CPSC 501: Advanced Programming Techniques

Assignment 4: Reflective Object Serialization, Mocking

Collaboration

Discussing the assignment requirements with others is a reasonable thing to do, and an excellent way to learn. However, the work you hand-in must ultimately be your work. This is essential for you to benefit from the learning experience, and for the instructors and TAs to grade you fairly. Handing in work that is not your original work, but is represented as such, is plagiarism and academic misconduct. Penalties for academic misconduct are outlined in the university calendar.

Here are some tips to avoid plagiarism in your programming assignments.

1. Cite all sources of code that you hand-in that are not your original work. You can put the citation into comments in your program. For example, if you find and use code found on a web site, include a comment that says, for example:

   ```
   # the following code is from
   https://www.quackit.com/python/tutorial/python_hello_world.cfm.
   ```

   Use the complete URL so that the marker can check the source.

2. Citing sources avoids accusations of plagiarism and penalties for academic misconduct. **However, you may still get a low grade if you submit code that is not primarily developed by yourself. Cited material should never be used to complete core assignment specifications. You can and should verify and code you are concerned with your instructor/TA before submission.**
3. Discuss and share ideas with other programmers as much as you like, but make sure that when you write your code that it is your own. A good rule of thumb is to wait 20 minutes after talking with somebody before writing your code. If you exchange code with another student, write code while discussing it with a fellow student, or copy code from another person's screen, then this code is not yours.
4. **Collaborative coding is strictly prohibited**. **Your assignment submission must be strictly your code**. Discussing anything beyond assignment requirements and ideas is a strictly forbidden form of collaboration. This includes sharing code, discussing code itself, or modelling code after another student's algorithm. **You can not use (even with citation) another student's code.**
5. Making your code available, even passively, for others to copy, or potentially copy, is also plagiarism.
6. We will be looking for plagiarism in all code submissions, possibly using automated software designed for the task. For example, see Measures of Software Similarity (MOSS - https://theory.stanford.edu/~aiken/moss/).
7. Remember, if you are having trouble with an assignment, it is always better to go to your TA and/or instructor to get help than it is to plagiarize. **The most common penalty is an F on a plagiarized assignment.**
8. For assignments **limited use of generative AI in writing assistance is acceptable. For example, grammar suggestion, or code suggestion tools for programming**. **Programming or text that is largely generative AI produced is not allowed**. Learners are ultimately accountable for the work they submit. **Use of AI tools must be documented in an appendix for the assignment. The documentation should include what tool(s) were used, how they were used, and how the results from the AI were incorporated into the submitted work. Failure to cite the use of AI generated content in an assignment will be considered a breach of academic integrity and subject to Academic Misconduct procedures.**

Late Penalty

Assignment Late Policy: Students may choose to submit one or more assignments late, for no more than a total of five (5) days over the entire semester for the combined 4 assignments.

Each 24-hour period late after an assignment deadline counts as one full day regardless of how many hours the assignment was late within that period. For example, deadlines are generally on a Friday, 11:59pm local time. That means an assignment submitted any time Saturday before 11:59pm local time will be considered as 1 day late and count against the total number of late days.

Use these days at your own discretion and without explanation during the course for assignment extensions. For example, you could submit your second assignment 3 days late and your final assignment 2 days late, or just your final assignment 5 days late.

If a student still has days left to use, then their assignment will be graded without penalty. If a student has no more days left, or their submission exceeds the days they have remaining, then they will receive a 0 grade for a late assignment.

Late day usage will be tracked in a D2L grading column.

Goal

Using reflection to serialize an object to JSON and then de-serialize from JSON back to an object

Use Mocking reflection to enable client-side testing without server dependencies (Admittedly I will force a design model that requires this. Some refactoring, that we won't be doing, would minimize this testing requirement.)

Technology

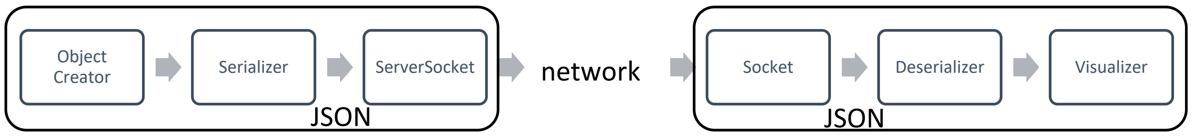
Java, JSON, Git, Gitlab, Reflection, Mocking

Specifics

Java 23, JUnit 5, Mockito, csgit.ucalgary.ca

Description

The goal of this assignment is to create two programs (that could be run on two separate computers) that communicate with each other over a network connection (see diagram below).



The **Sender** program will create an **object** (*that may have other internally referenced objects*). **Sender** will **serialize** the object created into a **JSON** String, and then send this **JSON** String as a stream of bytes to the **Receiver** program using a **socket** connection. The **Receiver** program will convert the incoming byte stream into a **JSON** String, **deserialize** the **JSON** into an **object**, and **display** the **object** to screen.

JSON is a popular dictionary style human-readable text-based storage structure. Games such as Spider-Man on PS4 and many web-based systems use **JSON** as a storage format for data. Spider-Man used it as a simple data platform to enable to creation of numerous game design tools that generated the required game world at the scale desired.

I have provided this code in the repository https://csgit.ucalgary.ca/jwhudson/cpsc501w25a4 that you can either copy code from, or fork and clone down for the assignment. It is a Maven project with a build file to pull the latest Junit, Mockito, and JSON.

The project contains a ObjectCreator, Serializer, Deserializer, Visualizer, and Inspector class that you should alter to complete the assignment.

The ObjectCreator and Visualizer need code completed to setup the network socket each time an object is sent, and code to change a JsonObject into a String. The Serializer/Deserializer change a JsonObject to your personal Object's created for this assignment.

There is an example of socket programming https://docs.oracle.com/javase/tutorial/networking/sockets/clientServer.html

Example of JSON changing of JsonObject to String. As well as starting code for serialization.

https://cspages.ucalgary.ca/~jwhudson/CPSC501W25/code/Reflection4GeneralPurpose/

The Object Creator

This part of your system will create objects under direction of the user. The object creator must allow the user to create one or more objects from a selection of objects using some sort of text-based menu system or GUI. **The user should be able to set different internal values each time they send an object.** This menu should loop letting the user create one object, then another, and then another until they are done. Each object is serialized immediately after creation and sent over the socket connection to the receiver before the next is created.

You must print out the JSON form of the object the user made to the shell (or your GUI) on the Sender side before you send it over the socket connection. This server side print out will allow you to demonstrate what is being sent.

You must be able to demonstrate that your system can handle the following kinds of objects (Please make a menu with each of these options):

1. **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. These fields should be public.
2. A simple object life in 1, except **the fields should be private.**
3. **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 reference-based graph with cycles).**
4. **An object that contains an 1D array of primitives.** Allow the user to set the values for the array elements to arbitrary values.
5. **An object that contains an 1D array of object references**. The other objects must also be created at the same time.
6. **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. (ex. a Java ArrayList)

The Serializer

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

public static JsonObject serializeObject(Object object)

This method will serialize the complete state of the **object** passed in as a parameter and produce a **JSON** object that could be turn into a string to be stored to file, printed out, or sent over a network connection. You can use a library to manage the creation of **JSON** objects, but must perform the reflection process yourself to put data in these objects. There are auto Java object to JSON libraries that exist but you cannot use them. We expect you to use javax.json.*;

The base of the **JSON** container must be a list of **objects**. The base object being sent will be first, and then any objects it references will be stored after it within this list. For example:

```
{"objects":[

        {"class":"class_name",…},

        {"class":"class_name",…}

]}
```

An entry for an object will have 4 entries. The name of the **class**, the unique integer **id** of the object (most likely created using **IdentityHashMap**), the **type** of the object (***object***/***array*** are the two types of types), and an entry which will contain each field of in **fields**. For example:

{"class":"Solution.ObjectA","id":"0","type":"object","fields":[…]}

Fields will be a list of the fields for the object. These fields may have been declared in the object itself, or by its parent, or parent's parent, etc. For example:

"fields":[

{"name":"x",…},

{"name":"y",…}

]

Each **field** will contain the field's **name**, the field's **declaring class**, and the ***value***/***reference*** of the field.

If the type of the field is a primitive, then store the **value** of the field.

{name":"field_name","declaringclass":"class_name","value":"1.0"}

If the field is a reference to a null.

{name":"field_name","declaringclass":"class_name","reference":"null"}

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

{name":"field_name","declaringclass":"class_name","reference":"1"}

Any additional object referenced by the root object, such as by the above **reference id** will also need to be serialized and stored in the root list of **objects** with the proper reference **id**. Each unique object should only be stored once, even if it is reference multiple times.

Array objects will be similar to the object element described first. We will use the type **array** to indicate that this is an array. We will also add the additional **length** attribute is used, and each element of the array will be stored in the list. As with fields a value entry will be stored as **value** and an object as a **reference.** For example:

{"class":"[I","id":"1","type":"array","length":"5","entries":[

{"value":"0"},

{"value":"0"},

{"value":"0"},

```
        {"value":"3"},
        {"value":"0"}
]}
```

The Network Connection

Java provides the **java.net.Socket** and **java.net.ServerSocket** 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. When demonstrating your system to the TAs must have evidence that your system would work between two computers. It must be clear that the code is working through a socket connection with **JSON** serialization.

The Deserializer

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

public static Object deserializeObject(JsonObject jsonObject)

This method will deserialize a **JSON** object assed in as parameter, returning the reconstituted object (and any objects it refers to).

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 3** may be modified to be used here in a text-based system. (Unlike Assignment 3 you only have to print class identifying information and field values and only have to fully follow recursion for each unique object once.) Be sure that this part of the system shows that the deserialized objects are properly recreated.

In the Visualizer you must use

public static Object read(Socket socket)

This function uses the provided open socket to read one String, turn it into a JsonObject, and then deserialize it into a Object. It then returns this object. You must continue to use this function as the following JUnit testing requirement will be built around it.

JUnit Testing Requirement

Use the Java library Mockito to Unit Test your Visualizer read(Socket) function. Make a test for each of the 6 types of objects requested earlier.

You should be able to use

InputStream inputStream = new ByteArrayInputStream(network_text.getBytes());

To turn to create an inputStream you can use when you Mock the Socket used in the read command. You can use it in place of

socket.getInputStream()

which would normally require a real socket with a real network link to have been made.

Note, to test object 2 which requires private fields you should use reflection to access to field values that are private (do not simply make accessors). We are making this test as you are not allowed to have accessors for this object. This object is allowed to have constructors that would set these private fields of course, when the object is made in the ObjectCreator.

MANDATORY Requirement

You must include a compile and execution readme.md that is clear enough that the TA can operate your system and the interactive menu it includes to verify that the requirements of the assignment are met. These instructions should be clear enough that the TA can quickly navigate your program to verify all requirements of the system have been fulfilled and implemented correctly. You must also use version control, unit testing, and refactoring throughout this assignment as in prior assignments.

Submit the following using the Assignment 4 Dropbox in D2L:

1. The complete source of your source code and your unit testing code.
2. Directions/access to the **csgit.ucalgary.ca** project so that your TA can access your project and read your report. TAs will need to be added as at least a 'Developer' role to your code.

Bonus:

Create a variant of your program that works with **XML as well as JSON**. Your objects should be serializable to **either JSON or XML** by the **Sender** and the **Receiver** should be able to deserialize from **both JSON and XML** depending on what is received. How to use this bonus XML mode must be clearly described in the readme.md such that the TA can access and execute the XML option.

Grading

ObjectCreator	Simple (with and without private fields), references, array primitives, array references, java.util.Collection	5	________
Serializer	Simple (with and without private fields), references, array primitives, array references, java.util.Collection	10	________
Deserializer	Simple (with and without private fields), references, array primitives, array references, java.util.Collection	10	________
Inspector	Inspector shows all relevant field data to verify recreation	4	________
Network	Network connection works	4	________
Version Control	Used appropriately	4	________
Visualizer Testing	Simple (with and without private fields), references, array primitives, array references, java.util.Collection	5	________
Refactoring	Used appropriately	4	________
Design quality	Clear how to use UI to send objects	4	________
Total		**50**	________
Bonus	JSON and XML format	5	________

Letter	**A+**	**A**	**A-**	**B+**	**B**	**B-**	**C+**	**C**	**C-**	**D+**	**D**	**F**
Points	**Above 50**	**47.5**	**45**	**42.5**	**40**	**37.5**	**35**	**32.5**	**30**	**27.5**	**25**	**Below 25**