Applying the Strategy Pattern

Re-submit Assignment

Due Mar 3 by 11:59pm **Points** 100 **Submitting** a website url

Applying the Strategy Pattern

The goal of this homework is to apply the strategy pattern to a problem. Recall that the strategy pattern allows you to define a family of algorithms that are interchangeable, which creates flexibility in your code modules.

This example will simulate a small database application that tracks car parts for a repair shop. The program allows a user to enter part information into the program and then to export or import data using several file formats. There are several options (strategies) we could use:

- JSON format
- XML format
- Comma-separated
- BSON, YAML, SGML, RTF, etc...

Our application will allow the following formats:

- Java objects using object serialization
- JSON using the Gson library
- XML using Java's DOM parser classes

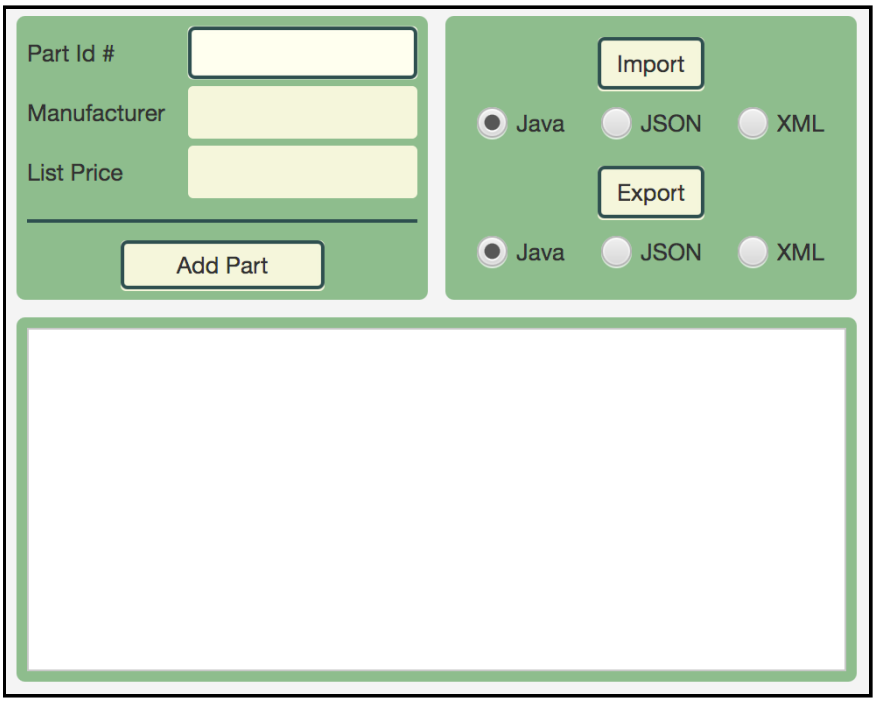
Project Setup

- Create a new Maven project in IntelliJ. Add dependency elements for the newest version of the following library.
  - Gson: **https://mvnrepository.com/artifact/com.google.code.gson/gson (https://mvnrepository.com/artifact/com.google.code.gson/gson)**
- I have already provided an IntelliJ project for you, which can be found **here**. This includes the following files:
  - PartsUI: contains code to generate a UI for a Java FX application. This class is complete and should not be altered.
  - CarPart: a simple Java class that represents a car part.
  - PartsModel: a small class that contains a collection of CarPart objects that are created by the user.

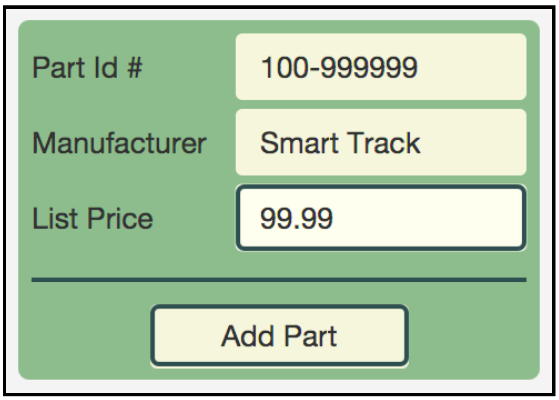
- PartsController: an incomplete controller that takes requests from the user and funnels them to our PartsModel class.

Database User Experience

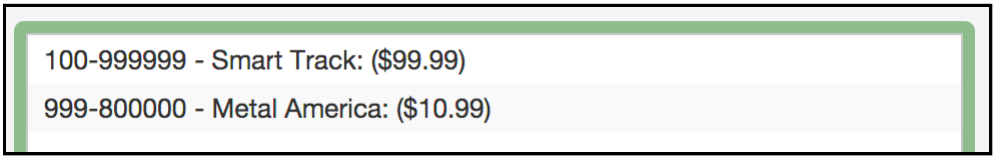
Below you can see an image of the GUI.



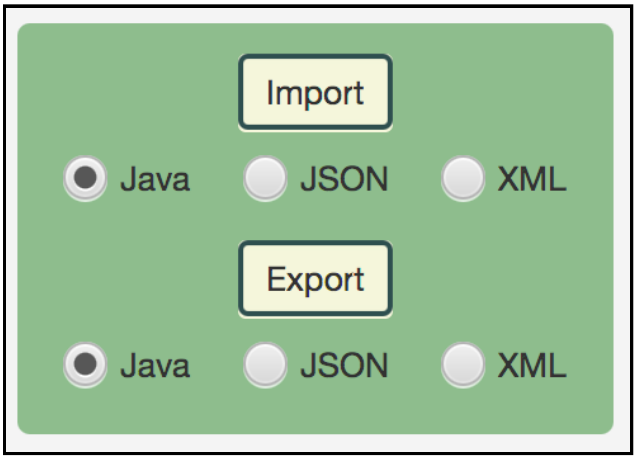
A user will first enter car parts on the left side of the GUI.



After all values are entered the user can press the "Add Part" button which will store the part internally in the application and display the part for the user. Multiple parts can be entered, as seen below.



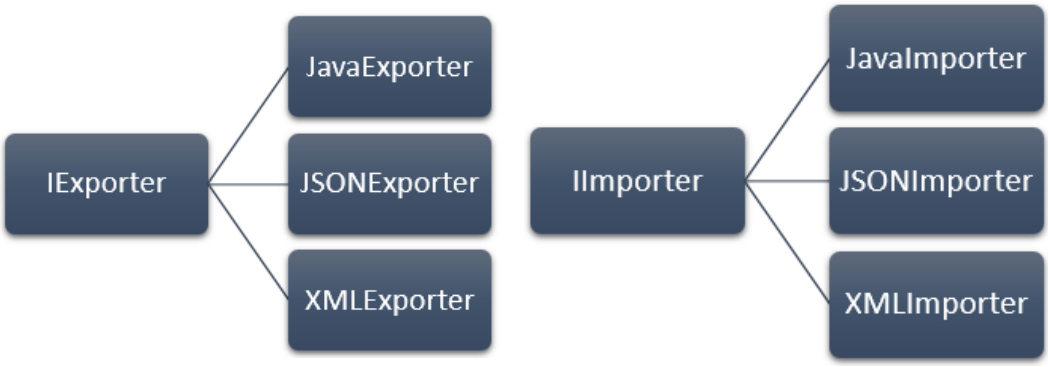
After entering records the user can choose to export them using the right side of the application. Once a user has exported their part records, they will be stored in one of the three chosen formats.



At a later point the user can then choose to import any car parts that are saved to file back into the application.

The Strategy Pattern

Our program allows several types of strategies for writing to files or reading from files. We will create an interface for each of our export types and import types. These interfaces will encapsulate a family of algorithms.



Each interface (and class) must have a single method:

IExporter

```
//Exports all CarPart objects in the application to a text file.
public void exportParts(PartsModel data);
```

IImportor

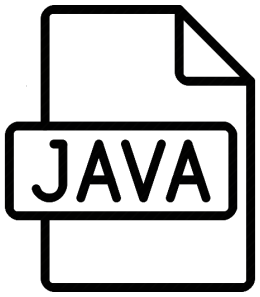
```
//Imports all CarPart objects from a text file to the application.
public void importParts(PartsModel data);
```

Expectations

You are expected to complete the application I have given. You should finish the following:

- Write the export and import classes seen above.
- Complete the unfinished methods in the PartsController class.

Serializing Java Objects (JavaExporter & JavaImporter)



The Java libraries contain two classes that allow you to read and write Java objects from files: ObjectOutputStream and ObjectInputStream.

- To use the classes you must declare your CarPart class as implementing the Serializable interface

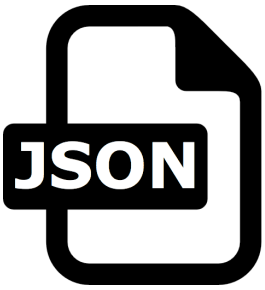
  Note: You may be required to declare a version id of the class you are writing. For example: public static final long serialVersionUID = 42L;

- Create a new ObjectOutputStream to write a CarPart object to a file
  - For more details see: **https://docs.oracle.com/javase/8/docs/api/java/io/ObjectOutputStream.html (https://docs.oracle.com/javase/8/docs/api/java/io/ObjectOutputStream.html)**
- Create a new ObjectInputStream to read a CarPart object from a file
  - For more details see: **https://docs.oracle.com/javase/8/docs/api/java/io/ObjectInputStream.html (https://docs.oracle.com/javase/8/docs/api/java/io/ObjectInputStream.html)**
- Data should be stored in a file called **parts.dat**
- The file contents of your Java objects will be mostly unreadable as can be seen below

```
 ¬¬ sr
model.CarPart¬#¬q¬7¬ D listPrice[
categoriest [Ljava/lang/String;L idt Ljava/lang/String;L manufacturerq ~ xp@X
¬\(¬ur [Ljava.lang.String;¬¬V¬¬{G xp t intaket manifoldt
100-999999t Smart Track
```

- You can write several objects to a file by repeatedly using the ObjectOutputStream.writeObject() method. Similarly, you can read several objects from a file using ObjectInputStream.readObject().

Working with JSON (JSONExporter & JSONImporter)



We will be using Google's Gson libraries to write objects to JSON format.

- Here is a code segment you can review: **https://www.mkyong.com/java/how-do-convert-java-object-to-from-json-format-gson-api/ (https://www.mkyong.com/java/how-do-convert-java-object-to-from-json-format-gson-api/)**
- Here is the API for the Gson framework: **https://google.github.io/gson/apidocs/com/google/gson/Gson.html**

(https://google.github.io/gson/apidocs/com/google/gson/Gson.html)

- Gson can be used to write an object directly to a file, converting the field-value pairs in the class to the key-value pairs in a JSON object
- The Gson.toJson() method accepts an object or array to convert to JSON text. You should pass this method an array of your CarPart objects.

Note: The default behavior of Gson object is to print all text to a single line in your output file. Use the following code segment to print newlines and make your output readable.

```
//prints to a single line
Gson gson = new Gson();

//prints using new line characters
Gson gson = new GsonBuilder().setPrettyPrinting().create();
```

- The Gson.fromJson() method reads JSON text from a file and converts it back into a Java object.
- You can use the following code segment to read an array of CarPart objects from a file into your program

```
CarPart[] parts = gson.fromJson(reader, CarPart[].class);
```

Data should be written to and read from a file called **parts.json**. Here is an example of your file contents:

```
[
  {
    "id": "100-999999",
    "manufacturer": "Smart Track",
    "listPrice": 99.99
  },
  {
    "id": "999-800000",
    "manufacturer": "Metal America",
    "listPrice": 10.99
  }
]
```

Binding Objects to an XML File (XMLExporter & XMLImporter)

We will read and write XML in our application using Java's DOM parser classes. This can be challenging and requires knowledge of the structure of documents on the web. Nevertheless, writing these code segments should be similar to what you have completed in your previous JavaScript class.

Here are two short tutorials on writing and reading to XML files using Java's DOM parser classes:

- Writing XML: https://www.mkyong.com/java/how-to-create-xml-file-in-java-dom/ (https://www.mkyong.com/java/how-to-create-xml-file-in-java-dom/)
- Reading XML: https://www.mkyong.com/java/how-to-read-xml-file-in-java-dom-parser/ (https://www.mkyong.com/java/how-to-read-xml-file-in-java-dom-parser/)

Note: You may not use any other library to read or write XML, including the Java architecture for XML binding (JAXB).
Note: If you are not familiar with XML, then peruse the W3Schools tutorial (http://www.w3schools.com/xml/) .

Data should be written to and read from a file called **parts.json**. Here is an example of your file contents:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<parts>
    <part>
        <id>100-999999</id>
        <manufacturer>Smart Track</manufacturer>
        <listprice>99.99</listprice>
    </part>
    <part>
        <id>999-800000</id>
        <manufacturer>Metal America</manufacturer>
        <listprice>10.99</listprice>
    </part>
</parts>
```

Requirements

Any submissions ignoring the following requirements will receive an immediate zero.

- You may not use any other frameworks for serializing Java objects, reading & writing JSON, and reading & writing XML.
- You may not alter the contents of the PartsUI file.

Submission

- All projects must be completed through the IntelliJ IDE
- Your project should be a Maven project with appropriate settings
- Your project files should be saved to a private GIT repository which is shared with myself
- All files must have proper documentation (comments and full Javadocs)
- All files must follow our style guidelines

Extra Credit (5 points)

Write a fourth pair of strategy classes using the CSV format. More can be read about the format here: https://www.computerhope.com/issues/ch001356.htm (https://www.computerhope.com/issues/ch001356.htm) . Make sure to add options on your user interface for this fourth option when importing or exporting records.

Applying the Strategy Pattern Rubric			
Criteria	**Ratings**		**Pts**
CarPart objects can be exported as serialized Java objects.	10.0 pts Full Marks	0.0 pts No Marks	10.0 pts
CarPart objects can be imported from serialized Java objects into the application.	10.0 pts Full Marks	0.0 pts No Marks	10.0 pts
CarPart objects can be exported in JSON format.	10.0 pts Full Marks	0.0 pts No Marks	10.0 pts
CarPart objects can be imported from a JSON file into the application.	10.0 pts Full Marks	0.0 pts No Marks	10.0 pts
CarPart objects can be exported in XML format.	10.0 pts Full Marks	0.0 pts No Marks	10.0 pts
CarPart objects can be imported from an XML file into the application.	10.0 pts Full Marks	0.0 pts No Marks	10.0 pts
All export and import classes use the appropriate interfaces, as described above. The code that selects a strategy for file formats should not be redundant.	15.0 pts Full Marks	0.0 pts No Marks	15.0 pts
IntelliJ project is set up correctly as a Maven project. All project files are saved to GIT in a private repository.	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
Full Javadocs are provided for all classes and public methods.	10.0 pts Full Marks	0.0 pts No Marks	10.0 pts
All code follows the style guide provided.	10.0 pts Full Marks	0.0 pts No Marks	10.0 pts
			Total Points: 100.0