**Marshalling and Unmarshalling**

**Important  : (java.io.Externalizable)**

In earlier version of Java, reflection was very slow, and so serializaing large object graphs (e.g. in client-server RMI applications) was a bit of a performance problem. To handle this situation, the java.io.Externalizable interface was provided, which is like java.io.Serializable but with custom-written mechanisms to perform the marshalling and [unmarshalling](http://en.wikipedia.org/wiki/Marshalling_%28computer_science%29) functions (you need to implement readExternal and writeExternal methods on your class). This gives you the means to get around the reflection performance bottleneck.

**Disadvantage :**A big downside of Externalizable is that you have to maintain this logic yourself – if you add, remove or change a field in your class, you have to change your writeExternal/readExternal methods to account for it.

**EXAMPLE:**

**Student Class**

import javax.xml.bind.annotation.\*;

@XmlRootElement

public class Student {

    private String name;

    private int id;

    private String subject;

    Student(){

    }

    Student(String name,int id,String subject){

        this.name=name;

        this.id=id;

        this.subject=subject;

    }

    @XmlElement

    public String getName() {

        return name;

    }

    public void setName(String name) {

        this.name = name;

    }

    @XmlAttribute

    public int getId() {

        return id;

    }

    public void setId(int id) {

        this.id = id;

    }

    @XmlElement

    public String getSubject() {

        return subject;

    }

    public void setSubject(String subject) {

        this.subject = subject;

    }

}

* **@XmlRootElement** specifies the root element for the XML document.
* **@XmlAttribute** specifies the attribute for the root element.
* **@XmlElement** specifies the sub-element for the root element.

try{

    //creating the JAXB context

    JAXBContext jContext = JAXBContext.newInstance(Student.class);

    //creating the marshaller object

    Marshaller marshallObj = jContext.createMarshaller();

    //setting the property to show xml format output

    marshallObj.setProperty(Marshaller.JAXB\_FORMATTED\_OUTPUT, true);

    //setting the values in POJO class

    Student student = new Student(“abhishek”, 1163, “hadoop”);

    //calling the marshall method

    marshallObj.marshal(student, new FileOutputStream(“/home/knoldus/Desktop/student.xml”));

} catch(Exception e) {

    e.printStackTrace();

}

**Unmarshalling: Converting XML to Objects**

try{

    //getting the xml file to read

    File file = new File(“/home/knoldus/Desktop/student.xml”);

    //creating the JAXB context

    JAXBContext jContext = JAXBContext.newInstance(Student.class);

    //creating the unmarshall object

    Unmarshaller unmarshallerObj = jContext.createUnmarshaller();

    //calling the unmarshall method

    Student student=(Student) unmarshallerObj.unmarshal(file);

    System.out.println(student.getName()+” “+student.getId()+” “+student.getSubject());

}catch(Exception e){

    e.printStackTrace();

}