

Reflection: Serialization

CPSC 501: Advanced Programming Techniques
Fall 2020

Jonathan Hudson, Ph.D
Instructor
Department of Computer Science
University of Calgary

Wednesday, August 5, 2020



What the cereal?

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;

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

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

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

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`
 - `IdentityHashMap` uses `==` instead of `equals()`
 - Choice to use it or `HashMap` depends on whether you want to maintain exact relative object connections

Reflective Serialization

2. Get a list of all the object's fields
 - Of all visibilities
 - Use `getDeclaredFields()` and traverse the inheritance hierarchy
 - Filter out static fields

3. Uniquely identify each field with its
 - Declaring class
 - Field name

Reflective Serialization

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
 - Use the new object's unique id number as a reference
 - Store the reference as the field value in the originating object
 - Don't serialize an object more than once
 - Occurs when you have several references to the same object
 3. If an array object, serialize it
 - Then serialize each element of the array
 - Use recursion if the element is an object

Readings

- Forman & Forman Chapter 2
- www.jdom.org
- Java API: `java.util.IdentityHashMap`

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
- Array classes are named using codes:

Dynamic Loading

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`

Dynamic Loading

- Array classes can be loaded using

forName()

- E.g. array of String objects

```
Class classObject;
```

```
classObject = Class.forName("[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
 - Use `getRootElement()` from Document class, and `getChildren()` from Element class

Reflective Deserialization

2. For each object, create an uninitialized instance:
 - i. Dynamically load its class using `forName()`
 - The class name is an attribute of the object element
 - ii. Create an instance of the class
 - If a non-array object, get the declared no-arg constructor, then use `newInstance()`
 - May need to `setAccessible(true)`
 - If an array object, use `Array.newInstance(...)`
 - Use `getComponentType()` to find element type
 - The length is an attribute of the object element

Reflective Deserialization

- iii. Associate the new instance with the object's unique identifier number using a table
 - `java.util.HashMap` is ideal
 - The id is the key
 - The object reference is the value
 - The id is an attribute of the object element

Reflective Deserialization

3. Assign values to all instance variables in each non-array object:
 - i. Get a list of the child elements
 - Use `getChildren()` from Element class
 - Each child is a field of the object
 - ii. Iterate through each field in the list
 - a. Find the name of its declaring class
 - Is an attribute of field element
 - b. Load the class dynamically

Reflective Deserialization

- c. Find the field name
 - Is an attribute of field element
- d. Use `getDeclaredField()` to find Field metaobject
- e. Initialize the value of the field using `set()`
 - If a primitive type, use the stored value (use `getText()` and create appropriate wrapper object)
 - If a reference, use the unique identifier to find the corresponding instance in the table
 - May need to `setAccessible(true)`

Reflective Deserialization

- Array objects are treated specially:
 - Find the element type with `getComponentType()`
 - Iterate through each element of the array
 - Set the element's value using `Array.set()`
 - As above, treat primitives differently than references

Onward to ... Java intercession.

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



UNIVERSITY OF
CALGARY