Assignment 4

Due date

- 11.59 PM EST on Thursday, April 24th.

Submit your code as per the provided instructions.

Updates

- Tue Apr 15 23:22:50 EDT 2014: changed "char" to "short" in StudentRecord.java.
- Thu Apr 17 18:58:48 EDT 2014: empty constructor for StudentRecord and EmployeeRecord

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

- CS442: you can work in a team of two students on this project.
- CS542: you have 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
  - Generating javadocs
- Your code should compile and run on *bingsuns* or 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 your team 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/C/C++.

Project Description

Checkpointing Student/Employee Records

The purpose of this assignment is to create a generic library for persistent storage of student and employee records. 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 writeDJSON(SerializableObject aRecord, string wireFormat);
}

public interface RestoreI extends StoreRestoreI {
      SerializableObject readDJSON(string wireFormat);
}
```

  - SerializablObject is an empty base class
  - StudentRecord extends SerializableObject
  - EmployeeRecord extends SerializableObject
  - StoreRestoreI.java is a tag interface
  - Pass an invocation handler to the createProxy() utility method.

- Define the objects StudentRecord and EmployeeRecord as per the code template. Search for the string "FIXME" to determine what code needs to be added.
- 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.
- From the command line, read two arguments: NUM_OF_OBJECTS and checkpointFile.
- Invoke a method on the invocation handler to set a file name for the checkpoint file.
- 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 writeDJSON is called, serialize the object to the file checkpointFile. Choose your own serialization format.

- If the method readDJSON is caled, read from the checkpointFile, deserialize into an object, and return it.
- Compare the objects that were serialized in writeDJSON to the objects that are returned from readDJSON. Correctly overload *equals* and *hashCode*.

- Flow of Control

  - create StudentRecord and EmployeeRecord in a loop and write to a file using the dynamic proxy.
  - create a Dynamic Proxy
  - call writeDJSON with an instance of StudentRecord
  - call readDJSON and compare the returned object with the one that was serialized
  - compare the serialized and deserialized object
  - call writeDJSON with an instance of EmployeeRecord
  - call readDJSON and compare the returned object with the one that was serialized
  - compare the serialized and deserialized object

Some General Design Considerations

- Same as before

Code Organization

- Your directory structure should be the following:

```
-firstName_lastName
---genericCheckpointing
----build.xml
----README.txt
   ----src
       ---genericCheckpointing
                ------driver
                        ---------Driver.java
                ------server
                        ---------StoreRestoreI.java [tag interface]
                        ---------StoreI.java
                        ---------RestoreI.java
                ------util
                        ---------StudentRecord.java
                        ---------EmployeeRecord.java
                        ---------ProxyCreator.java
                        ---------SerializableObject [empty base class]
                ------djsonStoreRestore
                       ---------StoreRestoreHandler.java  (implements InvocationHandler)

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

Code Templates

- Driver.java
- ProxyCreator.java
- EmployeeRecord.java
- StudentRecord.java

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

Grading Guidelines.

mgovinda at cs dot binghamton dot edu
Back to CS 442: Programming Design Patterns