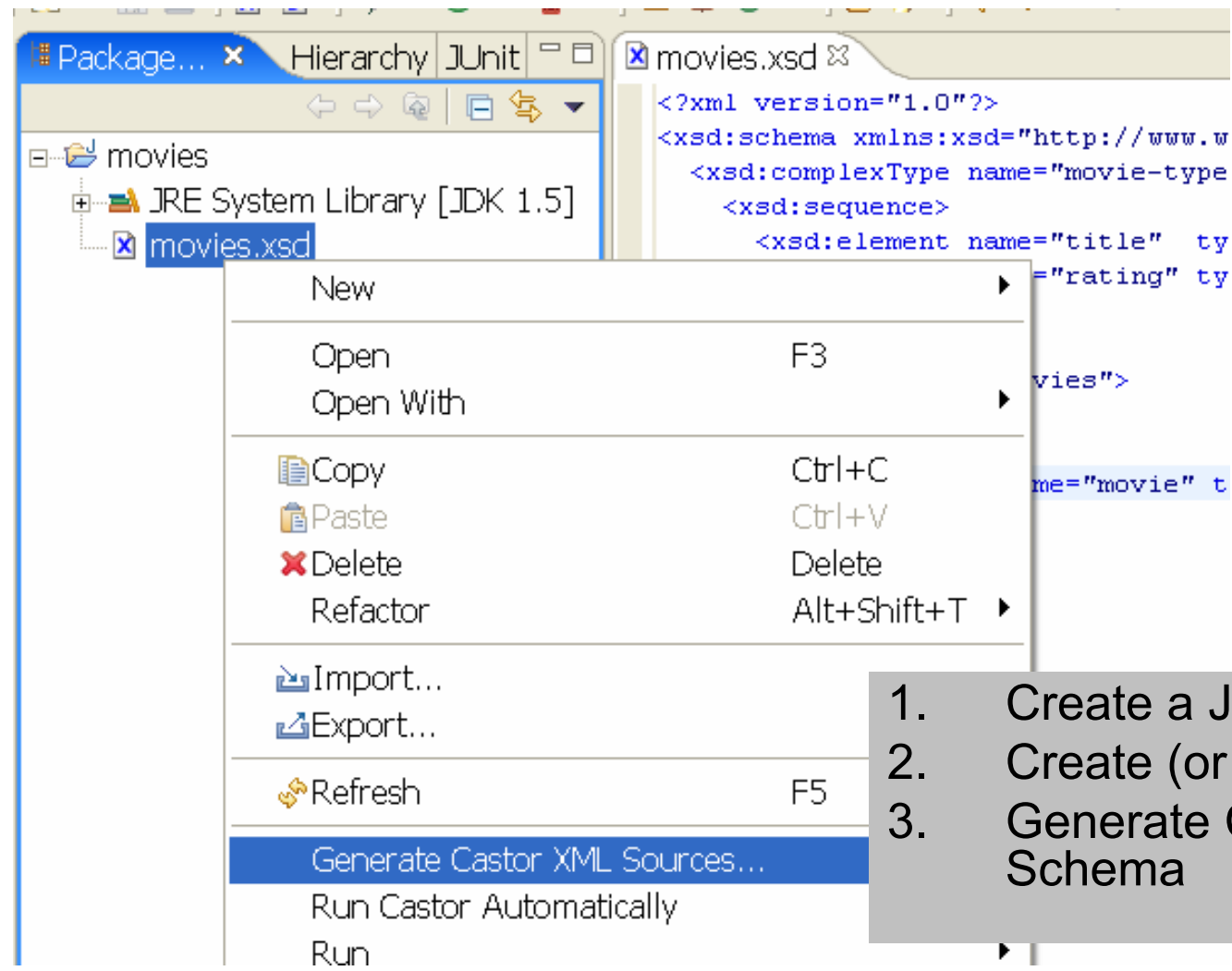
Java XML for Data Binding / Basic Idea

1. Conversion from XML to Java
 - Unmarshalling
 - Validating of XML document in background
 - Creating of necessary objects done in background
2. Manipulate Java Objects
3. Conversion from Java to XML
 - Marshalling
 - XML created from object tree

Castor

- Castor
 - Popular open source data binding framework
 - Needs XML Schemas
 - <http://castor.exolab.org>
 - Covers more than XML data binding and JAXB
 - Eclipse plugin exists
<http://xdoclipse.sourceforge.net>
 - Plugin comes with all necessary files

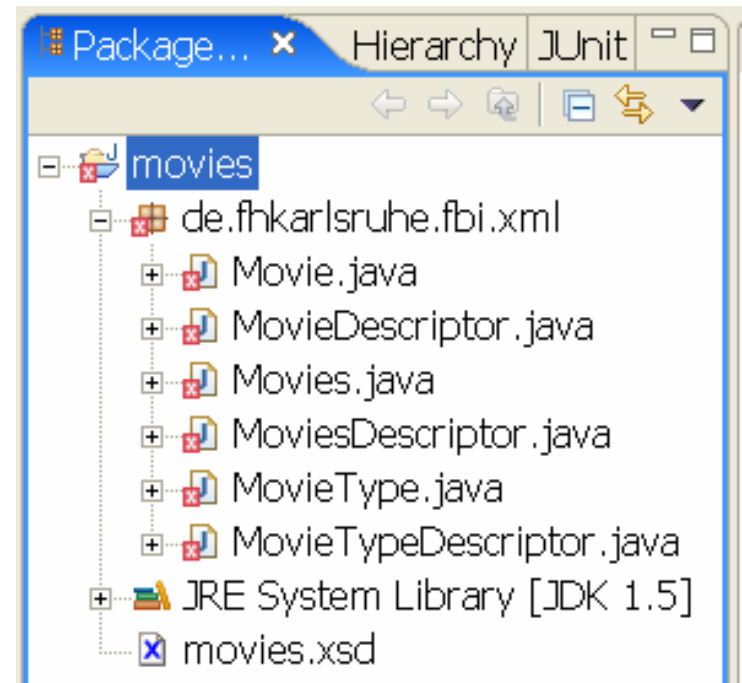
Castor / With Eclipse Plugin



1. Create a Java Project
2. Create (or load) a XML Schema
3. Generate Castor classes from Schema

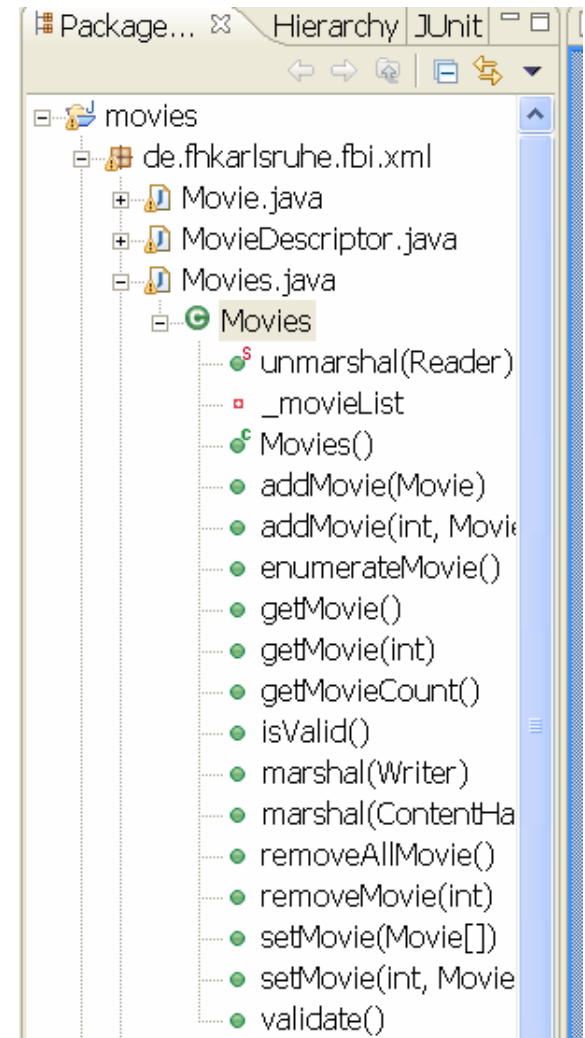
Castor / With Eclipse Plugin

- Two classes for each non-anonymous complex-type
 - MovieType.java
 - MovieDescriptor.java
- Two classes for each element
 - Movies.java
 - MoviesDescriptor.java
 - Movie.java
 - MovieDescriptor.java
- Some compile errors:
 - castor.jar is missing, add it to project
 - For running your code, add an XML parser like Xerces



Castor / Simple Example

- **Movies, Movie**
 - The derived Java class for manipulation
- **Contain**
 - Instance methods for accessing derived attributes and relations
 - Class methods for (un)marshalling



Caster / Simple Example

```
public class MovieTest {  
    public static void main(String argv[])  
        throws Marshallexception, ValidationException {  
        Movies movies = new Movies();  
        Movie movie = new Movie();  
        movie.setTitle("Die Hard");  
        movie.setRating(2);  
        movies.addMovie(movie);  
        movies.marshal(new OutputStreamWriter(System.out));  
    }  
}
```

```
Problems Javadoc Declaration Outline Console x  
<terminated> MovieTest [Java Application] C:\Programme\Java\jdk1.5.0\bin\javaw.  
<?xml version="1.0" encoding="UTF-8"?>  
<movies><movie><title>Die Hard</title><rating>2</rating></movie></movies>
```

Castor / Simple Example

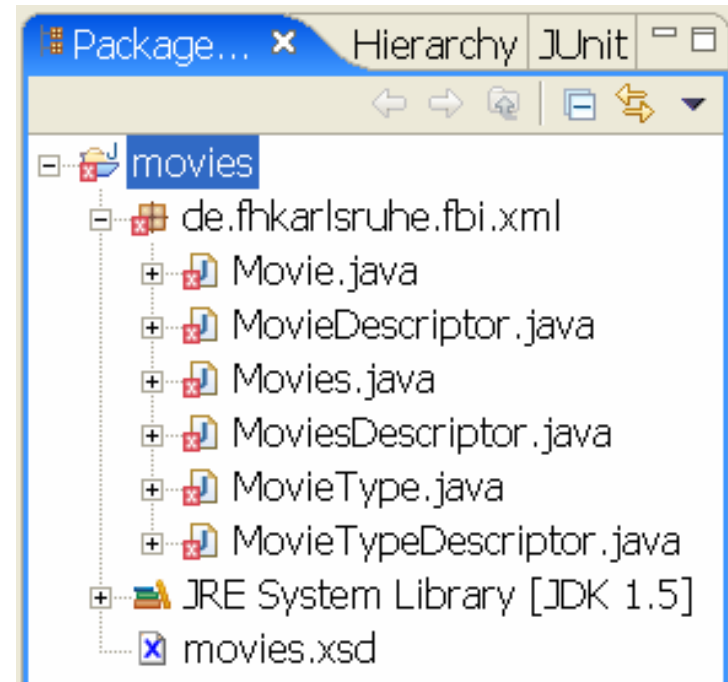
- **ValidationException**
 - Is thrown during marshalling when the Java Object model is not valid with respect to the restrictions of the XML Schema
- **Examples:**
 - At least one movie must be attached to a Movies instance “ValidationException: A minimum of 1 movie object(s) are required.”
 - Title and rating are mandatory (NULL not allowed): “ValidationException: title is a required field.”
- **Validate Java objects before writing**

Castor / Simple Example

```
public static void main(String argv[])
    throws MarshalException, ValidationException {
    Movies movies = new Movies();
    // ...
    movies.addMovie(movie);
    try {
        movies.validate();
    } catch (ValidatingException e) {
        // try to recover the data
    }
    movies.marshal(new OutputStreamWriter(System.out));
}
```

Castor

- Why Descriptor classes?
 - They carry information used by castor for mapping Java to XML
 - Element name (differs from class name), Namespaces, etc.
 - Options to eliminate them
- Why Type classes?
 - Castor uses inheritance for extended types and restricting types
 - A type can be used by more than one element



Castor / Data Binding

- Advantages
 - Easy to use
 - Fast results
 - Limitations of XML to Java Mapping
- Disadvantages
 - If XML changes often, the classes change, too
 - Too slow for large quantities of XML data
 - Memory consuming high
 - May be too slow if only a small part of XML data changes

Castor / Data binding

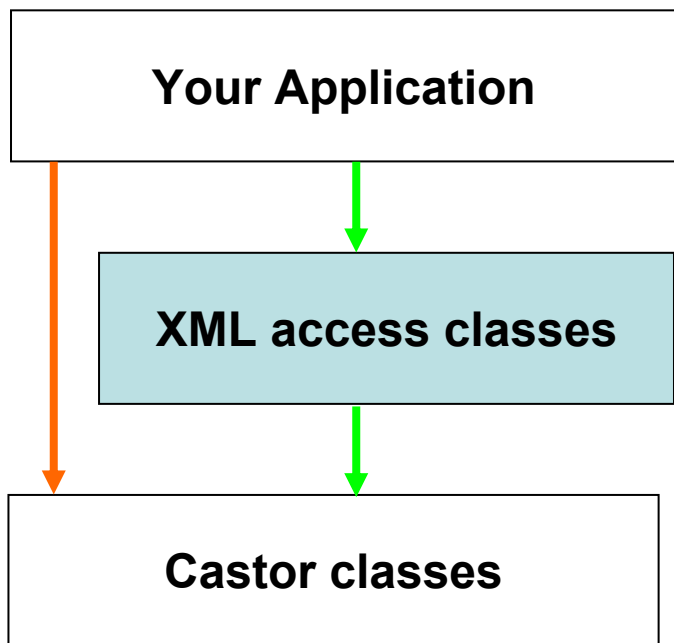
- When to use data binding?
 - For configurations files (small, read in memory once)
 - System integration: for small quantities of XML files
 - If no (or not enough) knowledge about SAX, DOM, JAXP etc. is on hand
 - Prototyping

Data binding / Application Design

- Designing software with data binding
- Eliminate direct references to castor classes
 - Information hiding
 - Castor classes may change too often, or may be replaced by other data binding framework
 - Store automatically generated classes in own package (de.fhkarlsruhe.xml.**castor**)
 - Provide additional flexibility for accessor methods, for instance, checking validity of content when set-Method is invoked

Data binding / Application Design

- Castor classes may change too often
 - Use **strict layering**
 - Introduce new classes that hide access to castor classes
 - Pack all classes of one layer in unique Java package (or C++, C# namespace)
- Main advantage
 - Application is independent of underlining product (Castor or other)



Layering

upper layer of classes only use classes of lower layers and not vice versa

Strict layering

upper layer of classes only use classes of *next* lower layer

Data binding / Application Design

- Layering by inheritance
 - Create new class Movie by extending existing Castor classes
 - Methods can be overridden
 - All castor methods still accessible (could be an disadvantage)
- Layering by wrapping
 - Create new class Movie with relation to existing Castor class
 - Methods must be implemented by delegating to existing Castor instance
 - Only needed methods are accessible (less dependencies)
- Layering by interfaces
 - Castor classes must implement interface (possible?)
 - Interfaces are used in upper layers
- Or combinations of all three variants
- Eliminate product depended exceptions as well!
- Layer A on top of layer B
 - all classes of B can be compiled without any class of B
 - Strict: all classes of A can be compiled with classes of B without any classes of layers below B

Data Binding / Application Design

de.fhkarlsruhe.fbi.application

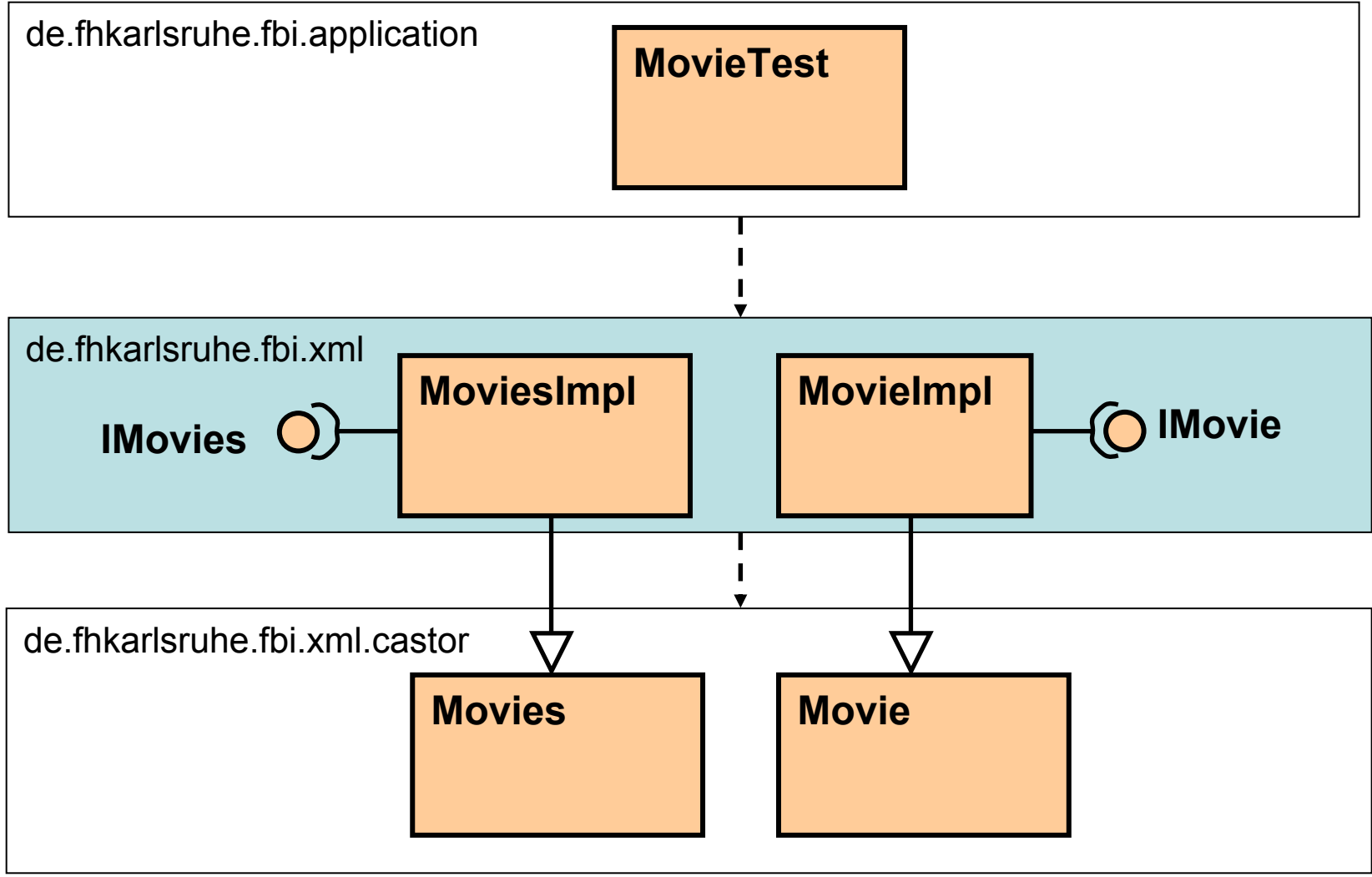
Only IMovie, IMovies and constructor of MovieImpl/MoviesImpl are used here

de.fhkarlsruhe.fbi.xml

Interfaces IMovie / IMovies go here.
New XmlException.
MovieImpl, MoviesImpl extend Castor classes and implement interfaces

de.fhkarlsruhe.fbi.xml.castor

Data Binding / Application Design



Castor / “Application”

```
public class MovieTest {  
  
    public static void main(String argv[]) {  
        IMovies movies = new MoviesImpl();  
        IMovie movie = new MovieImpl();  
        movie.setTitle("Die Hard");  
        movie.setRating(IMovie.RATING_AVERAGE);  
        movies.addMovie(movie);  
        try {  
            movies.check();  
            movies.marshal(new OutputStreamWriter(System.out));  
        } catch ( XmlException e ) {  
            System.out.println("validation failed.");  
            System.out.println(e);  
        }  
    }  
}
```

Castor / IMovie

```
public interface IMovie {  
  
    public static final int RATING_GOOD = 1;  
    public static final int RATING_AVERAGE = 2;  
    public static final int RATING_BAD = 3;  
  
    public String getTitle();  
  
    public void setTitle(String title);  
  
    public int getRating();  
  
    public void setRating(int Rating);  
}
```

Castor / Movies

```
public interface IMovies {  
  
    public void addMovie(IMovie movie);  
  
    // castor only provides removeMovie(int)  
    public void removeMovie(IMovie movie);  
  
    public void check()  
        throws XmlException;  
  
    public void marshal(OutputStreamWriter writer)  
        throws XmlException;  
}
```

Castor / Movies

```
public interface IMovies {  
  
    public void addMovie(IMovie movie);  
  
    // castor only provides removeMovie(int)  
    public void removeMovie(IMovie movie);  
  
    public void check()  
        throws XmlException;  
  
    public void marshal(OutputStreamWriter writer)  
        throws XmlException;  
}
```

Castor / Movies

```
public class MoviesImpl extends Movies
    implements IMovies{

    public void removeMovie(IMovie movie) {
        if (movie instanceof Movie) {
            for (int i=0; i < getMovieCount(); i++) {
                if ( super.getMovie(i).equals( (Movie) movie)) {
                    super.removeMovie(i);
                    break;
                }
            }
        }
    }

    // other methods omitted
}
```

Data binding / Application Design

- No direct dependencies
 - lower layers could be exchange with total different technologies (data base access instead XML) without much affection to application
- Wrapping/Inheritance
 - Allow more convenient methods: addMovie(Movie) instead addMovie(int)
- Costs
 - Additional programming (little bit)
- Costs pay off very soon
 - for mid-range applications, cause changes caused by may dependencies are expensive
 - for long-term application, cause technology changes
 - when using unknown technology (not familiar with Castor or XML at all)

Content

- Overview
- Java XML for Data Binding
- **Ant**

Ant

- Ant
 - Has nothing to do with accessing XML from java
 - Is an **automatic build tool** (similar too Unix make)
 - Ant's syntax is in XML
 - It is a de facto standard for Java
 - It is written itself in Java, therefore portable (unlike Unix make)

<http://ant.apache.org>

[Apache](#) > [Ant.apache](#)



[Home](#)

[Projects](#)

- ◆ Apache Ant
 - [Welcome](#)
 - [License](#)
 - [News](#)
- ◆ Documentation
 - [Manual](#)
 - [Related Projects](#)
 - [External Tools and Tasks](#)
 - [Resources](#)
 - [Frequently Asked Questions](#)
 - [Wiki](#)
 - [Having Problems?](#)
- ◆ Download
 - [Binary Distributions](#)
 - [Source Distributions](#)
- ◆ Contributing
 - [Mailing Lists](#)
 - [CVS Repositories](#)
 - [Bug Database](#)
 - [Enhancement Requests](#)

Welcome

Ant 1.6.2

July 16, 2004 - Ant 1.6.2 Available

Apache Ant 1.6.2 is now available for [download](#).

Nested elements for namespaced tasks and types may belong

All exceptions thrown by tasks are now wrapped in a buildexc

Ant 1.6.2 fixes a large number of bugs and adds a number of f

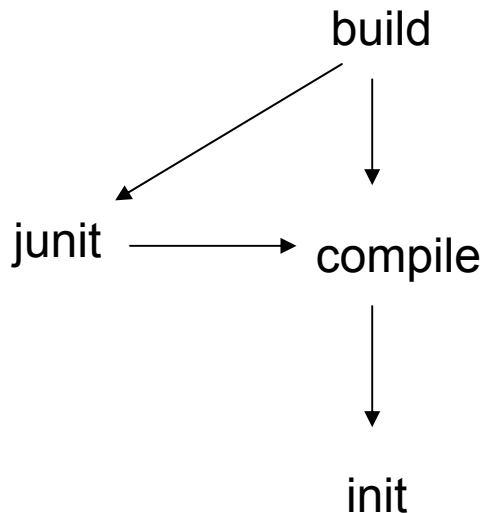
Apache Ant

Ant

- What is a build tool?
 - It helps developers to compile source files and build final software builds that can be deployed (installed) and executed
 - Software builds can be complex:
 - Control of software version
 - Several files including binary, configuration files, release-notes must be bundled
 - Testing of final build should be automatized
 - Dependencies between steps
- Eclipse (and all other Java IDEs) are build tools
 - But not automatic: Programmer controls most of the steps involved to create a final build (checking out new version, compiling, create deployable files, run automatic tests, etc)

Ant

- Ant checks dependencies between task
 - Checks based on time stamp of files and ant task used



build: create Java Archive with jar

Only executed if one of the class files is newer than last Java archive and the unit tests have been run

Compile: Create class files with javac

Only executed if one of the source files is newer than its class file

Ant

```
<?xml version="1.0" encoding="UTF-8"?>
<project default="compile" basedir=". ">

  <property name="src.dir" value="${basedir}"/>
  <property name="build.dir" value="${basedir}/build"/>
  <property name="lib.dir" value="${basedir}/lib"/>

  <target name="compile">
    <javac
      srcdir="${src.dir}"
      includes="**/*.java"
      destdir="${build.dir}">
      <classpath>
        <fileset dir="${lib.dir}">
          <include name="*.jar"/>
        </fileset>
      </classpath>
    </javac>
  </target>
</project>
```

- Root: project
- Target: defines one task, e.g. for compiling, initialization, etc.

Ant

```
<?xml version="1.0" encoding="UTF-8"?>  
<project default="build" basedir=".">
```

- default
 - when starting ant without explicit target name, then the target “build” is executed
- basedir
 - defines the directory with respect to all relative file/path names are defined
 - Unix like syntax: “.” for current directory

Ant

```
<property name="src.dir" value="${basedir}"/>
<property name="build.dir" value="${basedir}/build"/>
<property name="lib.dir" value="${basedir}/lib"/>
```

- property
 - Defines a reference to a value (something like a “variable”)
 - Makes it easy to rearrange overall structure of project (source file location, where to put class files, etc)
- attributes:
 - Name: defines the properties name
 - Value: the string value of this property
- Properties values can be used in other string values with \$ followed by the property name enclosed in { }
- Here:
 - Java source files are in the current directory
 - We want to generate class files into the directory “build”
 - Libraries necessary for compilation (*.jar files) are located in a “lib” directory

Ant

- target
 - defines a task that can be executed
 - Mandatory attribute **name** for unique target name
- javac
 - Ant build in command for compiling java sources
 - Attributes use here:
 - srcdir: specifies where the Java source files are located
 - includes: specifies which sources to be included (** matches every directory and subdirectory of \${src.dir}, *.java matches each Java source file)
 - builddir: specifies where the generated class files should be stored
 - Element used in javac-Content
 - classpath: specifies all additional libraries that have to be included to compile the sources, e.g. castor.jar; all libraries should be placed in the lib directory

```
<target name="compile">
  <javac
    srcdir="${src.dir}"
    includes="**/*.java"
    destdir="${build.dir}">
    <classpath>
      <fileset dir="${lib.dir}">
        <include name="*.jar"/>
      </fileset>
    </classpath>
  </javac>
</target>
```

Ant

- `${dest.dir}` may not exist, when executing ant-script for first time
 - New target “init”, that create all necessary directories or other stuff
 - Before executing “compile”, the new “init” target should be executed
 - “compile” now depends on “init”

```
<target name="compile" depends="init">
  <javac ...</javac>
</target>

<target name="init">
  <mkdir dir="${build.dir}"/>
</target>
```

- **mkdir** creates directory with **name** given by attribute “dir”

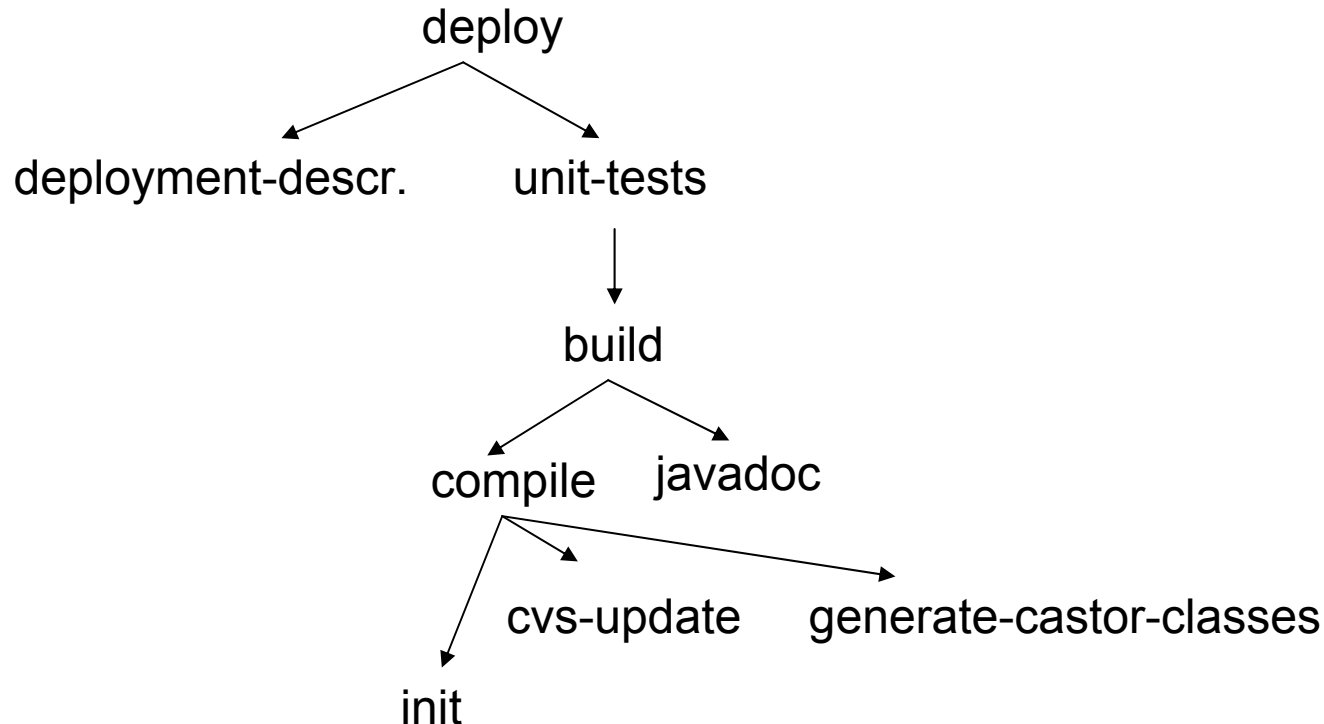
Ant

- Compiled class files (and other resources) should be packed together in one Java Archive (jar-file)
- New target “build”, depending on “compile”
- jar: creates an Java archive named with value of attribute “destfile”, with all files and directories given by the value of “basedir”

```
<target name="build" depends="compile">  
  <jar  
    destfile="movies.jar"  
    basedir="${build.dir}"  
  />  
</target>
```

Ant

- Large and mid-size projects contain a lot of tasks and dependencies
- Disadvantage of manually building software
 - to much time (may take 30 minutes or more to execute each step by hand)
 - humans often forget dependencies and command syntax



Ant

- A lot of build in commands exists
 - Managing files and directories
 - Retrieving source from software configuration systems (E.g. CVS)
 - Executing XSLT scripts
 - Executing JUnit tests
 - And more
- If no command one exists
 - Define your own command with taskdef
 - Program your own command
 - Execute any program

Ant

- Execute any program with „exec“
 - simulates input in command line
- Example
 - Generating Java classes from CORBA specification file “Hello.idl” with IDL to Java compiler

```
<target name="compile-idl">  
  <exec executable="idlj">  
    <arg value="-fall"/>  
    <arg value="-oldImplBase"/>  
    <arg value="Hello.idl"/>  
  </exec>  
</target>
```



```
Idlj -fall -oldImplBase Hello.idl
```

Ant

- Other build tools: Unix make

Ant	Make
Restricted to Java	General purpose
Platform independent (Ant is written in Java)	Unix (or Unix-like extensions for other OS)
De facto standard for Java	Standard on Unix systems
XML syntax	Own (simple) syntax
Fixed set of targets, but extensible	Integrates with every other unix command
Integrated with most Java IDEs	Start from command line