Java Object Communication

Instructor: Prasun Dewan (FB 150, dewan@unc.edu)
COMMUNICATION SO FAR: BYTE COMMUNICATION

Sender -> Receiver

bytes

Sockets
NIO
The receiving site creates in its memory a copy of a block of memory at the sending site.
Object Communication: Values of Multiple Types

Object communication even though communicating data structures that include objects and primitive values (Data structure communication)

Motivation and application?
Motivation for Non Byte Communication

- Originally, RPC was implemented directly on byte communication
- Java developed idea of object data communication

Remote procedure call (RMI)

- Blocking stream object communication (Object Stream, )
- Blocking byte communication (Sockets)

Separate layer can imply more flexibility
Data Communication Implemented by Who

- Infrastructure (Language, Library)
- Programmer (of Communicated Objects)
- Both (Programmer code called by Infrastructure)
ObjectOutputStream socketOut = new ObjectOutputStream(socket.getOutputStream());
socketOut.writeObject(3);
ObjectInputStream socketIn = ObjectInputStream(socket.getInputStream());
Integer readVal = (Integer) socketIn.readObject();
**Object Communication and External Data Representation (XDR)**

- **short** `s = 3`

**Little Endian**
- `s` (3)
- `0`

**Big Endian**
- `0`
- `3`

- **A value that has a direct XDR that does not have to be composed from other XDR’s**
- **Primitive, Wrapper**

**x86**

**Host-independent eXternal Data Representation**

**Receiver’s copy a clone of the sender’s copy?**

**Receiver’s copy semantically equivalent to sender’s copy**
**Structure Communication: Serialized XDR**

Serialization: gathering a program data structure possibly scattered in memory into a serial byte sequence (containing XDRs of primitive values)

Deserialization: Reconstructing byte sequence into equivalent data structure, possibly scattering the value

```java
Byte[] a = {3}
```
**Sending Graphs**

Received data structure is isomorphic to the sender’s data structure

And is of the same type

Corresponding leaf primitive values represent the same abstract value
Non Tree Structures

Object[] objects = new Object[1];
objects[0] = objects
**Programmer-Defined Class:**
ABMISpreadsheet

```java
public class ABMISpreadsheet implements BMISpreadsheet {
    double height = 1.77;
    double weight = 75;
    public double getWeight() { return weight; }
    public void setWeight(double newWeight) {
        weight = newWeight;
    }
    public double getHeight() { return height; }
    public void setHeight(double newHeight) {
        height = newHeight;
    }
    public double getBMI() { return weight / (height * height); }
}
```
PHYSICAL STRUCTURE

BMISpreadsheet b = new ABMISpreadsheet()

Physical structure defined by instance variables stored in memory

Each component of an object is a value stored in an instance variable declared in class

Internal nodes labeled by class name

Leaf nodes labeled type and value

Every object serializable?
**Location-Dependent Objects**

Requirement: Receiver’s copy semantically equivalent to sender’s copy

<table>
<thead>
<tr>
<th>System.in</th>
<th>Socket s</th>
<th>File f</th>
<th>TextField t</th>
</tr>
</thead>
</table>

Several objects have location-dependent semantics

<table>
<thead>
<tr>
<th>The OS or some other external software such as window system keeps state that define full semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>The external software may not exist at the other end, may not be known to serialization system, and may not be willing or able to create equivalent state</td>
</tr>
<tr>
<td>e.g. file, text field, socket, System.in</td>
</tr>
</tbody>
</table>
NOT ALL OBJECTS ARE SERIALIZABLE

Sending an isomorphic copy does not guarantee semantic equivalence.

How to tell some generic serialization system that some type is not serializable?

Programmer used some directive.

Ideally with language support.
SHALLOW TAGGING

An object is serialized only if it is tagged as serializable

Structure is often determined at runtime

Can add a serializable or non serializable component

List

byte 3

ABMISpreadsheet

height

weight

double 1.77
double 75
Deep Tagging

An object is serialized only if every component of its (logical or physical) structure is tagged as serializable.

Not serializable even though top object is Serializable

Runtime exception
**INTERFACE BASED TAG**

Object O is Serializable if O instance of Serializable

Serializable is a an empty interface – has no methods

Any class implements it if it says so
import java.io.Serializable;

public interface BMISpreadsheet extends Serializable {
    double getWeight();
    void setWeight(double newWeight);
    double getHeight();
    void setHeight(double newHeight);
    double getBMI();
}
public class ABMISpreadsheet implements BMISpreadsheet {
    double height = 1.77;
    double weight = 75;
    public double getWeight() { return weight; }
    public void setWeight(double newWeight) {
        weight = newWeight;
    }
    public double getHeight() { return height; }
    public void setHeight(double newHeight) {
        height = newHeight;
    }
    public double getBMI() { return weight/(height*height); }
}
public class ADistributedBMISpreadsheet extends ABMISpreadsheet {
    Socket peer;
    public Socket getSocket();
    ...
}

Instance of ADistributedBMISpreadsheet is instance of Serializable
FUNDAMENTAL PROBLEM WITH INHERITED TAG

```
import java.io.Serializable;
public interface BMISpreadsheet extends Serializable {
    double getWeight();
    void setWeight(double newWeight);
    double getHeight();
    void setHeight(double newHeight);
    double getBMI();
}
```

When a type is created, can guarantee behavior of only instances of that type

Not of instances of yet to be created subtypes
C# Attributes

An object is serializable if its class holds the non inheritable Serializable attribute

```
[Serializable]
public class ABMISpreadsheet : BMISpreadsheet {
    ...
}
```
**Semi-Serialized Structures?**

```java
public class ADistributedBMISpreadsheet extends ABMISpreadsheet {
    Socket peer;
    public Socket getSocket();
    ...
}
```

*Send structure without socket?*

*Socket may be constructed from serializable components such as host name and port*
<table>
<thead>
<tr>
<th>Semi-Serialized Structures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send part of the logical structure to another site?</td>
</tr>
<tr>
<td>Certain parts may be derived from other parts</td>
</tr>
<tr>
<td>Not sending these parts more efficient</td>
</tr>
<tr>
<td>These parts may not be serializable (e.g. some socket)</td>
</tr>
</tbody>
</table>
**Derived Structure in BMI Example?**

```java
public class AnotherBMISpreadsheet implements BMISpreadsheet {
    double height = 1.77;
    double weight = 75;
    double bmi = calculateBMI();

    public double getHeight() { return height; }
    public void setHeight(double newHeight) {
        height = newHeight;
        bmi = calculateBMI();
    }

    public double getWeight() { return weight; }
    public void setWeight(double newWeight) {
        weight = newWeight;
        bmi = calculateBMI();
    }

    public double getBMI() { return bmi; }
    double calculateBMI() {
        return weight / (height * height);
    }
}
```

How does infrastructure know what is derived?

Annotations, keywords, attributes can be used to tag
public class AnotherBMISpreadsheet implements BMISpreadsheet{
    double height = 1.77;
    double weight = 75;
    transient double bmi = calculateBMI();
    public double getHeight() { return height; }
    public void setHeight(double newHeight) {
        height = newHeight;
        bmi = calculateBMI();
    }
    public double getWeight() { return weight; }
    public void setWeight(double newWeight) {
        weight = newWeight;
        bmi = calculateBMI();
    }
    public double getBMI() { return bmi; }
    double calculateBMI() {
        return weight/(height*height);
    }
}
PARTIALLY SERIALIZED OBJECTS

BMISpreadsheet b = new AnotherBMISpreadsheet();

Constructors usually used to initialize variables
Both transient and non transient
How to initialize only transient when infrastructure is deserializing?
public class AnotherBMISpreadsheet implements BMISpreadsheet{
    double height = 1.77;
    double weight = 75;
    transient double bmi = calculateBMI();
    public double getHeight() {return height;}
    public void setHeight(double newHeight) {
        height = newHeight;
        bmi = calculateBMI();
    }
    public double getWeight() {return weight;}
    public void setWeight(double newWeight) {
        weight = newWeight;
        bmi = calculateBMI();
    }
    public double getBMI() { return bmi;}
    double calculateBMI() {
        return weight/(height*height);
    }
    public void initSerializedObject() {
        bmi = calculateBMI();
    }
}
A class can select which subobjects are serialized to serialize it.

Infrastructure determines if an object has been visited before and adds type names if necessary.
ObjectOutputStream socketOut = new ObjectOutputStream(socket.getOutputStream());
socketOut.writeObject("hello world");
**Serializable Pattern based Functions**

- **Serializable**
  - `private readObject(ObjectInputStream s)`
  - `private writeObject(ObjectOutputStream s)`

  These methods are called if they exist and are also used to initialize transients structures.

- **ObjectInputStream**
  - `defaultReadObject()`

- **ObjectOutputStream**
  - `defaultWriteObject()`

Default methods (de)serialize non transients.
public class AnotherBMISpreadsheet implements BMISpreadsheet{
    double height = 1.77;
    double weight = 75;
    transient double bmi = calculateBMI();
    public double getHeight() {return height;}
    public void setHeight(double newHeight) {
        height = newHeight;
        bmi = calculateBMI();
    }
    public double getWeight() {return weight;}
    public void setWeight(double newWeight) {
        weight = newWeight;
        bmi = calculateBMI();
    }
    public double getBMI() { return bmi;}
    double calculateBMI() {
        return weight/(height*height);
    }
    public void initSerializedObject() {
        bmi = calculateBMI();
    }
}
private void readObject(ObjectInputStream stream) {
    try {
        stream.defaultReadObject();
        initSerializedObject();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
Implicit System-Defined Write Object

```java
private void writeObject(ObjectInputStream stream)
    throws IOException{
    stream.defaultWriteObject();
}

Default behavior of Java
```
public class ANamedBMISpreadsheet extends AnotherBMISpreadsheet implements NamedBMISpreadsheet{
    public static final int HIGH_BMI = 27;
    transient boolean isOverWeight = calculateOverWeight();
    String name = "";
    boolean calculateOverWeight() { return bmi > HIGH_BMI; }
    public boolean isOverWeight() {
        return isOverWeight;
    }
    public String getName() { return name; }
    public void setName(String newValue) {
        name = newValue;
    }
    public void initSerializedObject() {
        super.initSerializedObject();
        isOverWeight = calculateOverWeight();
    }
}
private void readObject(ObjectInputStream stream) {
    try {
        stream.defaultReadObject();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

defaultReadObject() reads declared variables

No need to call super.readObject()

Java calls explicit or implicit readObject() on each super class

Cannot call super() : non private methods not recognized

Would cause multiple reads

Method not really needed as it is the default readObject()
private void readObject(ObjectInputStream stream) {
    try {
        stream.defaultReadObject();
        initSerializedObject();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private void readObject(ObjectInputStream stream) {
    try {
        stream.defaultReadObject();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
**EXTERNALIZABLE INTERFACE BASED FUNCTIONS**

- **Externalizable**
  - `readExternal (ObjectIn)`
  - `writeExternal(ObjectOut)`

These methods must exist if interface implemented.

**ObjectIn**
- `readObject(Primitive)(Object (Primitive))`

**ObjectOut**
- `writeObject(Primitive)(Object(Primitive))`

No default read and write of Object.
Both public interface methods must be implemented and no default read or write methods.
public void readExternal(ObjectInput in) {
    try {
        super.readExternal(in);
        name = in.readLine();
    } catch (Exception e) { e.printStackTrace(); }
}
public void writeExternal(ObjectOutput out) {
    try {
        super.writeExternal(out);
        out.write(name.getBytes());
    } catch (Exception e) { e.printStackTrace(); }
}

Can and must call super on public member
public class AnObjectHistory<ElementType> implements ObjectHistory<ElementType>
{
    public final int MAX_SIZE = 50;
    Object[] contents = new Object[MAX_SIZE];
    int size = 0;
    public int size() { return size; }
    public ElementType get(int index) {
        return (ElementType) contents[index];
    }
    boolean isFull() { return size == MAX_SIZE; }
    public void add(ElementType element) {
        if (isFull())
            System.out.println("Adding item to a full history");
        else {
            contents[size] = element;
            size++;
        }
    }
}
public class AnObjectHistory<ElementType> implements ObjectHistory<ElementType>

    transient public final int MAX_SIZE = 50;
    transient Object[] contents = new Object[MAX_SIZE];
    int size = 0;
    public int size() { return size; }
    public ElementType get(int index) {
        return (ElementType) contents[index];
    }
    boolean isFull() { return size == MAX_SIZE; }
    public void add(ElementType element) {
        if (isFull())
            System.out.println("Adding item to a full history");
        else {
            contents[size] = element;
            size++;
        }
    }
AN OBJECT HISTORY (INITIALIZATION AND EXTENSION)

```java
cpyriate void writeObject(ObjectOutputStream stream) {
    try {
        stream.defaultWriteObject();
        for (int i = 0; i < size; i++)
            stream.writeObject(contents[i]);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private void readObject(ObjectInputStream stream) {
    try {
        stream.defaultReadObject();
        contents = new Object[MAX_SIZE];
        for (int i = 0; i < size; i++)
            contents[i] = stream.readObject();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

ObjectHistory objectHistory = new AnObjectHistory();
objectHistory.add(objectHistory);
```
AnObjectHistory (Initialization and Extension)

```java
public void writeExternal(ObjectOutput out) throws IOException {
    out.writeInt(size);
    for (int i = 0; i < size; i++)
        out.writeObject(contents[i]);
}

public void readExternal(ObjectInput in) throws IOException,
    ClassNotFoundException {
    contents = new Object[MAX_SIZE];
    size = in.readInt();
    for (int i = 0; i < size; i++)
        contents[i] = in.readObject();
}
```
SERIALIZABLE VS. EXTERNALIZABLE

```java
private void readObject(ObjectInputStream stream) {
    try {
        stream.defaultReadObject();
        initSerializedObject();
    } catch (Exception e) { e.printStackTrace(); }
}

public void readExternal(ObjectInput in) {
    try {
        height = in.readDouble();
        weight = in.readDouble();
        initSerializedObject();
    } catch (Exception e) { e.printStackTrace(); }
}

public void writeExternal(ObjectOutput out) {
    try {
        out.writeDouble(height);
        out.writeDouble(weight);
    } catch (Exception e) { e.printStackTrace(); }
}
```
INHERITANCE AND SERIALIZABLE

Another BMISpreadsheet

implements Serializable

extends ANamedBMI Spreadsheet

double height
double weight
double bmi

String name

If object is instance of Serializable all of its non transient variables are serialized (by default)
INHERITANCE AND SERIALIZABLE

Another BMISpreadsheet

implements

Serializable

extends

ANamedBMI Spreadsheet

double height

double weight

double bmi

String name

Not Serializable exception when serializing instance of either class
INHERITANCE AND SERIALIZABLE

Variables of super classes that do not implement Serializable are not serialized.

Serialization looks at implements rather than instance of relation.
If object is instance of Serializable all of its non transient variables are serialized (by default)

If object is instance of Serializable all of its non transient variables declared in classes that implement Serializable are serialized
## Serializable vs. Externalizable

<table>
<thead>
<tr>
<th>Private Custom Methods</th>
<th>Public Custom Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Can make mistake in method signature</td>
<td>Must implement both methods</td>
</tr>
<tr>
<td>Can invoke default methods</td>
<td>No default methods</td>
</tr>
<tr>
<td>Can only initialize transients</td>
<td>Serialize both transient and non transients</td>
</tr>
<tr>
<td>Sends field names, field types and declaring super classes</td>
<td>Sends only values explicitly written by programmer</td>
</tr>
<tr>
<td>Less efficient</td>
<td>Cannot read data without class at receiver’s site</td>
</tr>
</tbody>
</table>
EXTRA SLIDES
COUPLING OF TRANSIENTS INITIALIZATION AND EXTENSIBILITY

Extensibility and initialization of transients is coupled

Often they are done together

Only one set of abstraction learnt

Disadvantages?
COUPLING OF INITIALIZATION AND EXTENSIBILITY

Serializable: App must be aware of and call defaultReadObject()

```
private void readObject (ObjectInputStream stream) {
    try {
        stream.defaultReadObject();
        initSerializedObject();
    } catch (Exception e) { e.printStackTrace(); }
}
```

Both: App must know about input stream and deserialization exception handling

```
public void readExternal (ObjectInput in) {
    try {
        height = in.readDouble();
        weight = in.readDouble();
    } catch (Exception e) { e.printStackTrace(); }
}
```

Externalizable: App must do complete serialization and deserialization

```
public void writeExternal (ObjectOutput out) {
    try {
        out.writeDouble(height);
    } catch (Exception e) { e.printStackTrace(); }
}
```

Separating interface (method signatures) for extensibility and transient initialization does not have this problem
COUPLING OF PROCESSING AND EXTENDED SERIALIZATION

Serialization extension must be done by the class being serialized

Allows extension to easily and efficiently access physical structure (without reflection)
**Inheriting Internal Serializer**

Cannot add serializer for class without source code

Must inherit and change references and not reuse code

**Diagram:**

- ABMISpreadsheet
- ADistributedBMISpreadsheet
- ACustomSerializedDistributedBMISpreadsheet
- ACustomSerializedBMISpreadsheet

**Relationships:**

- ABMISpreadsheet extends ADistributedBMISpreadsheet
- ADistributedBMISpreadsheet extends ACustomSerializedBMISpreadsheet
- No sharing of code!
# Coupling of Processing and Extended Serialization

<table>
<thead>
<tr>
<th>Cannot have multiple serializers for different contexts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text vs Binary, Logical vs. Physical, Mobile vs. Desktop, Single Language vs. Multiple Language</td>
</tr>
<tr>
<td>Binary serializer of physical structures that can be extended by internal routines</td>
</tr>
<tr>
<td>Text serializer of logical structures that is driven by serialization directives but cannot be extended</td>
</tr>
</tbody>
</table>
Java Serialization and Object Communication

Object Socket | Other applications (Object File)
---|---
Java Stream Serialization
Stream
Byte Socket | Other Byte Stores (File)

No (direct) serialization support for NIO