Reflection Applied: Serialization

CPSC 501: Advanced Programming Techniques
Winter 2025

Jonathan Hudson, Ph.D
Assistant Professor (Teaching)
Department of Computer Science
University of Calgary

Wednesday, March 5, 2025

UNIVERSITY OF CALGARY

What the cereal?

UNIVERSITY OF CALGARY

Serialization

- **Serialization:** the process of converting an object into a stream of bytes
    - Format can be binary,
    - or human-readable (text)

Serialization

- The byte stream may be:

  1. Stored to a file or database
     - Enables **object persistence**

  2. Transmitted to another program
     - For **remote method invocation** (RMI)

  3. Transmitted across a network
     - For **distributed objects**

De-serialization

- **Deserialization:** converts the byte stream (or text) into a recreation of the original object
    - i.e. its clone

De-serialization

- **Deserialization:** converts the byte stream (or text) into a recreation of the original object
  - i.e. its clone
  - You will not maintain exact object jvm identity (unique id assigned to each object made in java)
    - You will want identity of objects to be defined by
    - equals()
    - hashCode()
  - You can maintain relative object jvm identity

Serialization

- When you serialize an object, you are saving its **state**
    - i.e. the current value of all its instance variables

- To build a general-purpose serialization system, you need access to an object's metadata
    - i.e. requires reflection

Java cereal

Coffee in my cereal?

Java Serialization

- Java has a Serializable marker interface
  - If implemented by a class, its instances can be serialized automatically to a binary stream

  - Just use interface

java class MyClass implements Serializable

  - (optional) can indicate object versioning with class variable

private static final long serialVersionUID=42L;

Python does serialization using pickle library (custom objects need to design it in (like numpy does)

Java Serialization

- Java has a Serializable marker interface
    - java.io.ObjectInputStream
    - java.io.ObjectOutputStream
    - Let you read/write Serializable interface classes automatically to and from streamable locations

Java Serialization

As simple as this?

```
private static void write() throws Exception {
   FileOutputStream fos = new FileOutputStream(filename);
   ObjectOutputStream oos = new ObjectOutputStream(fos);
   oos.writeObject(new MyClass("name"));
}

private static void read() throws Exception {
   FileInputStream fis = new FileInputStream(filename);
   ObjectInputStream ois = new ObjectInputStream(fis);
   MyClass ob = (MyClass) ois.readObject();
   System.out.println(ob.getName());
}
```

Java Serialization

SerialVersionUID matters

```java
private static void write() throws Exception {
    FileOutputStream fos = new FileOutputStream(filename);
    ObjectOutputStream oos = new ObjectOutputStream(fos);
    oos.writeObject(new MyClass("name"));
}

private static void read() throws Exception {
    FileInputStream fis = new FileInputStream(filename);
    ObjectInputStream ois = new ObjectInputStream(fis);
    MyClass ob = (MyClass) ois.readObject();
    System.out.println(ob.getName());
}
```

```java
public class MyClass implements Serializable {

    private String name;
    private static final long serialVersionUID = 1L;
//    private static final long serialVersionUID = 2L;

    public MyClass(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}
```

Java Serialization

So does sub-class having UIDs

```
private static void write() throws Exception {
   FileOutputStream fos = new FileOutputStream(filename);
   ObjectOutputStream oos = new ObjectOutputStream(fos);
   oos.writeObject(new MyClass("name"));
}

private static void read() throws Exception {
   FileInputStream fis = new FileInputStream(filename);
   ObjectInputStream ois = new ObjectInputStream(fis);
   MyClass ob = (MyClass) ois.readObject();
   System.out.println(ob.getName());
}
```

```
public class MyClass implements Serializable {

   private static final long serialVersionUID = 1L;
   private String name;
   private OtherClass other;


public class OtherClass {}
```

General Mills Cereal

Coffee in my cereal?

General Purpose Serialization

- However a custom, general-purpose serializer that serializes to a text stream has several advantages:
  - The stream is easily read or modified with a text editor
  - Can send objects to a non-Java platform
  - Can be applied to third-party classes that don’t implement Serializable

XML

- XML (eXtensible Markup Language) is an ideal format for the text stream
  - Is self-describing
  - Encodes structured, hierarchical data
  - Is well supported with facilities that do parsing, presentation, etc.
    - E.g. via libraries DOM, JDOM, SAX

XML Structure

- XML uses pairs of tags to create an element

- Start tag: **<tag-name>**
- End tag: **</tag-name>**

- ***Content*** goes between the tags
- ***Child elements*** can be nested inside an element

- E.g.

```
<zoo>
        <animal>Panda</animal>
        <animal>Giraffe</animal>
    </zoo>
```

Reflective Serialization

- An **empty element tag** has the form

  <tag-name />

  - Equivalent to: **<tag-name></tag-name>**

- A start tag may also contain name-value pairs called **attributes**
  - Form:

  <tag-name attribute-name="attribute-value">

  - E.g.

  <zoo location="Paris" rank="12">

Reflective Serialization

- A file or stream of well-formed XML is called a document

- Each document must contain **one** root element
  - Contains all other content

Reflective Serialization

- We could do serialization by making code that dumps and loads objects by hand for each class
- (I’ve done this and it is quite feasible for 1-5 object structures)
- Doesn’t scale

```
public Node toElement(Document document) {
   Element element = document.createElement("MyClass");
   element.setAttribute("name", name);
   element.appendChild(other.toElement(document));
   return element;
}

public static MyClass createObject(Node node) {
   MyClass ob = new MyClass(node.getAttributes().getNamedItem("name").getNodeValue());
   ob.other = OtherClass.createObject(node.getChildNodes().item(0));
   return ob;
}
```

Reflective Serialization

- Using **reflection** to do serialization offers several advantages:

1. Does not require invasive changes to hundreds of classes

2. Works with all in-house, third-party, and JDK classes
   - And any classes created in the future

3. Debugging and maintenance is centralized to the serialization code

One two step

UNIVERSITY OF CALGARY

Reflective Serialization

- The reflective serializer should serialize any type of object passed in as a parameter
- Basic design:
  1. Give the object a unique identifier number
     - Could be done with java.util.IdentityHashMap
  2. Get a list of all the object's fields
  3. Uniquely identify each field with its (Declaring class, Field name)
  4. Get the value for each field
     1. If a primitive, simply store it so it can be easily retrieved
     2. If a non-array object, recursively serialize the object
     3. If an array object, serialize it as unique array type

Dynamic

Dynamic Loading

- A ordinary class can be loaded at runtime using

```
public static Class forName(String className)
```

  - E.g.

```
String name = . . .
Class classObject = Class.forName(name);
```

  - Throws **ClassNotFoundException** if the corresponding .class file is not found on the classpath

Dynamic Loading - Arrays

- Array classes do not have a .class file
  - i.e. do not have a "normal" class name
  - Are generated as needed by the JVM

Encoding	Element type
B	byte
C	char
D	double
F	float
I	int
J	long
L<element-type>	reference type
S	short
Z	boolean

Dynamic Loading

- For each dimension of the array, use a [
- Then add the element type code
- E.g.
  - 1D int array: [I
  - 2D float array: [[F
  - 1D array of objects: [Ljava.lang.String

Reverse it

Step two one

Reflective Deserialization

- Recreates objects from a byte stream
  - Requires:
    - Dynamic loading of classes
    - Reflective instantiation of objects
    - Setting fields reflectively

- Basic design:
  1. Get a list of objects stored in the XML document
  2. For each object, create an uninitialized instance:
     i. Dynamically load its class using forName()
     ii. Create an instance of the class
     iii. Associate the new instance with the object's unique identifier number using a table

Reflective Deserialization

3. Assign values to all instance variables in each non-array object:
   i. Get a list of the child elements
   ii. Iterate through each field in the list
      a. Find the name of its declaring class
      b. Load the class dynamically
      c. Find the field name
      d. Use getDeclaredField() to find Field metaobject
      e. Initialize the value of the field using set()
4. Array objects need you to getComponentType to create the array and then a loop to set each entry of the new array

Onward to … Java proxies.

Jonathan Hudson
jwhudson@ucalgary.ca
https://pages.cpsc.ucalgary.ca/~jwhudson/

UNIVERSITY OF CALGARY