External Data Representation and Marshalling

Information in programs consists of data structures

Information in messages consists of sequences of bytes

Data structures must be **FLATTENED** before transmission

Remember Little endians and big endians and Network Byte Order?

- Sockets
  - ntohs() ntohl()
  - htons(), htonl()
- Integers span multiple bytes in memory
  - 16-bit integers span 2 bytes
  - 32-bit integers span 4 bytes
  - The receiver must honor the same byte ordering used by the sender, otherwise, caos!
    - The Network format for integers is Network Byte Order
    - Translate integers to Network Byte Order before transmitting
    - Translate from Network Byte Order to the receiving hardware ordering before using the received data
- Marshalling of an integer consists of *consistently* sending its contituent bytes so that the receiver can recover the same integer that was sent
- Likewise, Complex data types must be **marshalized**: Objects, Classes, Interfaces, etc.

Marshalization in Java

Java object serialization

- In Java RMI, objects and primitive values may be passed as arguments and return values from remote method invocations
  - One object's class passed as an argument to a remote method must implement java.io.Serializable
  - This interface has no methods (It's a "marker interface")
  - Allows instances of a class to be serialized
  - ObjectOutputStream

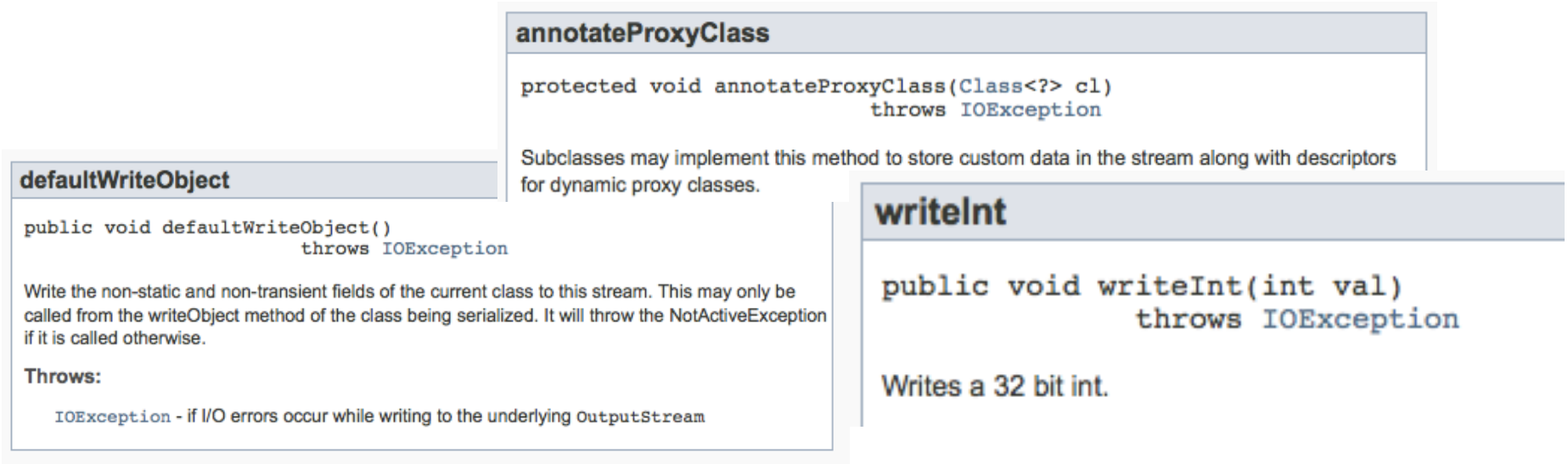
Marshalization in Java

Java object serialization

Serialization: Flattening an object into a serial form suitable for transmission or disk storage

Deserialization: Reinstating an object from its serialized byte stream

- In Java, the deserializing process has NO previous knowledge of the types of the objects included in the serialized form
- The serialized form itself contains information about the serialized objects



annotateProxyClass

```
protected void annotateProxyClass(Class<?> cl)
                                  throws IOException
```

Subclasses may implement this method to store custom data in the stream along with descriptors for dynamic proxy classes.

defaultWriteObject

```
public void defaultWriteObject()
                           throws IOException
```

Write the non-static and non-transient fields of the current class to this stream. This may only be called from the writeObject method of the class being serialized. It will throw the NotActiveException if it is called otherwise.

Throws:

`IOException` - if I/O errors occur while writing to the underlying `OutputStream`

writeInt

```
public void writeInt(int val)
                    throws IOException
```

Writes a 32 bit int.

Marshalization in Java

Java objects contain references to other objects

A Java object contains primitive values AND object references

- When the object is serialized all **values** are serialized
- When the object is serialized all its **object references** are serialized
  - References are serialized as **Handles**
    - Handles are references to other objects within the serialized format
- Serialization is a **recursive** procedure
- Each class is assigned a Handle and is written only once to the stream

Marshalization in Java

Java objects primitive-type instance values

- int, char, boolean, etc
- These are written to the output stream in a portable data format by using methods of ObjectOutputStream:
  - writeInt(), writeChar(), writeBoolean
  - UTF-8 (UNICODE Transformation Format)
    - Unicode 1-byte representation for ASCII
- Serialization is normally performed by middleware
- Ocassionally the app programmer may have to write serialization
  - Consult the Java Tutorial (Serialization) for further details

Figure 4.9
Indication of Java serialized form

```
• Person p = new Person("Smith", "London", 1984);
```

Serialized values

Person	8-byte version number		h0
3	int year	java.lang.String name:	java.lang.String place:
1984	5 Smith	6 London	h1

Explanation

class name, version number

number, type and name of instance variables

values of instance variables

The true serialized form contains additional type markers; h0 and h1 are handles (references to other objects)

Marshalization in Java

Java Reflection in Serialization

- **Reflection:** Ability of a class of reporting its properties
  - Which methods it has
  - Which fields it has
  - Which constructors
- Reflection allows us to carry out Serialization in a completely generic manner
  - With no previous knowledge about the properties of any object
  - No need to have its source code
- Serialization uses Java Reflection to find out the name of an object's class to be serialized, its types and its values

Marshalization in Java

Java Reflection in DeSerialization

1. The class name in the serialized form is used to CREATE a NEW CLASS
2. Create a new constructor with arguments from serialized form
3. The **constructor** is executed to create the new object and its instance variables from the serialized form

Marshalization in Java

Extensible Markup Language (XML)

- Defined by W3C
- A markup language
  - Textual encoding for text itself and its structure
  - Structured WEB documents
- HTML -> Appearance of web pages
  - XHTML is html compatible with XML
- XML -> Structured documents for the web
- Data in XML
  - Markup strings, tags
  - Define the logical structure of a document

Marshalization in Java

XML instance file example

```
<person id="12345678">
        <name> Pedro </name>
        <familyname> Pérez </familyname>
        <year>1984</year>
        <!– a comment -->
</person>
```

Marshalization in Java

Uses of XML

- Clients consume **Web Services** by exchanging XML data with the WS point of access
  - Marshalling with XML
- Also, Web Services **interfaces** are specified in XML
- Other uses include data archiving and retrieval

Marshalization in Java

Extensible

- HTML tags are fixed
- Users can make their own tags in XML
  - Tags need be published so interacting programs can communicate
- **CORBA CDR is not self-describing**
  - Because both interacting entities must have prior knowledge of the information being exchanged
- To resolve conflicts with naming of tags and provide meaning:
  - Namespaces