**Assignment 5**

**Project Description**

**Checkpointing Objects**

The purpose of this assignment is to create a generic library to serialize and deserialize objects. The code should allow the conversion of objects into a wire format. The code should be designed using dynamic proxies and reflection so that addition of new objects or serialization formats causes minimal changes (reduces the ripple effect).

* Creating the Dynamic Proxy
  + The Driver code should call the *createProxy* method in the ProxyCreator utility class to create a dynamic proxy reference. The code inside the createProxy method is shown below.
  + StoreRestoreI serDeserObj =
  + (StoreRestoreI)
  + Proxy.newProxyInstance(
  + getClass().getClassLoader(),
  + interfaceArray,
  + handler
  + );
  + Pass an array of interfaces to the createProxy method with the following interfaces (StoreI, RestoreI).
  + public interface StoreI extends StoreRestoreI {
  + void writeObj(MyAllTypesFirst aRecord, String wireFormat);
  + void writeObj(MySpecialTypes sRecord, String wireFormat);
  + }
  + public interface RestoreI extends StoreRestoreI {
  + SerializableObject readObj(String input);
  + }
  + SerializablObject is an empty base class
  + MyAllTypesFirst extends SerializableObject
  + MyAllTypesSecond extends MyAllTypesFirst
  + MySpecialTypes extends SerializableObject
  + StoreRestoreI.java is a tag interface
  + wireFormat is pseudoXML as shown in the sample files.
  + authID is an integer in the range of 1-9999
  + Pass an invocation handler to the createProxy() utility method.

 Design three Java classes MyAllTypesFirst, MyAllTypesSecond, MySpecialTypes, with data members that have names and types as shown in the serialized format shown in the file [MyAllTypes2.txt](http://www.cs.binghamton.edu/~mgovinda/courses/csx42/assignments/assign5/MyAllTypes2.txt). These classes should have the appropriate setX and getX methods, and empty/default constructors.

 Here is an example of multiple instances of the the two types in serialized pseudoXML format:

* [MyAllTypes2.txt](http://www.cs.binghamton.edu/~mgovinda/courses/csx42/assignments/assign5/MyAllTypes2.txt).

 Do NOT use an XML parser. The above format is NOT compliant with XML standards and so XML parsers will not work. Use the Java string library API to parse the pseudoXML format. For example, you can use substring library to find specific values within XML tags.

 Note that the data members for an instance may appear in a diffrent order in the serialized format.

 Note that you need to read the value of "xsi:type" to determine if a "genericCheckpointing.util.MyAllTypesFirst", "genericCheckpointing.util.MyAllTypesSecond", or "genericCheckpointing.util.MySpecialTypes", needs to be created, via reflection.

 If you add new method names to the interfaces note that the methods names in the proxy interfaces should be unique (don't use the same method name in two different interfaces, as it will cause problems with dynamic proxy usage).

 The driver code should invoke methods on the dynamic proxy, as if it is invoking methods on an object that implements the 2 interfaces (StoreI and RestoreI). Remember to cast the dynamic proxy to the correct interface before invoking the method.

 Each invocation will transfer control to the *invoke* method of the invocation handler.

 Invoke a method on the invocation handler to set a file name for the checkpoint file. Alternatively, you can add a parameter to the readObj and writeObj methods to include the file name.

 The invocation handler should have a method to open a file and a method to close the file.

 In the invocation handler do the following:

* If the method writeObj is called, serialize the object to the checkpointFile.
* You may need separate methods, perhaps in another class named SerializeTypes that has a method for each of the types that need to be serialized. So, the method String serializeInt(int value, String tagName), will be called if the field type of the argument is int, and fieldName and fieldValue of MyAllTypesFirst or MyAllTypesSecond will be passed. This method will then return a string such as "< myInt xsi:type="xsd:int">314</myInt>", wherein "myInt" and "314" are the arguments to the method and the rest are hardcoded.
* If the method readObj is called, deserialize into an object, and return it. Similar to SerializeTypes, you can use a class named DeserializeTypes with methods to deserialize each of the types.
  + Use reflection to create the object depending on the value in the complexType element.
  + Parse the names of the data members and then invoke the corresponding setX method to set the value for that data member.
* From the command line take these arguments: (1) mode and (2) checkpoint file name, (3) checkpoint-verify file name, and (4) debug value. The mode should be "deserser", or else thrown an exception, print a meaningful error message, and terminate the program.
* Once you read the objects into a data structure, serialize the objects in the data structure again using the writeObj(...) method back to the checkpoint-verify file name. We will compare the checkpoint file with checkpoint-verify file (To test your code, you can use the diff utility).
* Correctly override *equals* and *hashCode* for the three objects.
* Make sure your toString(...) method prints the data member names and values in a easy to read format, to STDOUT, when the appropriate debug level is used.
* If the TA changes the input file to be used to test deserialziation, or removes a field from one of the instances, your code should still run because that field will get the default value when the empty constructor gets called to instantiate.

Apply the strategy pattern for Deserialization and again for Serialization. The Strategy in each of these two cases is to use the XMLSerialization and XMLDeserialization for the given input. Note that there is only one strategy each to be used in this assignment for Serialization and Deserialization. However, employing the Strategy pattern supports adding multiple serialization and deserialization strategies.

* + - It is optional to have a separate method to serialize each type used in the MyAllTypes2.txt file. If you do so, place those methods in SerializeTypes.java and DeserializeTypes.java.
    - You may have a separate method to deserialize each complex type used in the MyAllTypes2.txt file.
  + Implement your own debug scheme and specify it in the README.md file
* Flow of Control
  + create a Dynamic Proxy
  + call the readObj(...) method on the proxy.
  + The deserialized instances should be stored in a vector, arrayList, or any other appropriate data structure.
  + call writeObj for each instance in the data structure so that the checkpoint-verify file is created via the proxy implementation.
  + call readObj to read the checkpoint file objects and create a vector with the return instances.

**Code Organization**

* Your directory structure should be the following:
* -firstName\_lastName\_assign5
* ---genericCheckpointing
* ----README.txt
* ----src
* ----build.xml
* ---genericCheckpointing
* ------driver
* ---------Driver.java
* ------server
* ---------StoreRestoreI.java [tag interface]
* ---------StoreI.java
* ---------RestoreI.java
* ------util
* ---------MyAllTypesFirst.java
* ---------MyAllTypesSecond.java
* ---------MySpecialTypes.java
* ---------ProxyCreator.java
* ---------SerializableObject.java [empty base class]
* ------xmlStoreRestore
* ---------StoreRestoreHandler.java (implements InvocationHandler)
* ------Any other Class/file you need