XSTREAM - IMPLICIT COLLECTIONS ALIASING

Implicit collections aliasing is used when a collection is to be represented in XML without displaying the roots. For example, in our case, we need to display each note one by one but not in the 'notes' root node. Let us modify our example again and add the following code to it.

```
xstream.addImplicitCollection(Student.class, "notes");
```

Let us test the above objects serialization using XStream.

Create a java class file named XStreamTester in **C:\>XStream_WORKSPACE\com\tutorialspoint\xstream**.

File: XStreamTester.java

```
package com.tutorialspoint.xstream;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;

import java.util.ArrayList;
import java.util.List;

import javax.xml.transform.OutputKeys;
import javax.xml.transform.Source;
import javax.xml.transform.Transformer;
import javax.xml.transform.sax.SAXSource;
import javax.xml.transform.sax.SAXTransformerFactory;
import javax.xml.transform.stream.StreamResult;

import org.xml.sax.InputSource;

import com.thoughtworks.xstream.XStream;
import com.thoughtworks.xstream.io.xml.StaxDriver;

public class XStreamTester {
   public static void main(String args[]){

      XStreamTester tester = new XStreamTester();
      XStream xstream = new XStream(new StaxDriver());

      xstream.alias("student", Student.class);
      xstream.alias("note", Note.class);
      xstream.aliasField("name", Student.class, "studentName");
      xstream.addImplicitCollection(Student.class, "notes");

      Student student = tester.getStudentDetails();

      //Object to XML Conversion
      String xml = xstream.toXML(student);
      System.out.println(formatXml(xml));
   }

   private Student getStudentDetails(){
      Student student = new Student("Mahesh");

      student.addNote(new Note("first","My first assignment."));
      student.addNote(new Note("second","My Second assignment."));

      return student;
   }

   public static String formatXml(String xml){
```

```
      try{
         Transformer serializer = SAXTransformerFactory.newInstance().newTransformer();
         serializer.setOutputProperty(OutputKeys.INDENT, "yes");
         serializer.setOutputProperty("{http://xml.apache.org/xslt}indent-amount", "2");

         Source xmlSource = new SAXSource(new InputSource(new
ByteArrayInputStream(xml.getBytes())));
         StreamResult res =  new StreamResult(new ByteArrayOutputStream());

         serializer.transform(xmlSource, res);

         return new String(((ByteArrayOutputStream)res.getOutputStream()).toByteArray());

      }catch(Exception e){
         return xml;
      }
   }
}

class Student {
   private String studentName;
   private List<Note> notes = new ArrayList<Note>();

   public Student(String name) {
      this.studentName = name;
   }

   public void addNote(Note note) {
      notes.add(note);
   }

   public String getName(){
      return studentName;
   }

   public List<Note> getNotes(){
      return notes;
   }
}

class Note {
   private String title;
   private String description;

   public Note(String title, String description) {
      this.title = title;
      this.description = description;
   }

   public String getTitle(){
      return title;
   }

   public String getDescription(){
      return description;
   }
}
```

Verify the Result

Compile the classes using **javac** compiler as follows:

```
C:\XStream_WORKSPACE\com\tutorialspoint\xstream>javac XStreamTester.java
```

Now run the XStreamTester to see the result:

```
C:\XStream_WORKSPACE\com\tutorialspoint\xstream>java XStreamTester
```

Verify the output as follow:

```
<?xml version="1.0" encoding="UTF-8"?>
<student>
  <name>Mahesh</name>
  <note>
    <title>first</title>
    <description>My first assignment.</description>
  </note>
  <note>
    <title>second</title>
    <description>My Second assignment.</description>
  </note>
</student>
```

In the above result, we can see that name is coming as a child node and we need it as an attribute of the root node.