<?xml?>

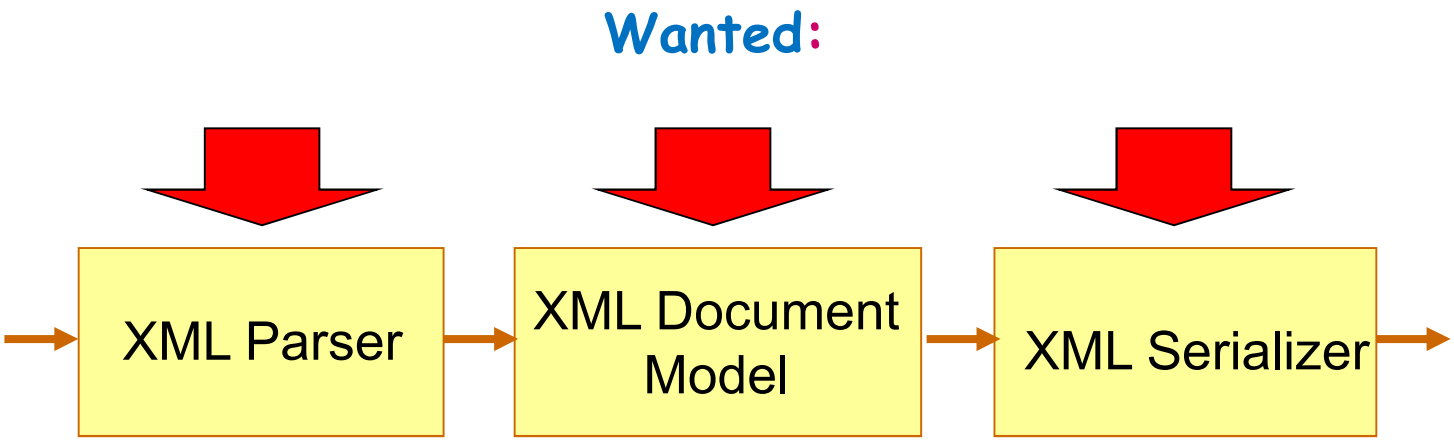
XML – Part B

Processing and Storing

Reading, Creating and Storing XML Documents

Manipulating XML Documents by programs means

- reading and parsing the document
- creating objects „containing“ the nodes of the documents (object tree)
- creating, modifying, deleting, moving document objects
- serializing the object tree into an XML text document



Part 3 – JAXB (Java Architecture for XML Binding)

Outline

- Introduction
- Marshal and Unmarshal
- Annotating classes
- Generating classes from XSD

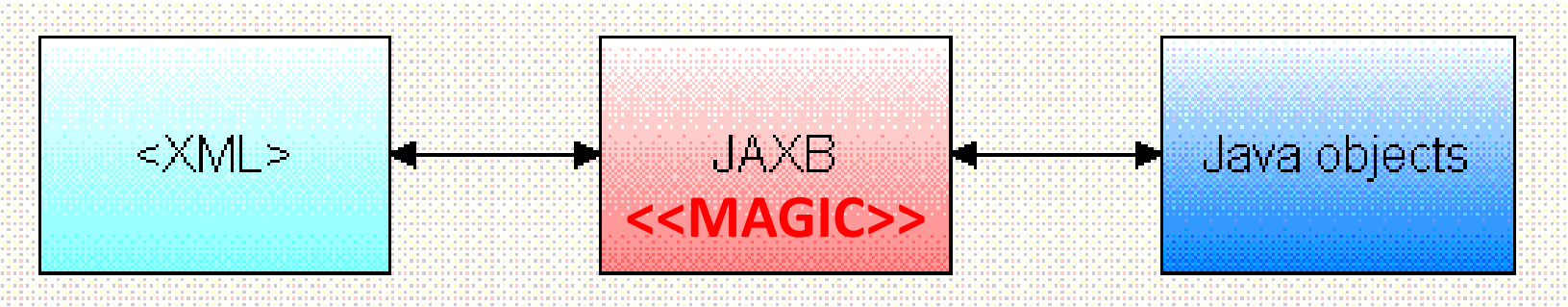
Outline

- Introduction
- Marshal and Unmarshal
- Annotating classes
- Generating classes from XSD

JAXB (Java Architecture for XML Binding)

JAXB is a Java standard that defines how Java objects are converted to/from XML (specified using a standard set of mappings.

- JAXB defines an API for **reading** and **writing** Java objects to/from XML documents
- JAXB offers methods which makes reading and writing of XML files very easy.



JAXB (Java Architecture for XML Binding)

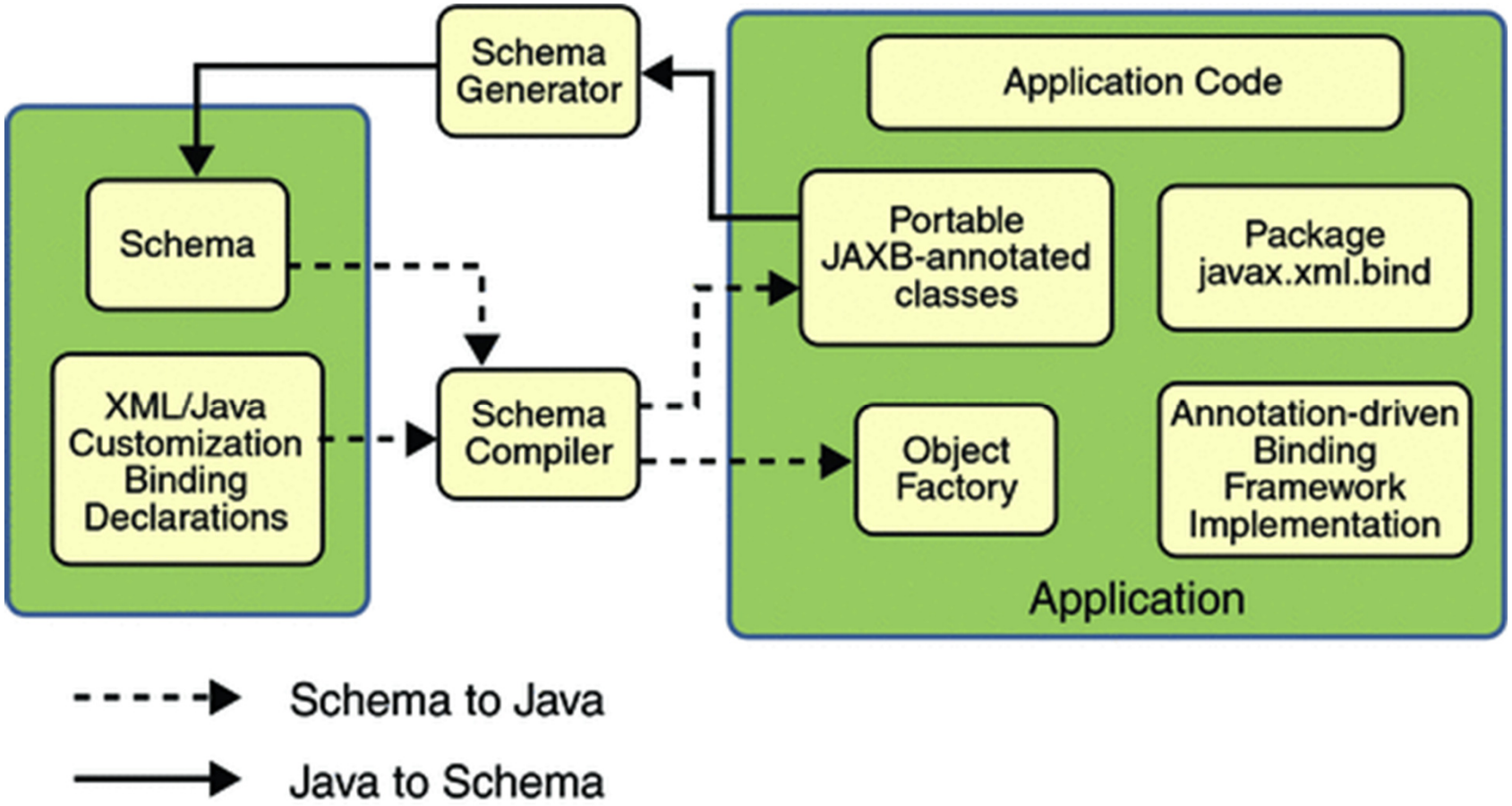
- History
  - JAXB has its origin in the Java Enterprise Edition (JEE)
    - Version 1: JSR 31 2003
    - Version 2: JSR 222 2006
    - Version 2.2 JSR 222 2008 (see jre 1.8)
    - Version 2.3 JSR 222 2017 (see jre 9)

    - JSR: Java Specification Request

JAXB (Java Architecture for XML Binding)

- JAXB consist of an API, which is defined in package javax.xml.binding
- To use JAXB you need a provider (implementation of JAXB)
  - Reference implementation (RI)
    - JAXP-RI by Metro-Project https://javaee.github.io/jaxb-v2/
  - Alternative provider
    - EclipseLink Moxy http://www.eclipse.org/eclipselink/#moxy
- The API and the RI are included in the Java Standard Edition (JSE)
  - jre 1.8 → JAXB 2.2.8
  - jre 9 → JAXB 2.3
    - However: javax.xml.bind is marked as deprecated in JSE 9!
      Will probably be moved somewhere else in Java 10

JAXB (Java Architecture for XML Binding)



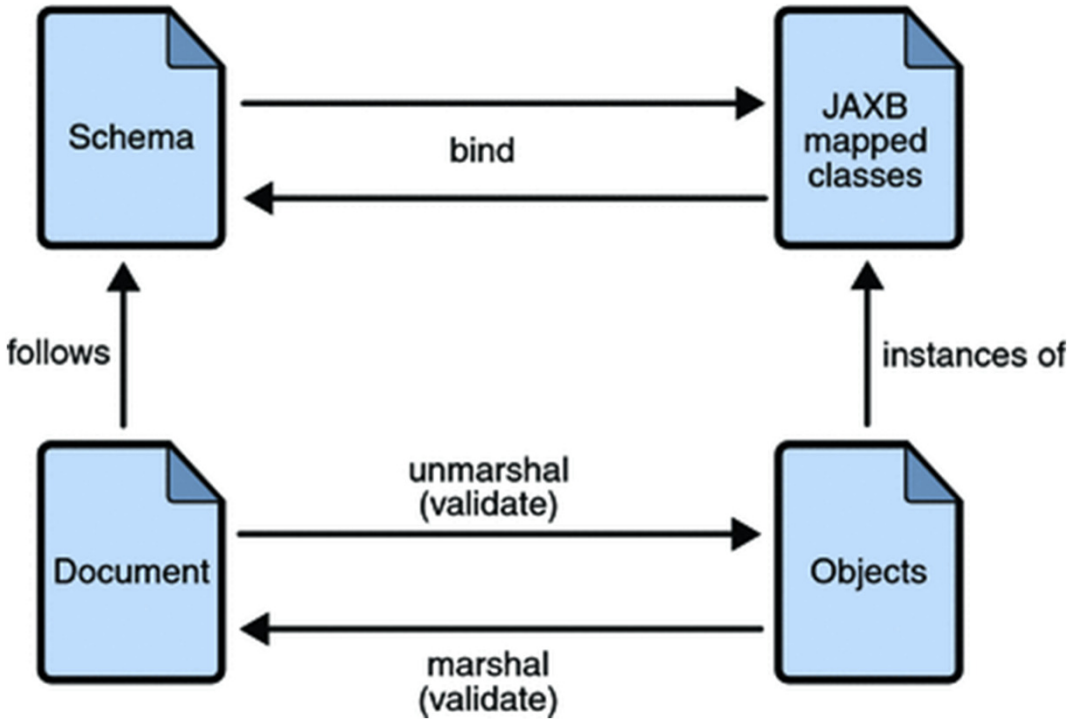
More details: https://docs.oracle.com/javase/tutorial/jaxb/intro/

Outline

- Introduction
- Marshal and Unmarshal
- Annotating classes
- Generating classes from XSD

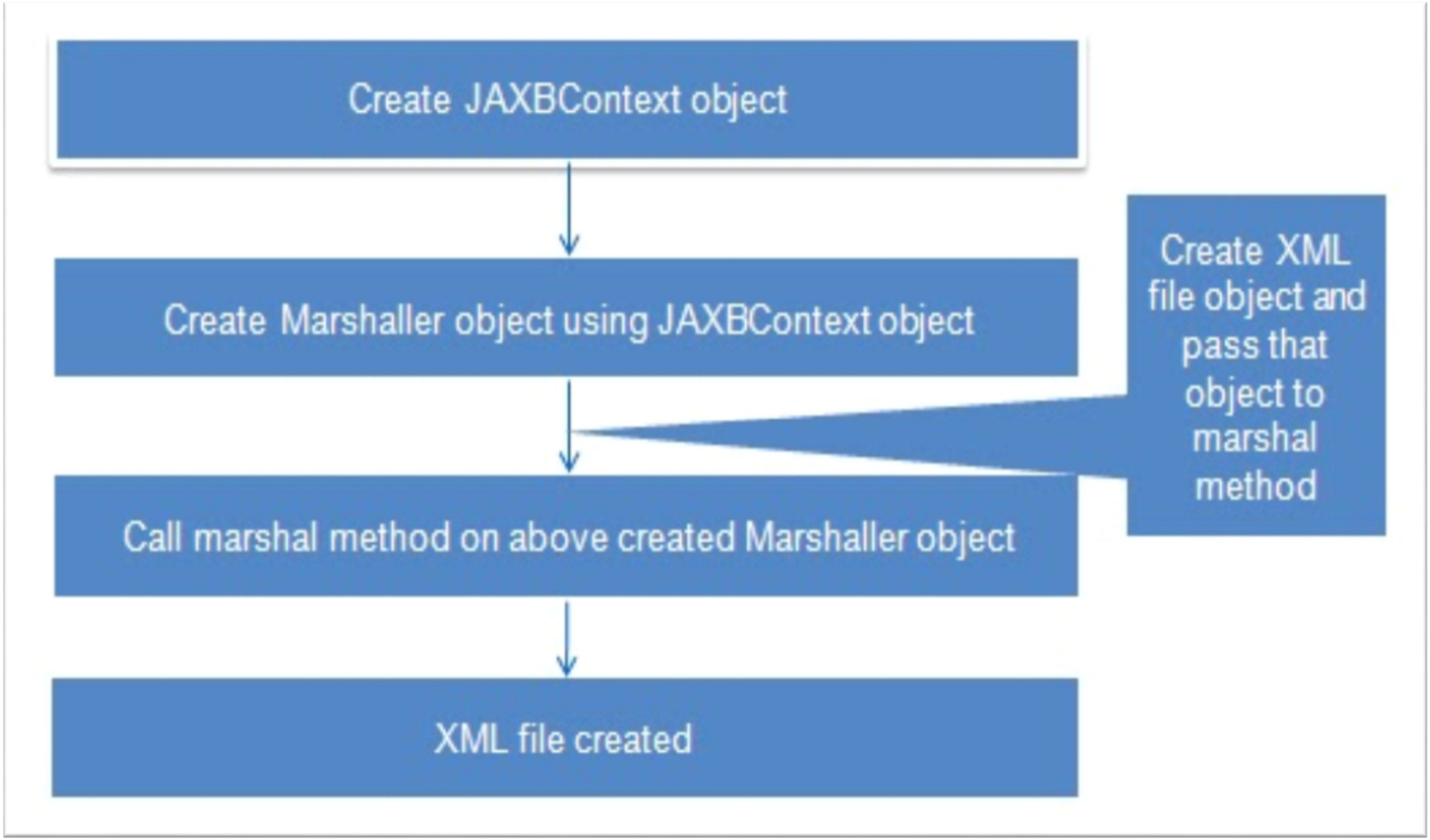
Using JAXB

- **With JAXB you can:**
  - **Save** Java Objects as XML file (**marshal**)
  - **Load** an XML file into Java (**unmarshal**)



- Prerequisite: Annotated java classes
- Two ways to get them:
  - Annotate by hand
  - Generate classes from xml schema

Using JAXB - Marshalling



Using JAXB - Marshalling

Pass main class

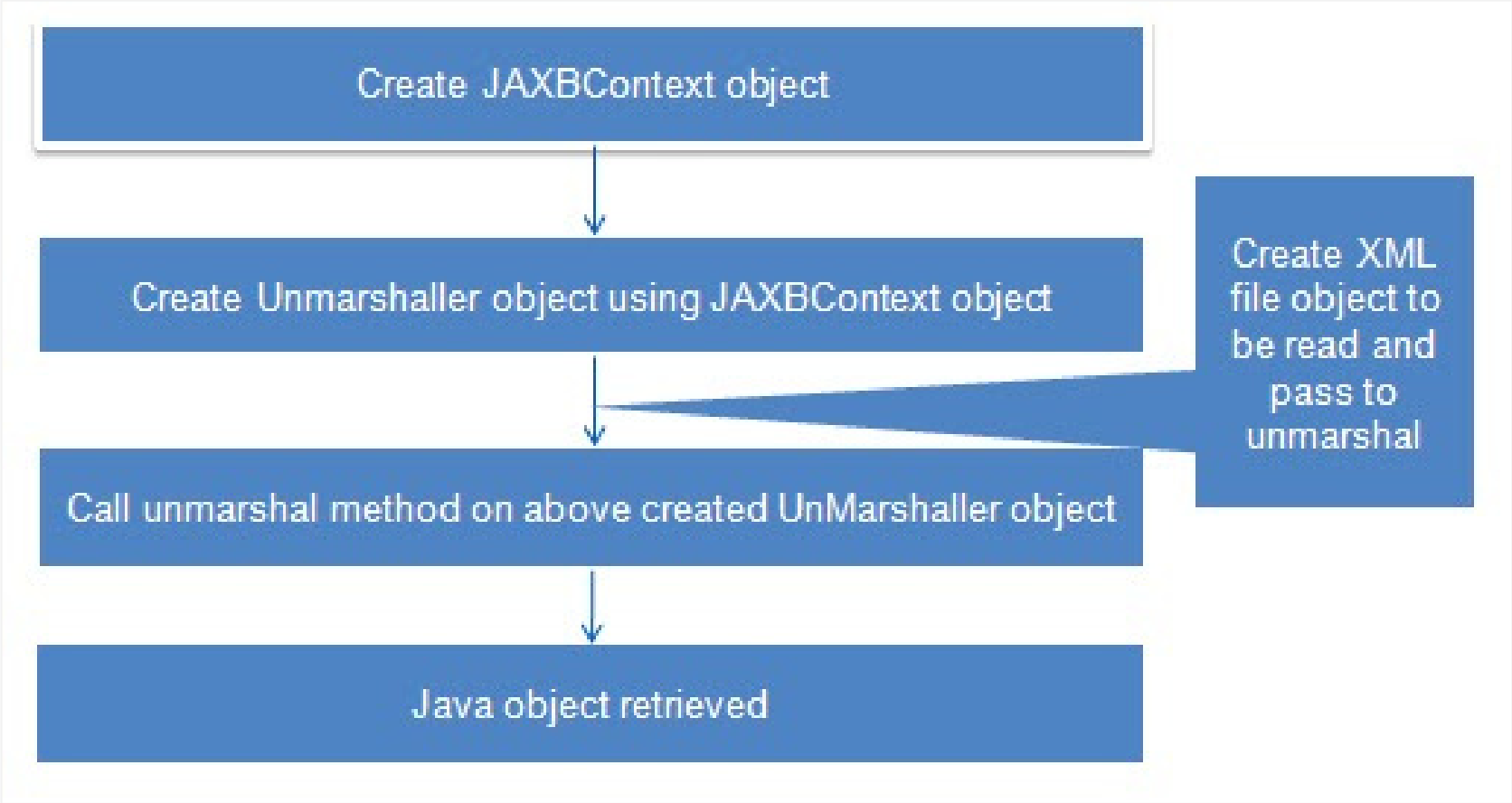
```
// create JAXB context object
JAXBContext context = JAXBContext.newInstance(BookStore.class);

// create Marshaller
Marshaller m = context.createMarshaller();
m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);

// Call marshall method
    // Write to System.out
    m.marshal(bookstore, System.out);
    // Write to File
    m.marshal(bookStore, new File("bookstore.xml"));
```

Instance of bookstore

Using JAXB - Unmarshalling



Using JAXB - Unmarshalling

Pass main class

```
// create JAXB context object
JAXBContext context = JAXBContext.newInstance(BookStore.class);

// create Unmarshaller
Unmarshaller um = context.createUnmarshaller();

// Call ummarshall method
BookStore bookStore2 =
    (BookStore) um.unmarshal(new FileReader("bookstore.xml"));
```

Outline

- Installation
- Marshal and Unmarshal
- Annotating classes
- Generating classes from XSD

JAXB Annotations

- JAXB uses annotations to indicate the central elements which should be expressed as xml elements
- An annotation, is a form of syntactic metadata that can be added to Java source code.

```
@AnnotationKeyword(parameter1, parameter2, …)
```

- An annotation is backed by plain Java classes. The compiler translates the annotations to objects of these classes.

Example Bookstore

- The given class `BookStore.java` contains an address (`Address.java`) and an `ArrayList` of books (`Book.java`).
- Annotate the classes `BookStore.java` and `Book.java` with JAXB annotations to persist the bookstore as XML.

- Given classes:
  - `Bookstore.java`
  - `Book.java`
  - `Address.java`
  - `BookMain.java`

JAXB Annotation – `@XmlRootElement`

```
@XmlRootElement(Name = "name", namespace = "namespace")
```

- Defines the root element for an XML tree, **the one** that can function as an XML document, should be annotated with `XmlRootElement`. This annotation corresponds to an `xsd:element` construct being used at the outermost level of an XML schema
- Its optional elements are `name` and `namespace`. By default, the class name is used as the name.
  - Use the element `name` to change the name of the tag
  - Use the element `namespace` to add a namespace to the xml document
- Example: **`BookStore.java`**

```
@XmlRootElement(name = "store", namespace = "ch.bfh.prog2.bookstore")
```

- **Attention**: Class must have a no-arg default constructor

JAXB Annotation – `@XmlElement`

```
@XmlElement(name = "newName")
```

- Define the XML element which will be used.
- Adds information, which isn't part of a Java class declaration.
- It permits to define the XML element name, the namespace, whether it is optional or nillable, a default value and the Java class.
  - Optional parameters: `name`, `nillable`, `required`
- Allows to turn fields or methods into xml elements, e.g. BookStore.getNmbOfBooks()

- Example: BookStore.java

JAXB Annotation – `@XmlElementWrapper`

```
@XmlElementWrapper
```

- **Optional**
- Additional element for lists to distinguish between a list that is absent and an empty list.
  - The name of the List object reference is used as surrounding xml element
- Example: **`BookStore.java`**

JAXB Annotation – `@XmlType`

```
@XmlType(propOrder={"field2","field1",.. })
```

- **Optional**
- Adds information useful for an xml type, which isn't part of a Java class declaration.
- The `namespace` attribute provides the name of the target namespace.
- The attribute `propOrder` establishes an ordering of the sub-elements. (if not used an alphabetic order is used)

- Example: Book.java

- **Attention**: Class must have a no-arg default constructor

- More annotations: https://docs.oracle.com/javase/tutorial/jaxb/intro/customize.html

JAXB Annotations

Annotation	Annotation for	Description
@XmlRootElement(Name = “name” namespace = "namespace")	(Main) Class	Defines the root element for an XML tree
@XmlElement(name = "newName")	Field/Method	Define the XML element which will be used
@XmlElementWrapper	Field/Method	Additional element for lists to distinguish between a list that is absent and an empty list.
@XmlType(propOrder = { "field2", "field1",.. })	Class	Allows to add more infos to a class, e.g. the order of fields
@XmlSeeAlso({class1, class2, …})	Class	Allows to use class hierachies with correct type information
@XmlAttribute(name = "name")	Field/Method	Defines an attribute
@XmlAccessorType(XmlAccessType.FIELD)	Class	Controls how a class is is serialized by default
…		

JAXB Adapters

- Used when a custom marshalling / unmarshalling is required
- Example:

```
@XmlJavaTypeAdapter(value=LocalDate2XsdDateAdapter.class)
private LocalDate birthdate;
```

```
public class LocalDate2XsdDateAdapter extends XmlAdapter<XMLGregorianCalendar, LocalDate> {

    @Override
    public XMLGregorianCalendar marshal(LocalDate birthdate) throws Exception {
        ...
    }

    @Override
    public LocalDate unmarshal(XMLGregorianCalendar birthdateXML) throws Exception {
        ...
    }
}
```

Outline

- Installation
- Marshal and Unmarshal
- Annotating classes
- Generating classes from XSD

Create Java Classes from XML Schema: xjc

- Often you have to comply with a given xml schema
- In such cases annotating by hand is rather cumbersome
- ➔ Use the XML to Java compiler **xjc**

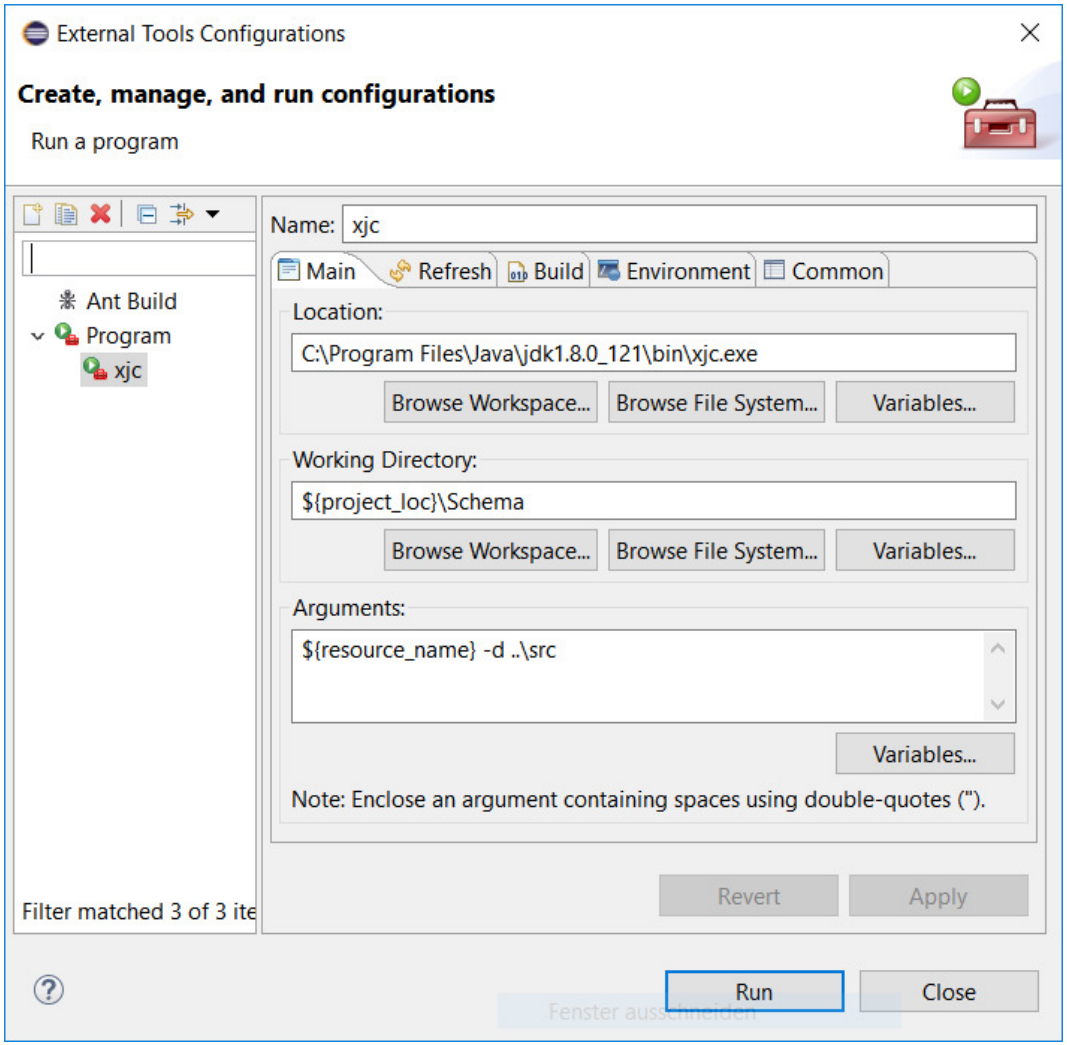
- On command line:
  - **xjc schemaFile –d destinationDircetory**
  - See https://docs.oracle.com/javase/8/docs/technotes/tools/unix/xjc.html

```
D:\BFH\OOP2\workspace\Exercise5TaskX\Schema>"C:\Program Files\Java\jdk1.8.0_121\bin\xjc.exe" Persons.xsd -d ..\src
Ein Schema wird geparst ...
Ein Schema wird kompiliert ...
https\github_com\bti7055_objectorientedprogramming\fs18_documents\topic04\MaritalstatusType.java
https\github_com\bti7055_objectorientedprogramming\fs18_documents\topic04\ObjectFactory.java
https\github_com\bti7055_objectorientedprogramming\fs18_documents\topic04\PersonType.java
https\github_com\bti7055_objectorientedprogramming\fs18_documents\topic04\PersonsType.java
https\github_com\bti7055_objectorientedprogramming\fs18_documents\topic04\package-info.java

D:\BFH\OOP2\workspace\Exercise5TaskX\Schema>
```

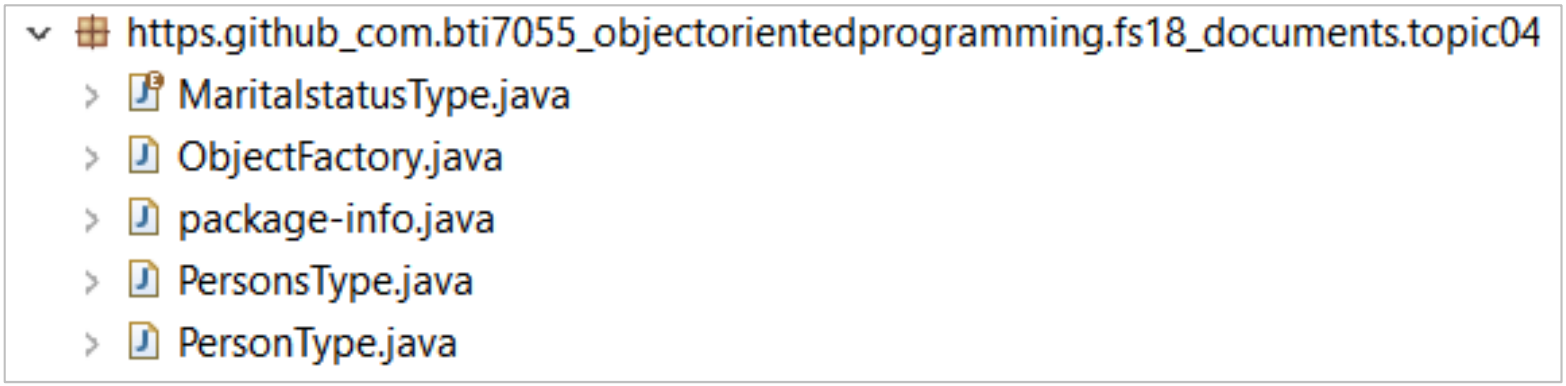
Create Java Classes from XML Schema: xjc

- Create an External Tool Configuration named xjc in your Eclipse project
- Schema file must be in directory called Schema
- Select schema file and run xjc directly within Eclipse 
- Remark: Eclipse allows to install a plugin for JAXB projects. Unfortunately this plugin has a bug, when you use Java 8.



Create Java Classes from XML Schema 1/5

- Example: Persons.xsd from Exercise 4, Task 6
  - Generated classes



- Let's have a look at the generated code

Create Java Classes from XML Schema 2/5

The generated code

- **package-info.java**
  - Maps the package to the corresponding namespace

- **PersonsType.java**

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "PersonsType", propOrder = {
    "person"
})
public class PersonsType {

    @XmlElement(name = "Person")
    protected List<PersonType> person;
    @XmlAttribute(name = "version", namespace = "https://...")
    protected String version;
    // getters/setters …
```

Create Java Classes from XML Schema 3/5

- **PersonType.java**

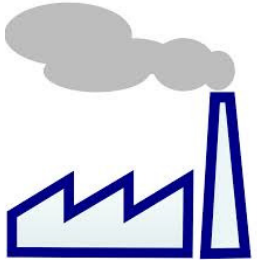
```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "PersonType", propOrder = {
    "name", "firstname", "birthdate", "maritalstatus"
})
public class PersonType {
    @XmlElement(name = "Name", required = true)
    protected String name;
    @XmlElement(name = "Firstname", required = true)
    protected String firstname;
    @XmlElement(name = "Birthdate", required = true)
    @XmlSchemaType(name = "date")
    protected XMLGregorianCalendar birthdate;
    @XmlElement(name = "Maritalstatus", required = true)
    @XmlSchemaType(name = "string")
    protected MaritalstatusType maritalstatus;
    // getters/setters ...
```

Create Java Classes from XML Schema 4/5

- **MaritalstatusType.java**

```
@XmlType(name = "MaritalstatusType")
@XmlEnum
public enum MaritalstatusType {
    // enum values …
```

Create Java Classes from XML Schema 5/5



- **ObjectFactory.java**
  - Provides create-method for object representing the root element in the XML document

```
@XmlRegistry
public class ObjectFactory {
  private final static QName _Persons_QNAME =
     new QName("https://...", "Persons");

  @XmlElementDecl(namespace = "https://...", name = "Persons")
  public JAXBElement<PersonsType> createPersons(PersonsType value) {
    return new JAXBElement<PersonsType>(
       _Persons_QNAME, PersonsType.class, null, value);
  }
  …
```

  - … and further create-methods for PersonsType and PersonType objects

Marshaller

```
JAXBContext jaxbContext =
  JAXBContext.newInstance("https.github.com....");

Marshaller marshaller = jaxbContext.createMarshaller();

marshaller.setProperty(
  Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);

marshaller.setProperty(
  Marshaller.JAXB_SCHEMA_LOCATION,
  "https://github.com/... Schema/Persons.xsd");
```

- Use package name in context
- Set schema location property in marshaller

Marshalling with Generated Classes

```
ObjectFactory factory = new ObjectFactory();

PersonsType personsXML = factory.createPersonsType();
personsXML.setVersion("FS2018");

for (Person person : persons) {
  PersonType personXML = factory.createPersonType();
  // fill in field values …
  personsXML.getPerson().add(personXML);
}

JAXBElement<PersonsType> personsElement =
  factory.createPersons(personsXML);

marshaller.marshal(personsElement, out);
```

Validating Unmarshaller

```
JAXBContext jaxbContext =
  JAXBContext.newInstance("https.github.com....");

Unmarshaller unmarschaller = jaxbContext.createUnmarshaller();

Schema schema = SchemaFactory.newInstance(
  XMLConstants.W3C_XML_SCHEMA_NS_URI).newSchema(
    new File("Schema/Persons.xsd"));

unmarschaller.setSchema(schema );
```

- Create a schema object and set it within the unmarshaller

Unmarshalling with Generated Classes

```
JAXBElement<PersonsType> personsElement =
   (JAXBElement<PersonsType>) unmarschaller.unmarshal(in);

for (PersonType p : personsElement.getValue().getPerson()) {
  // …
}
```