EGR 327 Fall 2017
Dr. Im
Project
Due November 6, 2017 – 11:59 PM

In this project, you are to implement the following REST API and client code to call your API.

PART 1)
Implement the following REST API:

-
  ```
  @RequestMapping(value = "/addVehicle", method = RequestMethod.POST)
  public Vehicle addVehicle(@RequestBody Vehicle newVehicle) throws IOException {
  ```
  - addVehicle() will take a vehicle object and write it to a local text file.
  - It will always APPEND to the file.
  - So if I make 5 POST requests to /addVehicle, the local file will contain 5 vehicles in JSON.

-
  ```
  @RequestMapping(value = "/getVehicle/{id}", method = RequestMethod.GET)
  public Vehicle getVehicle(@PathVariable("id") int id) throws IOException {
  ```
  - getVehicle() will take a given id, and find the vehicle that has the matching id.
  - It will iterate the local file line-by-line, check if the id matches, and if there is a match return the vehicle object.

-
  ```
  @RequestMapping(value = "/updateVehicle", method = RequestMethod.PUT)
  public Vehicle updateVehicle(@RequestBody Vehicle newVehicle) throws IOException {
  ```
  - updateVehicle() will do the following given a vehicle object passed in:
    - Iterate the local file line-by-line
    - Check if the current line’s vehicle’s id matches the vehicle id that is passed in
    - If there is a match, update the current line with the vehicle that was passed in

-
  ```
  @RequestMapping(value = "/deleteVehicle/{id}", method = RequestMethod.DELETE)
  public ResponseEntity<String> deleteVehicle(@PathVariable("id") int id) throws IOException {
  ```
  - deleteVehicle() simply takes the given id and deletes from the local file.
  - It will iterate the local file line-by-line to check if the given id exists, then perform a delete.

-
  ```
  @RequestMapping(value = "/getLatestVehicles", method = RequestMethod.GET)
  public List<Vehicle> getLatestVehicles() throws IOException {
  ```
  - getLatestVehicles() will return the most recent 10 vehicles (as a list) that were added to the inventory.

PART 2) Implement client code to call your REST API.

Create a file called MyTasks.java and implement the following @Scheduled jobs.

-
  ```
  @Scheduled(some periodic rate)
  public void addVehicle() {
  ```
  - At some periodic rate, make POST request to add a vehicle.
  - Simply create a vehicle with random values and write it to file.
    - Vehicle Id should start from 1 and increment by 1 each time.
    - Vehicle makeAndModel should be a randomly generated string.
    - Vehicle year should be a random number between 1986-2016.
    - Vehicle retailPrice should be a random number between 15000-45000.

-
  ```
  @Scheduled(some periodic rate)
  public void deleteVehicle() {
  ```
  - At some periodic rate, make DELETE request to delete a vehicle.
  - Generate a random id, with some reasonable range (0-100)
  - Then make the DELETE request.

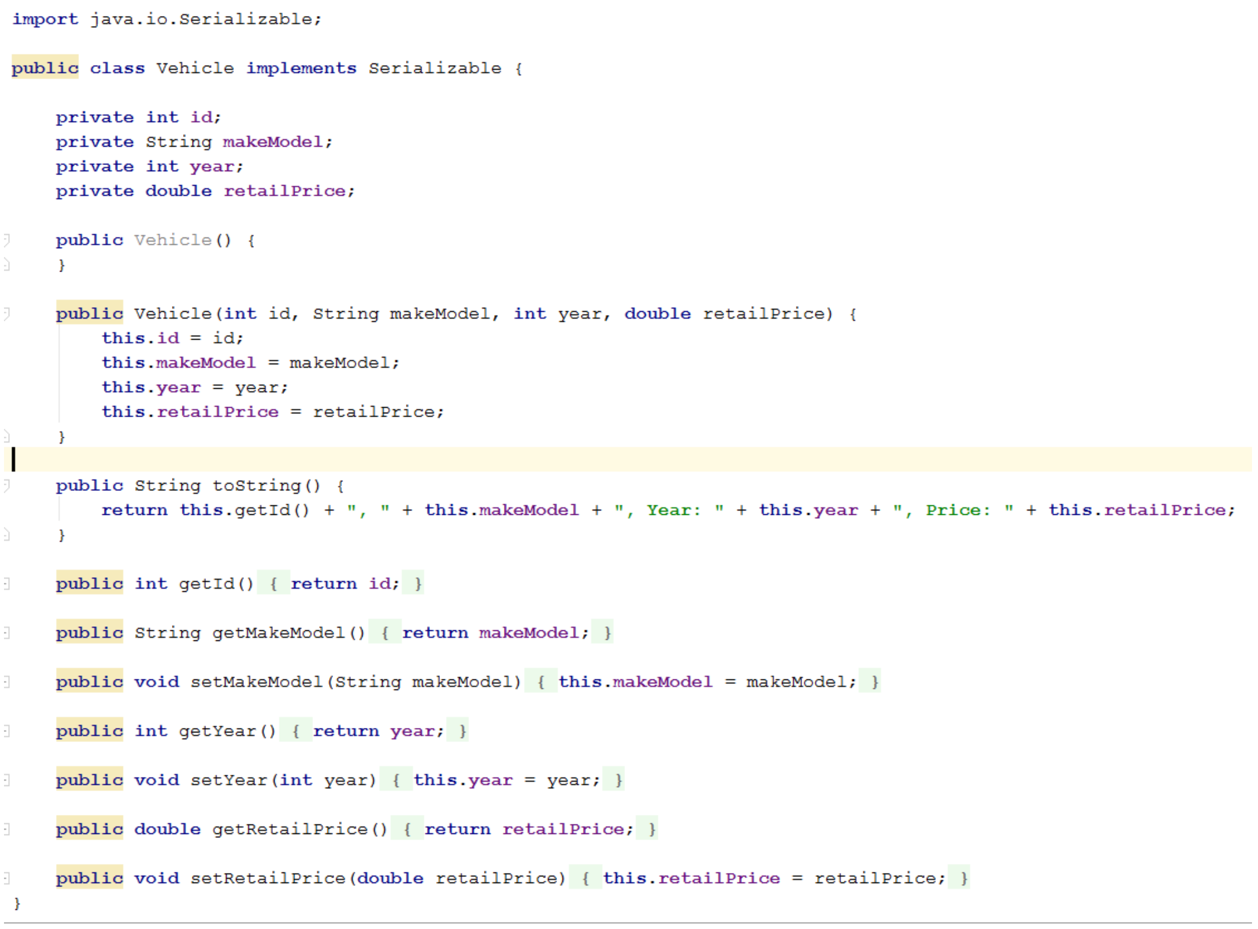
-
  ```
  @Scheduled(some periodic rate)
  public void updateVehicle() {
  ```
  - At some periodic rate, make PUT request to update a vehicle.
  - Create a new vehicle object with random values
    - Id should be some reasonable random number between 0 – 100
      - The id should be likely to exist in the file already
    - makeAndModel, year, and retail price can be hard-coded with some special values in this case to easily identify that you actually updated the vehicle.
  - After the update, make a GET request on the same id
    - To verify that you actually updated the vehicle

-
  ```
  @Scheduled(cron expression for at the top of each hour)
  public void latestVehiclesReport() {
  ```
  - At the top of each hour (ie. 9:00, 10:00, 11:00, etc… for every hour in the day), make a GET request to /getLatestVehicles. The latest 10 vehicles added to the inventory should be printed on the console.

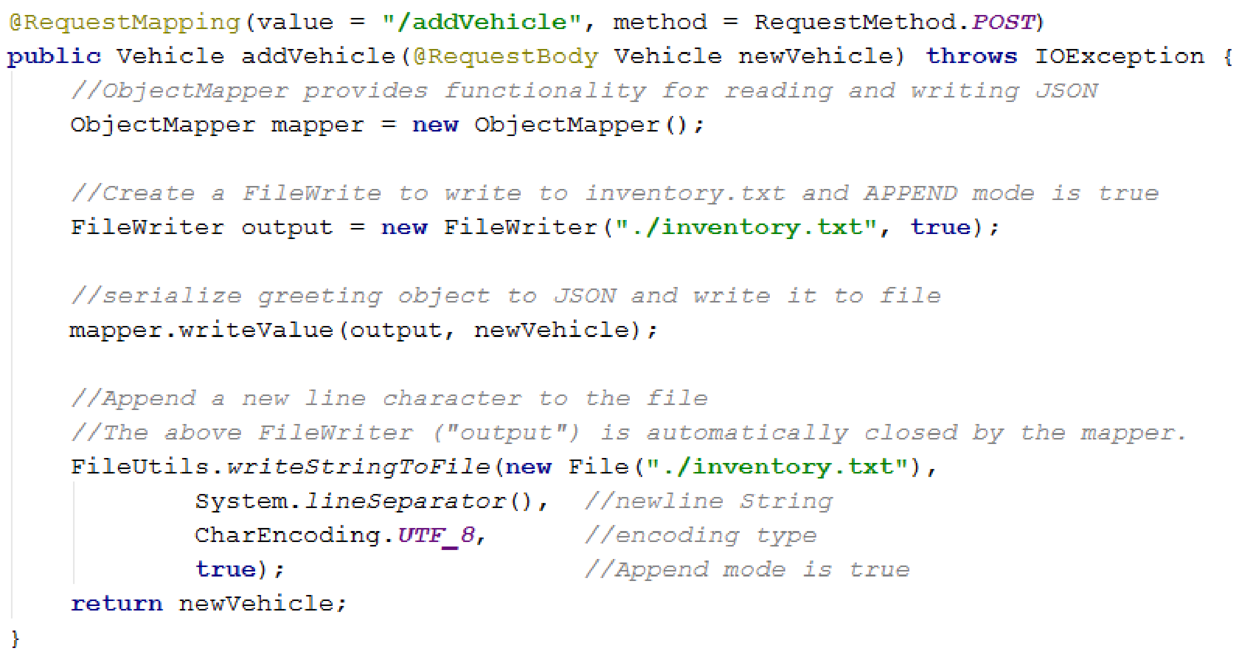
PART 3)

- Deploy your project to Google Cloud Platform (as will be discussed on Lecture 16).
  - Specifically, we will be using Google App Engine
- Remember to configure your app to use manual scaling with just 1 instance
- Once your app is deployed (and tested), copy paste your URL into your Word file as part of your submission.
  - See the last page for submission details.

Provided Code)

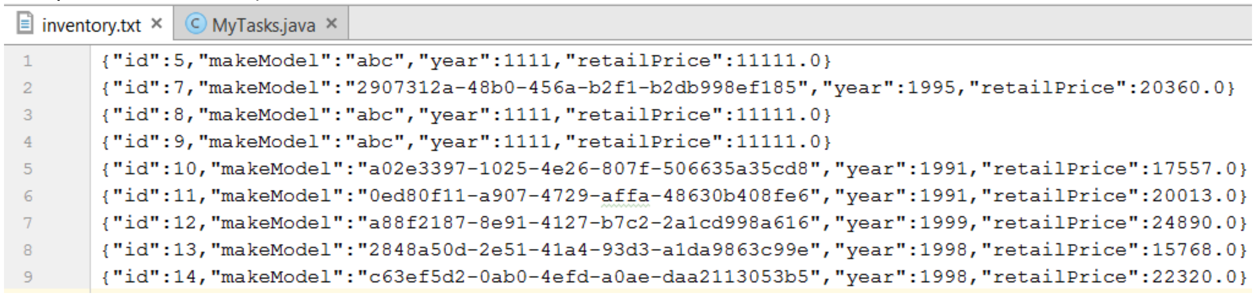


```java
import java.io.Serializable;

public class Vehicle implements Serializable {

    private int id;
    private String makeModel;
    private int year;
    private double retailPrice;

    public Vehicle() {
    }

    public Vehicle(int id, String makeModel, int year, double retailPrice) {
        this.id = id;
        this.makeModel = makeModel;
        this.year = year;
        this.retailPrice = retailPrice;
    }

    public String toString() {
        return this.getId() + ", " + this.makeModel + ", Year: " + this.year + ", Price: " + this.retailPrice;
    }

    public int getId() { return id; }

    public String getMakeModel() { return makeModel; }

    public void setMakeModel(String makeModel) { this.makeModel = makeModel; }

    public int getYear() { return year; }

    public void setYear(int year) { this.year = year; }

    public double getRetailPrice() { return retailPrice; }

    public void setRetailPrice(double retailPrice) { this.retailPrice = retailPrice; }
}
```



```java
@RequestMapping(value = "/addVehicle", method = RequestMethod.POST)
public Vehicle addVehicle(@RequestBody Vehicle newVehicle) throws IOException {
    //ObjectMapper provides functionality for reading and writing JSON
    ObjectMapper mapper = new ObjectMapper();

    //Create a FileWrite to write to inventory.txt and APPEND mode is true
    FileWriter output = new FileWriter("./inventory.txt", true);

    //serialize greeting object to JSON and write it to file
    mapper.writeValue(output, newVehicle);

    //Append a new line character to the file
    //The above FileWriter ("output") is automatically closed by the mapper.
    FileUtils.writeStringToFile(new File("./inventory.txt"),
            System.lineSeparator(),  //newline String
            CharEncoding.UTF_8,      //encoding type
            true);                   //Append mode is true
    return newVehicle;
}
```

Sample File Contents)



```
{"id":5,"makeModel":"abc","year":1111,"retailPrice":11111.0}
{"id":7,"makeModel":"2907312a-48b0-456a-b2f1-b2db998ef185","year":1995,"retailPrice":20360.0}
{"id":8,"makeModel":"abc","year":1111,"retailPrice":11111.0}
{"id":9,"makeModel":"abc","year":1111,"retailPrice":11111.0}
{"id":10,"makeModel":"a02e3397-1025-4e26-807f-506635a35cd8","year":1991,"retailPrice":17557.0}
{"id":11,"makeModel":"0ed80f11-a907-4729-affa-48630b408fe6","year":1991,"retailPrice":20013.0}
{"id":12,"makeModel":"a88f2187-8e91-4127-b7c2-2a1cd998a616","year":1999,"retailPrice":24890.0}
{"id":13,"makeModel":"2848a50d-2e51-41a4-93d3-a1da9863c99e","year":1998,"retailPrice":15768.0}
{"id":14,"makeModel":"c63ef5d2-0ab0-4efd-a0ae-daa2113053b5","year":1998,"retailPrice":22320.0}
```

Id 5, 8, and 9 shows updated vehicles with some hard-coded values to easily identify they were updated.
The rest of the line shows randomly generated vehicles added to the file.
Id 1, 2, 3, 4, and 6 are now shown because they were deleted through the DELETE request.

Submission Guidelines:

1. Push your code to github and name your repository as **<u>“Project”</u>** and share it with me.
    a. Username: **kyungsooim**
2. Take screenshots of your **code and output,** paste it into Word, and upload to Blackboard.
    a. Yes, screenshots of code this time or simply **copy/paste the code into Word (better option)**
3. URL of your deployed app in Google App Engine.