Assignment 5 (Final Project)

Due date

- 11.59 PM EST on December 5th.

Submit your code as per the provided instructions.

Updates

- Git: https://classroom.github.com/a/rPR9C13F

Assignment Goal

Apply the design principles you have learned so far to develop and test code for the given problem. Apply the dynamic proxy and any other applicable pattern(s).

Team Work

- You are required to work alone on this project.

You CANNOT collaborate or discuss the design, implementation, or debugging ideas with any other student. However, discussion on design is encouraged via the listserv.

Programming Language

You are required to program this project in Java.

Compilation Method

- You are required to use ANT for the following:
  - Compiling the code
  - running the code
  - Generating a tarball for submission
- Your code should compile and run on the *debian-pods* in the Computer Science lab in the Engineering Building.

Policy on sharing of code

- EVERY line of code that you submit in this assignment should be written by you or be part of the code template provided for this assignment. Do NOT show your code to any other group. Our code-comparison software can very easily detect similarities.
- Post to the listserv if you have any questions about the requirements. Do NOT post your code to the listserv asking for help with debugging. However, it is okay to post design/concept questions on programming in Java.

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, int authID, String wireFormat);
      void writeObj(MyAllTypesSecond bRecord, int authID, String wireFormat);
}

public interface RestoreI extends StoreRestoreI {
      SerializableObject readObj(String wireFormat);
}
```

  - SerializableObject is an empty base class
  - MyAllTypesFirst extends SerializableObject
  - MyAllTypesSecond extends SerializableObject
  - StoreRestoreI.java is a tag (mark) interface
  - wireFormat is pseudoXML as shown in the sample files.
  - authID is an integer in the range of 1-9999 (both 1 and 9999 are included)
  - Pass an invocation handler to the createProxy() utility method.

- Design two Java classes MyAllTypesFirst and MyAllTypesSecond with data members that have names and types as shown in the serialized format shown in the file MyAllTypes.txt. These two classes should have the appropriate setX and getX methods.
- Here is an example of multiple instances of the the two types in serialized pseudoXML format:
  - MyAllTypes.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" or "genericCheckpointing.util.MyAllTypesSecond" 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 seerialized. 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 hardcode.
  - If the method readObj is caled, 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 three arguments: (1) mode, (2) N and (3) checkpoint file name. The mode could be "serdeser" or "deser". The checkpoint file name we will use while testing for the "serdeser" mode is "checkpoint.txt". In the "serdeser" mode, N referes to the NUM_OF_OBJECTS of MyAllTypesFirst and MyAllTypesSecond each. In the "deser" mode, N refers to the number of objects to be deserialized..
  - Compare the objects that were serialized in writeObj to the objects that are returned from readObj. Correctly override *equals* and *hashCode*. For example, if you were using the vector data structure, you would compare if vector_old[i] is equal to vector_new[i]. Vector_old refers to the vector with the objects that were serialized. vector_new refers to the deserialized objects. Report how many objects did not match. If you run in "serdeser" mode then you should report "0 mismatched objects" to stdout for cases where all the filds are serialized. If the TA changes the input file to be used to test deserialziation,in between runs for the "deser" mode, then you should deserialize the file and print each object that was read to stdout. So, make sure your toString(...) method prints the data member names and values in a easy to read format, to stdout.

    Apply the strategy pattern for Serialization and again for Deserialization. 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.
    - NEW Design and implement a visitor, PrimeVisitor, to determine the total number of unique prime numbers in all the integer values in the instances of MyAllTypesFirst and MyAllTypesSecond that were serialized.
    - NEW Design and implement a visitor, PalindromVisitor, to determine the palindromes in the strings in the instances of MyAllTypesFirst and MyAllTypesSecond that were serialized.
      - 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.
  - Flow of Control
    - create a Dynamic Proxy
    - create a vector (or array list) of SerializableObject. Populate it with instances of MyAllTypesFirst and MyAllTypesSecond
    - when you create instances of MyAllTypesFirst and MyAllTypesSecond (using randomly generated values), if the value of an int, double, or long is less than 10, then that field should NOT be serialized. As shown in MyAllTypes2.txt, some fields could be missing from the serialized output.
    - call writeObj for each instance of the vector so that the checkPoint file is created via the proxy implementation.
    - call readObj to read the checkpoint file objects and create a vector with the return instances.

- compare the serialized and deserialized vectors and report how many instances match. Note that the match may not be perfect if the TA, during testing of your submission, changes the values of some of the data members in the checkPoint file.

Some General Design Considerations

- Same as before

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
                        ---------ProxyCreator.java
                        ---------SerializableObject.java [empty base class]
                ------xmlStoreRestore
                       ---------StoreRestoreHandler.java  (implements InvocationHandler)
                ------visitor
                       ---------PrimeVisitorImpl.java
                       ---------PalindromVisitorImpl.java
                       ---------VisitorI.java  (Visitor Interface)

                        ------Any other Class/file you need
```

Code Templates

- Driver.java
- ProxyCreator.java
- Notes on implementing the Strategy Pattern
- Example code snippet for serialization

Submission

- Same as Assignment-1.

Late Submissions

- The policy for late submissions is that you will lose 10% of the grade for each day that your submission is delayed.

Grading Guidelines

Here are the Grading Guidelines.

mgovinda at cs dot binghamton dot edu
Back to CSx42: Programming Design Patterns