Assignment 5 -Final

Due date

- ~~11.59 PM EST, December 13th.~~ Noon on December 15th.

Submit your code as per the provided instructions. A signup sheet will be provided to you during class to setup an appointment with the TA to provide a demo of your project.

Updates

- Removed the requirement to report the performance of your code.
- 2 Sample input files have been posted.
- Due date has been changed
- Removed requirement for NUM_ITERATIONS, as there is no performance component in this assignment.

Assignment Goal

Comparing Objects in Java.

- 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 *remote.cs.binghamton.edu* or *bingsuns* or the *debian-pods* in the Computer Science lab in the Engineering Building.

Project Description

Java Reflection and Object Comparison in Java

- Project requirements:
  - Review the sample input file below to determine the data members and design a Java class, First, in the following way:
    - private data members
      - int IntValue;
      - String StringValue;
      - .. // similarly for other data members
    - empty constructor
    - define the public method *void setIntValue(int iIn) {... }*
    - define the public method *void setStringValue(String sIn) {... }*
    -
    - .. // similarly for other data members
  - Review the sample input file below to determine the data members and design a Java class, Second, in the following way:
    - data members
      - double DoubleValue;
      - int IntValue;
      - ... // similarly for the other data members
    - empty constructor
    - define the public method *void setIntValue(int iIn) {... }*
    - define the public method *void setDoubleValue(double dIn) {... }*
    - ... // similarly for the other data members
  - Define a class PopulateObjects that has data structures (as data members) to store instances of *First* and *Second*. Choose the data structure(s) that are

efficient to determine the total number of non-duplicate object instances, and the total number of object instances (includes duplicates).
- PopulateObjects should have a method deserObjects(...) to read data member values from an inputFile and accordingly create instances of First and Second. Decide the appropriate return value and parameters for the method *deserObjects*
- In First and Second, override equals and hashCode, with annotation, appropriately.
- You can design additional methods in PopulateObjects as needed.
- The class FileProcessor should be used to read one line at a time from the file.
- The input file will have data in the following format (note that the order of first and second is random. So, you have to read the fqn value to determine if what follows is a serialized format of first or second

```
fqn:genericDeser.util.First
type=int, var=IntValue, value=17
type=float, var=FloatValue, value=19.3
type=short, var=ShortValue, value=9
type=String, var=StringValue, value=abc
fqn:genericDeser.util.Second
type=int, var=IntValue, value=19
type=double, var=DoubleValue, value=3.14
type=boolean, var=BooleanValue, value=false
fqn:genericDeser.util.Second
type=int, var=IntValue, value=199
type=double, var=DoubleValue, value=33.14
type=boolean, var=BooleanValue, value=false
fqn:genericDeser.util.First
type=int, var=IntValue, value=177
type=float, var=FloatValue, value=199.3
type=short, var=ShortValue, value=99
type=String, var=StringValue, value=cbaabc
```

- Use java reflection (*newInstance* method) to create an object using the value given for fqn. Please note that fqn is an abbreviation for *Fully Quallified Class Name*.
- Here is an example of Java reflection code to create an object from given fqn value. This code then shows how to invoke a method.

```
String clsName = "genericDeser.util.First";         // generalize
Class cls = Class.forName(clsName);
Class[] signature = new Class[1];
signature[0] = Integer.TYPE;                        // generalize
String methdName = "set" + "IntValue";              // generalize
Method meth = cls.getMethod(methodName, signature);
Object obj = cls.newInstance();
Object[] params = new Object[1];
params[0] = new Integer(17);                        // generalize
Object result = meth.invoke(obj, params);
```

- Populate the data structures with instances of First and Second.
- Read the following link about boxed primitives, Integer.TYPE, and Integer.class in the context of Java reflection.
- Generalize the above code so it works for both First and Second objects. For example, you need to set signature[0] value by looking up a map that returns "Integer.TYPE" for the key "int".
- Populate the data structure in PopulateObjects class withe instances of First and Second that are read from the file.
- Design and implement methods in the PopulateObjects class to return the number of non-duplicate instances of First and Second.
- Design and implement methods in the PopulateObjects class to return the total number of instances of First and Second.
- The Driver code should call the PopulateObjects class to populate the data structures and print output on the number of objects.
- So, your final output will be 4 lines:

```
Number of unique First objects: 17
Total Number of First objects: 29
Number of unique Second objects: 19
Total Number of Second objects: 31
```

- The following should be read from command line (in this order): input file name, ~~the value of NUM_ITERATIONS~~, and DEBUG_VALUE. The input file should be read from the same folder as "build.xml".
- Your code should work when run in the following way. So, do NOT use hardcoded arguments in the build.xml file.

```
ant run -Darg0=input.txt ~~-Darg1=iterations~~ -Darg1=0
```

- Use the Logger class from the previous assignment along with your own debug scheme. The DEBUG_VALUE=0 should be reserved for just printing the 5 output lines shown above.

Sample Inputs

- First sample
  - Input and InputGenerator.java. [Thanks, Deepak]
- Another Input
  - Input-2 and Output. [Caution: Very large input file] [Thanks, Parshant].

Design Requirements

- Same as previous assignments, except that javadoc is now optional.

Code Organization

- Your directory structure should be the following:

```
lastName_firstName_assign5
  ---genericDeser
    ----- README.txt
    ----- build.xml
    ----- input.txt
    ----- src
      ---genericDeser
          ----------driver
                          ----------Driver
          ----------util
                     ----------First.java
                     ----------Second.java
                     ----------PopulateObjects.java
          ----------fileOperations
                     ----------FileProcessor.java
          ----------other packages that you need
                     [package and class(es) to use reflection to create objects
```

Code Templates

- None provided for this assignment.

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 Design Patterns