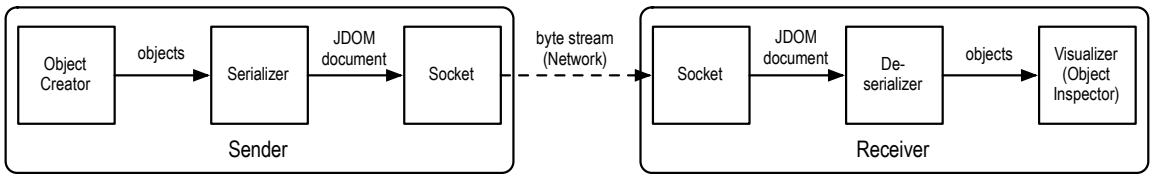
Advanced Programming Techniques
Assignment 3

Distributed Objects using Reflective Serialization/Deserialization

The goal of this assignment is to create two programs (on two separate computers) that communicate with each other over a network connection (see diagram below). The *Sender* program will create one or more arbitrary objects, serialize these objects into a JDOM document, and then send this document as a stream of bytes to the *Receiver* program using a socket connection. The *Receiver* program will convert the incoming byte stream into a JDOM document, deserialize the document into objects, and display the objects to screen.



The Object Creator

This part of your system will create arbitrary objects under control of the user. Allow the user to create one or more objects from a selection of objects using some sort of text-based menu system or GUI. You must demonstrate that your system can handle the following kinds of objects:

- A simple object with only primitives for instance variables. The user of your program must also be able to set the values for these fields.
- An object that contains references to other objects. Of course, these other objects must also be created at the same time, and their primitive instance variables must be settable by the user. Your program must also be able to deal with circular references (i.e. objects connected in a graph).
- An object that contains an array of primitives. Allow the user to set the values for the array elements to arbitrary values.
- An object that contains an array of object references. The other objects must also be created at the same time.
- An object that uses an instance of one of Java's collection classes to refer to several other objects. These objects, too, must be created at the same time.

The Serializer

Serialization will be implemented in a Java class called Serializer, and will be invoked using the method:

public org.jdom.Document serialize(Object obj)

This method will serialize the complete state of the object passed in as parameter, and produce an XML document that can be stored to file, printed out, or sent over a network connection. Use the facilities provided by JDOM to help you do this, in particular the Document and XMLOutputter classes.

The XML document must have one root element with the tag-name *serialized*. The object, and any other objects that may need to be serialized, will be nested within the root element, listed one after the other. For example:

```
<serialized>
      <object . . .>
            . . .
      </object>
      <object . . .>
```

```
            . . .
      </object>
</serialized>
```

The element *object* will have two attributes: *class* and *id*. The *class's* value will be the name of the class of the object, and the *id's* value will be object's unique identifier number, most likely created using IdentityHashMap. For example:

```
      <object class="Zoo" id="0">
```

Nested within each *object* element will be 0 or more *field* elements. Each field element will have two attributes: *name* and *declaringclass*. The *name's* value will be the name of the field, and the *declaringclass's* value will be the name of the field's declaring class. For example:

```
            <field name="city" declaringclass="Zoo">
                  . . .
            </field>
```

If the type of the field is a primitive, store the primitive as content of a *value* element. For example:

```
            <field . . .>
                  <value>23.7</value>
            </field>
```

If the field is a reference to another object, store the *id* of that object as content of a *reference* element. For example:

```
            <field . . .>
                  <reference>5</reference>
            </field>
```

Of course, the object being referred to must also be serialized, and will be another *object* element nested inside the root element. Array objects will be similar to the *object* element described above, except that an additional *length* attribute is used, and each element of the array will be stored as content to a *value* or *reference* element, depending on the component type. For example:

```
      <object class="[C" id="8" length="5">
            <value>S</value>
            <value>m</value>
            <value>i</value>
            <value>t</value>
            <value>h</value>
      </object>
```

The Network Connection

Java provides the java.net.Socket class to help implement a network connection between two programs. Initially, while testing your system, you can run the two programs on the same computer. However, when demonstrating your system to the TAs, you must show that it works over a network connecting two separate computers.

The Deserializer

Deserialization will be implemented in a Java class called Deserializer, and will be invoked using the method:

public Object deserialize(org.jdom.Document document)

This method will deserialize an XML document passed in as parameter, returning the reconstituted object (and any objects it refers to). Use the facilities provided by JDOM to help with this, in particular the Document and SAXBuilder classes.

The Visualizer

This part of your system displays the deserialized objects to screen. You can use a text-based or graphical user interface for this. The Object Inspector you created in Assignment 2 may be helpful here. Be sure that this part of the system shows that the deserialized objects are properly recreated.

Other Requirements

You must demonstrate a working program to your TA during lab time shortly after the due date. Make certain you show that all requirements of the system have been fulfilled and implemented correctly. You must also use version control, unit testing, and refactoring throughout this assignment.

Submit the following:

1. An electronic copy of your source code, unit tests, your version control logs, and a record of your refactorings (in a Word, PDF or text file call *refactorings*). Use the *Assignment 3* Dropbox Folder in D2L to submit electronically.

Advanced Programming Techniques
Assignment 3 Grading

Student:_____

Functionality

Object Creation			
Simple object	2	_____	
Object with references to other objects	2	_____	
Object with array of primitives	2	_____	
Object with array of references	2	_____	
Object using collection instance	2	_____	
Serialization to JDOM document			
Simple object	2	_____	
Object with references to other objects	2	_____	
Object with array of primitives	2	_____	
Object with array of references	2	_____	
Object using collection instance	2	_____	
Sending/Receiving of byte steam	4	_____	
Deserialization from JDOM document			
Simple object	2	_____	
Object with references to other objects	2	_____	
Object with array of primitives	2	_____	
Object with array of references	2	_____	
Object using collection instance	2	_____	
Object Visualization	10	_____	

Other Requirements

Version control (show log files)	4	_____	
Unit Tests	4	_____	
Refactoring (described in *refactorings* file)	4	_____	
Design Quality	4	_____	
Total	**60**	_____	_____%