**The University of West Florida  
Department of Computer Science**

**Advanced Computer Programming Assignment**

**John W. Coffey**

**Project Topics and Goals**

* program refactoring to improve structure
* work with the Derby database and the Java Reflection API
* automate database creation from class definitions and instances

**Program Description**

This project will involve improving the structure of this basic database program: [testDB.zip](https://a8052-3329977.cluster86.canvas-user-content.com/courses/8052~14934/files/8052~3329977/course%20files/testDB.zip) and automating the creation of a database table by extracting the table layout from attributes of a class. To do this, we will utilize the Java reflection API. The example program is poorly structured since all functionality is in main(). Also, the program must be run from a command prompt, which is inconvenient.Your first task will be to refactor this example into a more modularized program with reusable methods to create a table in the database, to add records to the table, to issue basic queries on the database and to set up the program so that it does not have to run from the command line. You may use the derby.jar file provided. Your user's manual must be very clear on how to set up and run your program so that it will work. You must set up a batch file so that the program will run by executing the batch file.

You will create a class Vehicle. The class should have five instance fields: make (values: Chevy, Ford, Toyota, Nissan, Hyundai) size (values: compact, intermediate, fullSized), weight (double), engineSize (double) and import (boolean). You should implement this class utilizing the knowledge you gained in Intermediate Programming, such as setting instance fields with mutators (which, for example will ensure that the numeric values are not negative numbers).

You will use[Java serialization](https://www.javatpoint.com/serialization-in-java) to create a disk file with ten vehicles by randomly selecting makes and models from the above enumerations, and randomly select weights between 1500 and 4000 pounds. Weights must be such that compact cars (1500-2000) weigh less than intermediate cars (2000-2500) and intermediate cars weigh less than the rest. Full-sized cars are in the weight range (2500-4000). Create the 10 instances of class Vehicle and write them to a disk file named Vehicles.dat.

You will use the Reflection API to analyze all the instance fields in class Vehicle and to issue a command to create a table with the proper fields and proper data types for all fields. Your program will create a log file named "dbOperations.log" that notes all major database events in this process (all table creation, addition of values, queries, table deletions, etc). After creating the file and using reflection to set up the database table, you will read Vehicles.dat to get data to add to the database.

Note that the object oriented concept of composition, in which one class is composed of instances of another class, and classes that have containers, quickly complicate the process of performing this type of task. Object-Relational modeling seeks to address the mismatch between the way data is represented in objects and in relational databases. We will discuss this later.

Furthermore, the larger the set of data types we employ, the larger the overall problem we are addressing becomes. Consequently, we will limit data types our program must manage to ints, booleans, doubles and Strings. However, if your program encounters a field type that is none of the ones your program must handle, it must generate a log entry mentioning the field in the class that was not created in the database. Presumably, in a deployed system, a human could make decisions regarding how to handle these situations.  
  
  
**When the project runs:**

When your program runs, it will create the Vehicle instances and write them to Vehicles.dat. It will then analyze class Vehicle using the reflection API to extract the instance fields and it will create a SQL command to create the database table. It will issue that command. After doing that, it will read the 10 vehicle instances. It will then go through the vehicle instances, create SQL commands to add the values to the table, and issue the commands. Finally, the program will issue three SQL queries to display:

* all the vehicles that have been stored in the database.
* all Chevys and Toyotas
* all vehicles weighing more than 2500 pounds

Your program will create a log entry for each SQL operation performed. At the end, your program will read and display the log file. All SQL commands should ideally be issued as[prepared statements](https://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html).No interaction with the user should take place. When the program runs, it should do its work and display progress indicators along the way (for instance, " Creating Vehicles.dat", "Creating the database table", "Putting data in the table", and "Issuing query <querystring>". The output of the queries should also go to the console. Please do not have any file output after creating the Vehicle instances in the local file system.

**Deliverables**

You will submit the following for this project:  
1. A final UML class diagram that reflects your final program.

2. A User's manual for your program

3. Your source code in Java and any other utility files needed to run your program (eg: database.properties, Vehicle.java, etc). Do not submit .class files. Your batch file must automate compiling and running your program.

4. The batch file that compiles and runs your program.

Please review the policy on Academic dishonesty.

**Submission requirements:**

1. Compile and run your program one last time before submitting it. Your program must run with JDK 11.0.  
2. Place ALL files in your submission into a folder named <firstInitial><lastName>-p<number>.txt. If I were submitting project 1, my name would be jcoffey-p1.  
3. zip that folder into a .zip file with the SAME NAME. This means that inside your zip file, you will have exactly one folder (from the example: jcoffey-p1) showing at the top level.  
4. DO NOT make separate folders for documentation and source files (or anything else) inside the main folder. Having such a setup simply necessitates more navigation time to get where we need to go to grade.  
5. Any needed input files should be in the top-level folder along with the source code.  
6. MacOS users - remove the \_MacOS\_ utility folder before you zip up the file. If you cannot, delete it from the archive once the archive is created. It just takes up space and is not needed for anything we do with your submissions.  
7. Login into Canvas. Select our course.  
9. Go to the Assignments and choose Project 1.  
10. Click the "Submit Assignment" button and upload your .zip file. ALWAYS give yourself enough time. If you are trying to submit at 11:57pm on your machine, the clock might be off and the dropbox might already be closed.

Please review the policy on ACADEMIC MISCONDUCT. This is an *individual assignment.*