Object Serialization

Objectives

- Learn how to persist object state using serialization
- Learn how to control the serialization process
- Understand object versioning

Serialization

- What if you want to save the state of an object between sessions?
  - User interface geometry
  - Session parameters (ip addresses, modem init strings, etc.)
  - Email address book in a mail reader program
  - Objects drawn by the user in your new jPhotoshop software

Serialization (cont'd)

- Serialization allows you to save object state to a stream
- Provides a simple persistence mechanism for objects
- Default mechanism saves the value of nonstatic, nontransient attributes
- Most Java API objects are serializable, but you should check if unsure

The `Serializable` Interface

- **To be serialized, an object's class must implement the** `java.io.Serializable` **interface**
- `Serializable` **contains no methods (it's a marker interface, like** `Cloneable`**)**

What Serialization Does

- Serialization is a “deep copy” operation… it will attempt to serialize all contained objects in turn (and all of those objects’ contained objects, and so on)
- If a contained object somewhere does not implement `Serializable`, a `java.io.NotSerializableException` will be thrown

Serialization File Format

- File format
  - File begins with a magic number AC ED
  - Version number, currently 00 05
  - Objects
- Objects have their ids
  - String 74, Class 72, Object 73
- String “Java” will be saved as :

Magic #		Version		Id	Size		“Java”			
AC	ED	00	05	74	00	04	4A	61	76	61

Storing Object Descriptions

- Saving arbitrary classes requires saving the class info itself
  - Name of a class
  - Serial version unique ID (fingerprint)
  - Description of method used
  - Description of data fields
- Methods in `ObjectStreamConstants`
  - `SC_WRITE_METHOD`
  - `SC_SERIALIZABLE`
  - `SC_EXTERNALIZABLE`

Storing Object Data

- Field descriptors
  - 1-byte type code
    - `J     long`
    - `Z     boolean`
    - `L     object`
    - `[`   array
    - Other primitives by first letter
  - 2-byte length
  - Field name
  - Class name (if an object)

`ObjectOutputStream`

- **To write an object's state to the stream, use the `ObjectOutputStream` class**
  - **Use `writeObject(Object o)` to save an object's state to a stream**
- **Constructor takes any `OutputStream`**
  - **Most often wrapped around a `FileOutputStream`**

`ObjectInputStream`

- To read an object's state from a stream, use the `ObjectInputStream` class
  - Use `readObject()` to read an object's state from the stream
- Constructor takes any `InputStream`
  - Most often wrapped around a `FileInputStream`
- `readObject()` returns an `Object`; you must cast it to the appropriate type

Object Stream Exceptions

- **Note which** `Exceptions` **have to be caught, and where**
  - `IOException` **for streams**
  - `ClassNotFoundException` **on** `ObjectInputStream.readObject()`

Serialization Example

– Example: Serializing an Object

```
try {
    AnObject object = new AnObject();

    ObjectOutputStream out =
       new ObjectOutputStream(
          new FileOutputStream("filename.ser"));
    out.writeObject(object);
    out.close();
 }
 catch(IOException ex) {
    ex.printStackTrace();
 }
```

Deserialization Example

– **Example: Deserializing an Object**

```
    try {
        ObjectInputStream in =
           new ObjectInputStream(
              new FileInputStream("filename.ser"));
        AnObject object = (AnObject)in.readObject();
        in.close();
     }
     catch(ClassNotFoundException ex) {
        ex.printStackTrace();
     }
     catch(IOException ex) {
        ex.printStackTrace();
     }
```

What Should not be Serialized

- What kinds of classes are not Serializable?
  - Streams
  - DB and network connections
  - Graphics contexts
  - Anything that can only be determined from current context
- If your class contains these types, you can still make your class serializable by marking them with the transient keyword:
  - `private transient FileInputStream file;`
- Transient fields are ignored during serialization

Custom Serialization

- But might doing this not mess up your class' state?
- You can control serialization beyond what is built in to the JDK
- Implement two methods in your class:

```
private void writeObject(ObjectOutputStream out)
private void readObject(ObjectInputStream in)
```

Custom Serialization (cont’d)

- **Within custom methods you can use the default method of the stream, then do any needed additional processing**

```
    private void readObject(ObjectInputStream in) {
          in.defaultReadObject();
          // Process calculated attributes here
          today = calendar.getTime();
    }
```

Ultimate Serialization Control

- **Implement** `Externalizable` **(which extends** `Serializable`**)**
- **Defines two methods:** `writeExternal()` **and** `readExternal()`
- **These allow you to specify exactly how objects are stored to the underlying stream**

`Externalizable`

- When you want to do all the work
- Extends `Serializable`
- Implementing class has complete control over serialization process
  - All data of superclass
  - any versioning issues
  - etc.

Externalizable

- **Two methods**

```
void writeExternal( ObjectOutput )
void readExternal( ObjectInput )
```

CubeEx Example

```
public class ExCube implements java.io.Externalizable {
   ...
   public void writeExternal(ObjectOutput out)
   throws IOException {
      out.writeDouble(height);
      out.writeDouble(width);
      out.writeDouble(depth);
   }

   public void readExternal(ObjectInput in)
   throws IOException, ClassNotFoundException {
      height = in.readDouble();
      width = in.readDouble();
      depth = in.readDouble();
   }
   ...
```

Class Versioning

- Fingerprint is SHA computed 20 bytes
  - ~100% fingerprint changes if data is changed
  - Java uses only 8 bytes - still OK
  - Checks data and methods(!)
- Why worry?
  - If class layout is changed, original data may corrupt the memory
  - Class definition can be altered to hack into programs

Versioning

- Versioning is used for compatibility with older software
  - `serialver` tool in JDK
  - `static final long serialVersionID = 2121…21L;`
- Used instead of computing SHA fingerprint
- Dealing with different versions
  - Variable changes type
  - Variables added
  - Variables deleted

Object Versioning

- Each time an object is written, its serial version UID (unique identifier is written with it
- Attempting to de-serialize and object saved with a different UID causes an `InvalidClassException`

ObjectStreamField

- Maps field name and type
- Allows definition of the persisted fields
  - An alternative to transient
  - `serialPersistentFields` identifies fields to be persisted

```
private static final
        ObjectStreamField[] serialPersistentFields
```

- Object streams read/write fields
- Works with default serialization
- May aid in class evolution
  - Map persisted fields to actual fields
  - Set `serialVersionID`

Supporting Elements

- `ObjectInputStream`
  - `GetField` **inner class**
    - `*type* get( String, *type* )`
  - `GetField readFields()`
- `ObjectOutputStream`
  - `PutField` **inner class**
    - `void put( String, *type* )`
  - `void writeFields()`
  - `PutField putFields()`

Cube Version 1

```
public class Cube implements java.io.Serializable {
   private double height;
   private double width;
   private double depth;
   static final long serialVersionUID = 8233351143186842863L;
   private static final ObjectStreamField[] serialPersistentFields = {
                       new ObjectStreamField("x", double.class),
                       new ObjectStreamField("y",  double.class),
                       new ObjectStreamField("z",  double.class)
                    };

   public Cube( double height, double width, double depth ) {
      this.height = height;
      this.width = width;
      this.depth = depth;
   }
```

Cube Version 1

```
    private void readObject(ObjectInputStream ois)
    throws ClassNotFoundException, IOException {
       ObjectInputStream.GetField fields = ois.readFields();
       width = fields.get("x", 0.0 );
       height = fields.get("y", 0.0);
       depth = fields.get("z", 0.0);
    }

    private void writeObject(ObjectOutputStream oos)
    throws IOException {
       ObjectOutputStream.PutField fields = oos.putFields();
       fields.put("x", width);
       fields.put("y", height);
       fields.put("z", depth);
       oos.writeFields();
    }
}
```

Cube Version 2

```
public class Cube implements java.io.Serializable {
   private Rect   rect;
   private double depth;
   static final long serialVersionUID = 8233351143186842863L;
   private static final ObjectStreamField[] serialPersistentFields = {
                       new ObjectStreamField("x", double.class),
                       new ObjectStreamField("y",  double.class),
                       new ObjectStreamField("z",  double.class)
                    };

   public Cube(double height, double width, double depth) {
      this.rect = new Rect(width, height);
      this.depth = depth;
   }
```

Cube Version 2

```
   private void readObject(ObjectInputStream ois)
   throws ClassNotFoundException, IOException {
      ObjectInputStream.GetField fields = ois.readFields();
      double w = fields.get("x", 0.0);
      double h = fields.get("y", 0.0);
      depth = fields.get("z", 0.0);
      rect = new Rect( w, h );
   }

   private void writeObject(ObjectOutputStream oos)
   throws IOException {
      ObjectOutputStream.PutField fields = oos.putFields();
      fields.put("x", rect.getWidth());
      fields.put("y", rect.getHeight());
      fields.put("z", depth);
      oos.writeFields();
   }
}
```

Substituting Objects

- It is some time necessary to replace the object read from stream with one of your choosing
  - Singletons
  - Type safe-enumerations
- Classes may implement the `readResolve` method to return the “chosen” object

Singleton Example

```
public class ASingleton implements Serializable {
   private ASingleton theInstance;

   private ASingleton() {
   }

   public static ASingleton getInstance() {
      if (theInstance == null) {
         theInstance = new ASingleton();
      }
      return theInstance;
   }

   private Object readResolve() {
      return getInstance();
   }


   ...
}
```

Serialization Proxy

- Serializing a proxy instance prevents security problems from making instances
- Process
  - Define private static nested class representing enclosing classes state - the proxy
    - Implement `readResolve()` method to create and return an instance of the enclosing class
  - In the enclosing class
    - Implement `writeReplace()` method in enclosing class
    - Implement `readObject(ObjectInputStream)` to throw `InvalidObjectException` in enclosing class

Serialization Proxy Example

```
import java.io.InvalidObjectException;
import java.io.ObjectInputStream;
import java.io.Serializable;

public final class Cube implements Serializable {
    private double height, width, depth;

    public Cube(final double width, final double height, final double depth) {
        this.width = width;
        this.height = height;
        this.depth = depth;
    }

    public double getHeight() { return height; }

    public double getWidth() { return width; }

    public double getDepth() { return depth; }
```

Serialization Proxy Example

```
    private Object writeReplace() {
         return new SerializationProxy(this);
     }

     private void readObject(ObjectInputStream ois) throws InvalidObjectException {
         throw new InvalidObjectException("Proxy required");
     }

     private static class SerializationProxy implements Serializable {
         private double x, y, z;

         SerializationProxy(final Cube cube) {
             y = cube.height;
             x = cube.width;
             z = cube.depth;
         }

         private Object readResolve() {
             return new Cube(x, y, z);
         }
     }
}
```

Validation Mechanics

- Object being deserialized:
  - Provides private `readObject` method
  - Registers validation object prior to reading the object from stream

    `registerValidation(ObjectInputValidation,int)`
  - Validator invoked after object completely read
- Validation object:
  - Implements `ObjectInputValidation`
  - The class of object being read or a class higher in the hierarchy

`SafeCube` Example

- `SafeCube` Class
  - `height`, `width`, `depth`, `surfaceArea`, `volume`
  - `surfaceArea`, and `volume` are transient
  - Save and restore the object, won't restore two dimensional cubes
  - Initialize `surfaceArea` and `volume`

SafeCube

```
public class SafeCube implements Serializable, ObjectInputValidation {
   private transient double surfaceArea;
   private transient double volume;
   private double    height;
   private double    width;
   private double    depth;

   public SafeCube( double height, double width, double depth ) {
     this.height = height;
     this.width = width;
     this.depth = depth;
     surfaceArea = (width*height + width*depth + height*depth) * 2;
     volume = width * height * depth;
   }

   ...
```

SafeCube

```
...
public void validateObject() throws InvalidObjectException {
   if (height == 0 || width == 0 || depth == 0) {
     throw new InvalidObjectException(
                                    "Cube is not three dimensional!");
   }
   surfaceArea = (width*height + width*depth + height*depth) * 2;
   volume = width * height * depth;
   System.out.println("Initializing surface area and volume.");
}

private void readObject(ObjectInputStream in)
throws IOException, ClassNotFoundException {
   in.registerValidation(this, 0);
   in.defaultReadObject();
}
...
```

Serialization Related Warnings

- Use of serialization commonly causes a number of warnings:
  - Unchecked warning, the compiler is unable to verify the correctness of a casting operation

    ```
    warning: [unchecked] unchecked cast
    ```

  - Serial warning, a class is declared `Serializable` but doesn't define a `serialVersionUID` class field

    ```
    warning: [serial] serializable class classname
    has no definition of serialVersionUID
    ```

@SuppressWarnings

- **Annotation suppresses the named warnings**
  - **Can be applied to different scopes; class, method, block or statement**
  - **Should be applied to the narrowest scope**
- **Syntax:** `@SuppressWarnings(value)`
  - **Where** `value` **is as either a string literal containing a warning name or for multiple warnings an array of string literals**

```
@SuppressWarnings("serial")
@SuppressWarnings({"serial", "unchecked"})
```

@SuppressWarnings

- To apply `@SuppressWarnings("unchecked")` to a single assignment statement (with a cast) the target variable declaration and assignment must be a single statement

Alternatives to Serializable

- XML Serialization
- JSON
- JAXB

XML Serialization

- Only works with JavaBeans
  - Uses getters and setters
  - Uses no argument constructor
  - Need not implement `java.io.Serializable`
- Implemented with encoder and decoder classes
  - `java.beans.XMLEncoder`
  - `java.beans.XMLDecoder`

Cube XML Example

```
public class Cube {
  private double width, height, depth;

  public Cube() {}

  public Cube(double width, double height, double depth) {
    this.height = height;
    this.width = width;
    this.depth = depth;
  }

  public double getWidth() {
    return this.width;
  }

  public void setWidth(double width) {
    this.width = width;
  }
  ...
```

Cube XML Example

```
public static void main(String args[]) {
   Cube kube = new Cube(2.0, 1.0, 3.0);
   System.out.println(kube);
   try {
      FileOutputStream f = new FileOutputStream("cube.xml");
      XMLEncoder out = new XMLEncoder(f);
      out.writeObject(kube);
      out.close();
      FileInputStream fis = new FileInputStream("cube.xml");
      XMLDecoder in = new XMLDecoder(fis);
      System.out.println();
      Cube c = (Cube)in.readObject();
      in.close();
      System.out.println(c);
    }
    catch(IOException ex) {
      System.out.println(ex);
    }
}
```

Serialized XML

```
<?xml version<?xml version="1.0" encoding="UTF-8"?>
<java version="1.6.0_29" class="java.beans.XMLDecoder">
  <object class="org.xml.jaxb.Cube">
    <void property="depth">
      <double>3.0</double>
    </void>
    <void property="rect">
      <void property="height">
        <double>1.0</double>
      </void>
      <void property="width">
        <double>2.0</double>
      </void>
    </void>
  </object>
</java>
```

JSON

JSON

- JavaScript Object Notation
- www.json.org
- RFC 4627
- Language independent
  - Implementations available in dozens of languages
  - Multiple Java implementations, vary in capabilities
  - API not standardized

JSON Basic Types

- Number
  - Type not specified, double precision floating-point format in practice
- String
  - Double-quoted Unicode
    - UTF-8 by default
    - Backslash escaping
- Boolean
  - Literals `true` or `false`

JSON Basic Types

- Array
  - An ordered sequence of values
    - enclosed in brackets
    - comma-separated
    - values do not need to be of the same type
- Object
  - An unordered collection of key:value pairs
    - enclosed in curly braces
    - comma-separated
    - The ':' character separates the key and the value
    - Keys must be strings and should be distinct
- null (empty)

JSON Sntax
Number

- Number
  - Digits 0 through 9
  - Optional sign
  - Optional decimal point
  - Optional exponent
    - E or e
    - Optional sign
    - Sequence of digits

JSON Syntax
String

- Sequence of any Unicode characters (except backslash and quote)contained in double-quotes
- Special characters escaped with backslash
  - Quotation mark - \“
  - Backslash - \\
  - Backspace - \b
  - Formfeed - \f
  - Newline - \n
  - Carriage return - \r
  - Tab- \t
  - Unicode - \uxxxx (exactly 4 hexadecimal digits)

JSON Syntax
Array

- Any of:
  - String
  - Number
  - Object
  - Array
  - Literals – `true`, `false`, `null`

JSON Syntax
Object

- Comma separated name value pair list enclosed in braces
- Name value pair
  - Name is a string
  - Value is any legal value
  - Separated by the colon character

JSON Example
Code (`Portfolio`)

```java
public class Portfolio {
    private static final String NL = System.getProperty("line.separator");
    String id;
    List<Holding> holdings;

    public Portfolio() {}

    public Portfolio(String id, List<Holding> holdings) {
        this.id = id;
        this.holdings = holdings;
    }

    public String getId() { return id; }

    public void setId(String id) { this.id = id; }

    public List<Holding> getHoldings() { return holdings; }

    public void setHoldings(List<Holding> holdings) { this.holdings = holdings; }

    ...
}
```

JSON Example

Code (Holding)

```
public class Holding {
    private Stock stock;
    private int shares;

    public Holding() { }

    public Holding(Stock stock, int shares) {
        this.stock = stock;
        this.shares = shares;
    }

    public Stock getStock() { return stock; }

    public void setStock(Stock stock) { this.stock = stock; }

    public int getShares() { return shares; }

    public void setShares(int shares) { this.shares = shares; }

    public String toString() {
        return Integer.toString(shares) + " shares of " + stock;
    }
}
```

JSON Example

Code (Stock)

```java
public class Stock {
    private String stockSymbol;
    private int currentSharePrice;

    public Stock() { }

    public Stock(String stockSymbol, int currentSharePrice) {
        this.stockSymbol = stockSymbol;
        this.currentSharePrice = currentSharePrice;
    }

    public String getStockSymbol() { return stockSymbol; }

    public void setStockSymbol(String stockSymbol) { this.stockSymbol = stockSymbol; }

    public int getCurrentSharePrice() { return currentSharePrice; }

    public void setCurrentSharePrice(int currPrice) { this.currentSharePrice = currPrice; }

    public String toString() {
        return stockSymbol + " @ " + currentSharePrice;
    }
}
```

JSON Example
Code

```java
import java.io.File;
import java.util.ArrayList;

// Using Jackson, http://jackson.codehaus.org
import org.codehaus.jackson.map.ObjectMapper;

public class JsonEx {

    public static void main(String[] args) throws Exception {
        // Create the portfolio object graph
        ArrayList<Holding> holdings = new ArrayList<Holding>();
        holdings.add(new Holding(new Stock("AAPL", 13000), 10000));
        holdings.add(new Holding(new Stock("MSFT", 11000), 1000));
        holdings.add(new Holding(new Stock("F", 5400), 5000));
        Portfolio p = new Portfolio("Abc123", holdings);

        // Print the original
        System.out.println(p);
```

JSON Example

Code

```
        // Create a file
        File file = new File("portfolio.json");

        // Write it out
        ObjectMapper mapper = new ObjectMapper();
        mapper.writeValue(file, p);

        Portfolio px = mapper.readValue(file, Portfolio.class);
        // Print the copy
        System.out.println(px);
    }
}
```

JSON Example
Serialized File

```
{"id":"Abc123","holdings":[
    {"stock":{"stockSymbol":"AAPL","currentSharePrice":13000},
     "shares":10000},
    {"stock":{"stockSymbol":"MSFT","currentSharePrice":11000},
     "shares":1000},
    {"stock":{"stockSymbol":"F","currentSharePrice":5400},
     "shares":5000}
]}
```

JAXB

JAXB

- Java Architecture for XML Binding
- http://www.oracle.com/technetwork/articles/javase/index-140168.html#xmp1
- Framework for XML documents into Java objects
- Annotations used to direct the binding process

JAXB Type Mapping

- Scalar datatypes of the XML Schema Language are mapped to Java data types
- Lists of values and certain element groupings are mapped to Java's `java.util.List`
- XML Schema structures that fail automated binding may use binding declarations to dictate binding

XML Schema to Java

Data Type	
XML Schema	Java
anySimpleType (for xsd:element of this type)	java.lang.Object
anySimpleType (for xsd:attribute of this type)	java.lang.String
base64Binary	byte[]
boolean	boolean
byte	byte
date	java.xml.datatype.XMLGregorianCalendar
dateTime	javax.xml.datatype.XMLGregorianCalendar
decimal	java.math.BigDecimal
double	double
duration	javax.xml.datatype.Duration
float	float
g	java.xml.datatype.XMLGregorianCalendar
hexBinary	byte[]
int	int
integer	java.math.BigInteger
long	long
NOTATION	javax.xml.namespace.QName
Qname	javax.xml.namespace.QName
short	short
string	java.lang.String
time	java.xml.datatype.XMLGregorianCalendar
unsignedByte	short
unsignedInt	long
unsignedShort	int

Java to XML Schema

Data Type	
Java	**XML Schema**
boolean	boolean
byte	byte
double	double
float	float
long	long
int	int
javax.activation.DataHandler	base64Binary
java.awt.Image	base64Binary
java.lang.Object	anyType
java.lang.String	string
java.math.BigInteger	integer
java.math.BigDecimal	decimal
java.net.URI	string
java.util.Calendar	dateTime
java.util.Date	dateTime
java.util.UUID	string
javax.xml.datatype.XMLGregorianCalendar	anySimpleType
javax.xml.datatype.Duration	duration
javax.xml.namespace.QName	Qname
javax.xml.transform.Source	base64Binary
short	short

Supported Types

- Elemental Types
  - Numbers
  - Booleans
  - Strings
  - Date/Time
  - References
  - URLs

Supported Types

- Other Types
  - Types and subtypes
  - Lists
  - Enums
  - Binary data
  - And more…

Key JAXB Classes

- `javax.xml.bind`
  - `JAXBContext`
    - **Provides a context for XML binding via factory methods**
  - `JAXBException`
    - **Root JAXB exception**
  - `Marshaller`
    - **Serializes objects to XML**
  - `Unmarshaller`
    - **Deserializes objects from XML**

Key JAXB Classes

- `javax.xml.stream`
  - `XMLEventReader`
    - **Interface for handling events generated from parsing XML**
  - `XMLInputFactory`
    - **Abstract factory for obtaining an XML reader.**

Key JAXB Annotations

- `@XmlRootElement`
  - Maps a class or an enum type to an XML element.
    - Top level class
    - Enum type
- `@XmlElement`
  - Maps a JavaBean property to a XML element derived from property name
    - JavaBean property
    - Non static, non transient field
    - Within `XmlElements`

Key JAXB Annotations

- `@XmlElementWrapper`
  - Creates a wrapper element around a collection
    - JavaBean collection property
    - Non static, non transient collection field
- `@XmlAttribute`
  - Maps a JavaBean property to a XML attribute
    - JavaBean property
    - Field

Key JAXB Annotations

- `@XmlType`
  - Maps a class or an enum type to a XML Schema type
    - Top level class
    - Enum type
- `@XmlAccessorType`
  - Controls whether fields or JavaBean properties are serialized by default
    - Package
    - Top level class
- Many others

Value Restriction/Validation

- Strings
  - length
  - Regular expression match
- Numbers
  - Max
  - Min
  - Default
- And much more!

Tooling included in JDK

- `xjc`
  - Generates Java classes from XML Schema
- `schemagen`
  - Derives XML Schema from Java classes
- Runtime libraries
- Other tools
  - Ant tasks for `xjc` and `schemagen`
    - jaxb-xjc.jar

JAXB Example Code (Portfolio)

```
public import java.util.List;

import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlElementWrapper;

@XmlRootElement(name="portfolio")
public class Portfolio {
    @XmlElement String id;
    @XmlElementWrapper(name="holdings")
    @XmlElement(name="holding")
    List<Holding> holdings;

    public Portfolio() { }

    public Portfolio(String id, List<Holding> holdings) {
        this.id = id;
        this.holdings = holdings;
    }

    ...
}
```

JAXB Example
Code (Holding)

```
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name="stock")
public class Holding {
    private Stock stock;
    private int shares;

    public Holding() { }

    public Holding(Stock stock, int shares) {
        this.stock = stock;
        this.shares = shares;
    }

    public Stock getStock() { return stock; }

    public void setStock(Stock stock) { this.stock = stock; }

    public int getShares() { return shares; }

    public void setShares(int shares) { this.shares = shares; }

    public String toString() { return Integer.toString(shares) + " shares of " + stock; );
}
```

JAXB Example
Code (Stock)

```
import javax.xml.bind.annotation.XmlAttribute;

public class Stock {
    @XmlAttribute(name="stockSymbol")
    private String stockSymbol;
    private int currentSharePrice;

    public Stock() { }

    public Stock(String stockSymbol, int currentSharePrice) {
        this.stockSymbol = stockSymbol;
        this.currentSharePrice = currentSharePrice;
    }

    public String ticker() { return stockSymbol; }

    public void ticker(String stockSymbol) { this.stockSymbol = stockSymbol; }

    @XmlAttribute(name="currentSharePrice”)
    public int getCurrentSharePrice() { return currentSharePrice; }

    public void setCurrentSharePrice(int currPrice) { this.currentSharePrice = currPrice; }
    ...
}
```

JAXB Example

Code

```
import java.io.File;
import java.io.FileReader;
import java.util.ArrayList;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
import javax.xml.bind.Unmarshaller;
import javax.xml.stream.XMLEventReader;
import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamException;

public class JaxbEx {
    public static void main(String[] args) throws Exception {
        // Create the portfolio object graph
        ArrayList<Holding> holdings = new ArrayList<Holding>();
        holdings.add(new Holding(new Stock("AAPL", 13000), 10000));
        holdings.add(new Holding(new Stock("MSFT", 11000), 1000));
        holdings.add(new Holding(new Stock("F", 5400), 5000));
        Portfolio p = new Portfolio("Abc123", holdings);

        // Print the original
        System.out.println(p);
```

JAXB Example

Code

```
        // Write it out
        JAXBContext context = JAXBContext.newInstance("ex.jaxb");
        Marshaller m = context.createMarshaller();
        File xmlFile = new File("portfolio.xml");
        m.marshal( p, xmlFile);

        // Read it in
        Unmarshaller unmarshaller = context.createUnmarshaller();
        XMLInputFactory xif = XMLInputFactory.newInstance();
        FileReader isr = new FileReader(xmlFile);
        XMLEventReader xer = xif.createXMLEventReader(isr);
        Portfolio px = (Portfolio) unmarshaller.unmarshal(xer);
        // Print the retrieved object graph
        System.out.println(px);
    }
}
```

JAXB Example

Code (package-info.java)

```
@javax.xml.bind.annotation.XmlSchema
(elementFormDefault=javax.xml.bind.annotation.XmlNsForm.UNQUALIFIED)
package xmlser.jaxb;
```

JAXB Example

Serialized File

```
{"id":"Abc123","holdings":[
    {"stock":{"stockSymbol":"AAPL","currentSharePrice":13000},
     "shares":10000},
    {"stock":{"stockSymbol":"MSFT","currentSharePrice":11000},
     "shares":1000},
    {"stock":{"stockSymbol":"F","currentSharePrice":5400},
     "shares":5000}
]}
```

JAXB Example

Schema

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema elementFormDefault="unqualified" version="1.0" xmlns:xs="http://www.w3.org/2001/
XMLSchema">

  <xs:element name="portfolio" type="portfolio"/>

  <xs:element name="stock" type="holding"/>

  <xs:complexType name="holding">
    <xs:sequence>
      <xs:element name="shares" type="xs:int"/>
      <xs:element name="stock" type="stock" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="stock">
    <xs:sequence/>
    <xs:attribute name="stockSymbol" type="xs:string"/>
    <xs:attribute name="currentSharePrice" type="xs:int" use="required"/>
  </xs:complexType>
```

JAXB Example

Schema

```
  <xs:complexType name="portfolio">
    <xs:sequence>
      <xs:element name="id" type="xs:string" minOccurs="0"/>
      <xs:element name="holdings" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="holding" type="holding" minOccurs="0" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

JAXB Example

Serialized File

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<portfolio>
  <id>Abc123</id>
  <holdings>
    <holding>
      <shares>10000</shares>
      <stock currentSharePrice="13000" stockSymbol="AAPL"/>
    </holding>
    <holding>
      <shares>1000</shares>
      <stock currentSharePrice="11000" stockSymbol="MSFT"/>
    </holding>
    <holding>
      <shares>5000</shares>
      <stock currentSharePrice="5400" stockSymbol="F"/></holding>
    </holdings>
</portfolio>
```