What is XStream?

XStream is a simple library to serialize objects to XML and back again.

Features

- **Ease of use.** A high level facade is supplied that simplifies common use cases.
- **No mappings required.** Most objects can be serialized without need for specifying mappings.
- **Performance.** Speed and low memory footprint are a crucial part of the design, making it suitable for large object graphs or systems with high message throughput.
- **Clean XML.** No information is duplicated that can be obtained via reflection. This results in XML that is easier to read for humans and more compact than native Java serialization.
- **Requires no modifications to objects.** Serializes internal fields, including private and final. Supports non-public and inner classes. Classes are not required to have default constructor.
- **Full object graph support.** Duplicate references encountered in the object-model will be maintained. Supports circular references.
- **Integrates with other XML APIs.** By implementing an interface, XStream can serialize directly to/from any tree structure (not just XML).
- **Customizable conversion strategies.** Strategies can be registered allowing customization of how particular types are represented as XML.
- **Security framework.** Fine-control about the unmarshalled types to prevent security issues with manipulated input.
- **Error messages.** When an exception occurs due to malformed XML, detailed diagnostics are provided to help isolate and fix the problem.
- **Alternative output format.** The modular design allows other output formats. XStream ships currently with JSON support and morphing.

How does it work?

Here's a simple classe. XStream can convert instances of these to XML and back again.

```
public class ActorTO {

        private String firstName;
        private String lastName;
        private int actorId;
        private Date lastUpdate;

        public ActorTO() {
                // TODO Auto-generated constructor stub
        }
 // ... constructors and methods
}
```

Notice that the fields are private. XStream **doesn't care about the visibility** of the fields. No getters or setters are needed. Also, XStream does not limit you to having a default constructor.

To use XStream, simply instantiate the XStream class:

```
XStream xstream = new XStream();
```

You require xstream-[version].jar, xpp3-[version].jar and xmlpull-[version].jar in the classpath. Xpp3 is a very fast XML pull-parser implementation.

```
String xml = xstream.toXML(actor);
```

The resulting XML looks like this:

```
<com.h2k.dto.ActorTO>
        <firstName>Niel</firstName>
       <lastName>Armstrong</lastName>
        <actorId>100</actorId>
        <lastUpdate>2017-10-29 13:56:02.15 UTC</lastUpdate>
</com.h2k.dto.ActorTO>
```

You definitely don't need complete class name as parent node. So let's add an Alias to this Class.

@XStreamAlias("Actor")

```
public class ActorTO {
       private String firstName;
       private String lastName;
       private int actorId;
       private Date lastUpdate;
}
```

Also, ask XStream instance to process Annotations from Actor Class.

```
xstream.processAnnotations(ActorTO.class);
String xmlActor = xstream.toXML(actor);
```

XML node looks much prettier this way. :)

```
<Actor>
  <firstName>Niel</firstName>
  <lastName>Armstrong</lastName>
  <actorId>100</actorId>
  <lastUpdate>2017-10-29 14:02:31.368 UTC</lastUpdate>
</Actor>
```

It's that simple. Look at how clean the XML is.

How is reverse process?

To reconstruct an object, purely from the XML:

```
ActorTO fromXmlActor = (ActorTO) xstream.fromXML(xmlActor);
```

And that's how simple XStream is!

Suppose that our client has defined a base XML file that we should make XStream read/write:

```
<blog author="Guilherme Silveira">
  <entry>
    <title>first</title>
    <description>My first blog entry.</description>
  </entry>
  <entry>
    <title>tutorial</title>
    <description>
        Today we have developed a nice alias tutorial. Tell your friends! NOW!
    </description>
  </entry>
</blog>
```

Based on the XML file above we shall create some model classes and configure XStream to write/read from this format.

```
public class Blog {
        private Author writer;
        private List entries = new ArrayList();

        public Blog(Author writer) {
                this.writer = writer;
        }
        public void add(Entry entry) {
                entries.add(entry);
        }
        public List getContent() {
                return entries;
        }
}
```

A basic author with name

```
public class Author {
        private String name;
        public Author(String name) {
                this.name = name;
        }
        public String getName() {
                return name;
        }
}
```

A blog entry contains a title and description:

```
public class Entry {
        private String title, description;
        public Entry(String title, String description) {
                this.title = title;
                this.description = description;
        }
}
```

We can easily instantiate a new blog and use it with xstream:

```
public static void main(String[] args) {
        Blog teamBlog = new Blog(new Author("Guilherme Silveira"));
        teamBlog.add(new Entry("first","My first blog entry."));
        teamBlog.add(new Entry("tutorial",
                "Today we have developed a nice alias tutorial. Tell your friends! NOW!"));
        XStream xstream = new XStream();
        System.out.println(xstream.toXML(teamBlog));

}
```

And the resulting XML is not so nice as we would want it to be:

```
<com.thoughtworks.xstream.Blog>
  <writer>
    <name>Guilherme Silveira</name>
  </writer>
  <entries>
    <com.thoughtworks.xstream.Entry>
      <title>first</title>
      <description>My first blog entry.</description>
    </com.thoughtworks.xstream.Entry>
    <com.thoughtworks.xstream.Entry>
      <title>tutorial</title>
      <description>
        Today we have developed a nice alias tutorial. Tell your friends! NOW!
      </description>
    </com.thoughtworks.xstream.Entry>
  </entries>
</com.thoughtworks.xstream.Blog>
```

Lets add some annotations and make it prettier. As List output was not desired therefore we will annotate the content list to be recognized as an implicit collection:

```
@XStreamAlias("Blog")
public class Blog {
       @XStreamAlias("author")
       private Author writer;

       @XStreamImplicit
       private List<Entry> entries = new ArrayList<>();

       public Blog(Author writer) {
           this.writer = writer;
       }
       public void add(Entry entry) {
            entries.add(entry);
       }
           public List getContent() {
                    return entries;
    }
}
```

The implicit annotation can also be used for arrays and maps. In the latter case you should provide the field name of the values that are used as key of the map.

```
@XStreamAlias("author")
public class Author {

        @XStreamAsAttribute
         private String name;
     public Author(String name) {
              this.name = name;
     }
     public String getName() {
              return name;
     }
}
```

The client may asks for the type tag and the importance flag to be an attribute inside the message tag, All you need to do is add the @XStreamAsAttribute annotation like above.

```
@XStreamAlias("Entry")
public class Entry {
        private String title, description;
    public Entry(String title, String description) {
            this.title = title;
            this.description = description;
    }
}
```

After making changes in Classes with annotations, let's run our test code and see the results.

```
public class TesterClass {

        public static void main(String[] args) {
                Blog teamBlog = new Blog(new Author("Guilherme Silveira"));
                teamBlog.add(new Entry("first","My first blog entry."));
                teamBlog.add(new Entry("tutorial",
                  "Today we have developed a nice alias tutorial. Tell your friends! NOW!"));

                XStream xstream = new XStream();
                xstream.processAnnotations(Blog.class);
                xstream.processAnnotations(Author.class);
                xstream.processAnnotations(Entry.class);

                System.out.println(xstream.toXML(teamBlog));
                }
        }
```

http://x-stream.github.io/ ← Official Site for XStream where you can get latest versions, updates and examples.