Military Vehicle Database

CSI 2300 - Object Oriented Programming

Brandon Praet

April, 22, 2025

Agenda

1. Guiding Principles
2. UI
3. UML
4. Implementation
5.
6. Demo / Questions

Guiding Principles

(Context: My background is in Automotive)

Main Course Objectives:

- Build a Java program using object-oriented principles
- Apply UML class diagrams
- Design a GUI with JavaFX

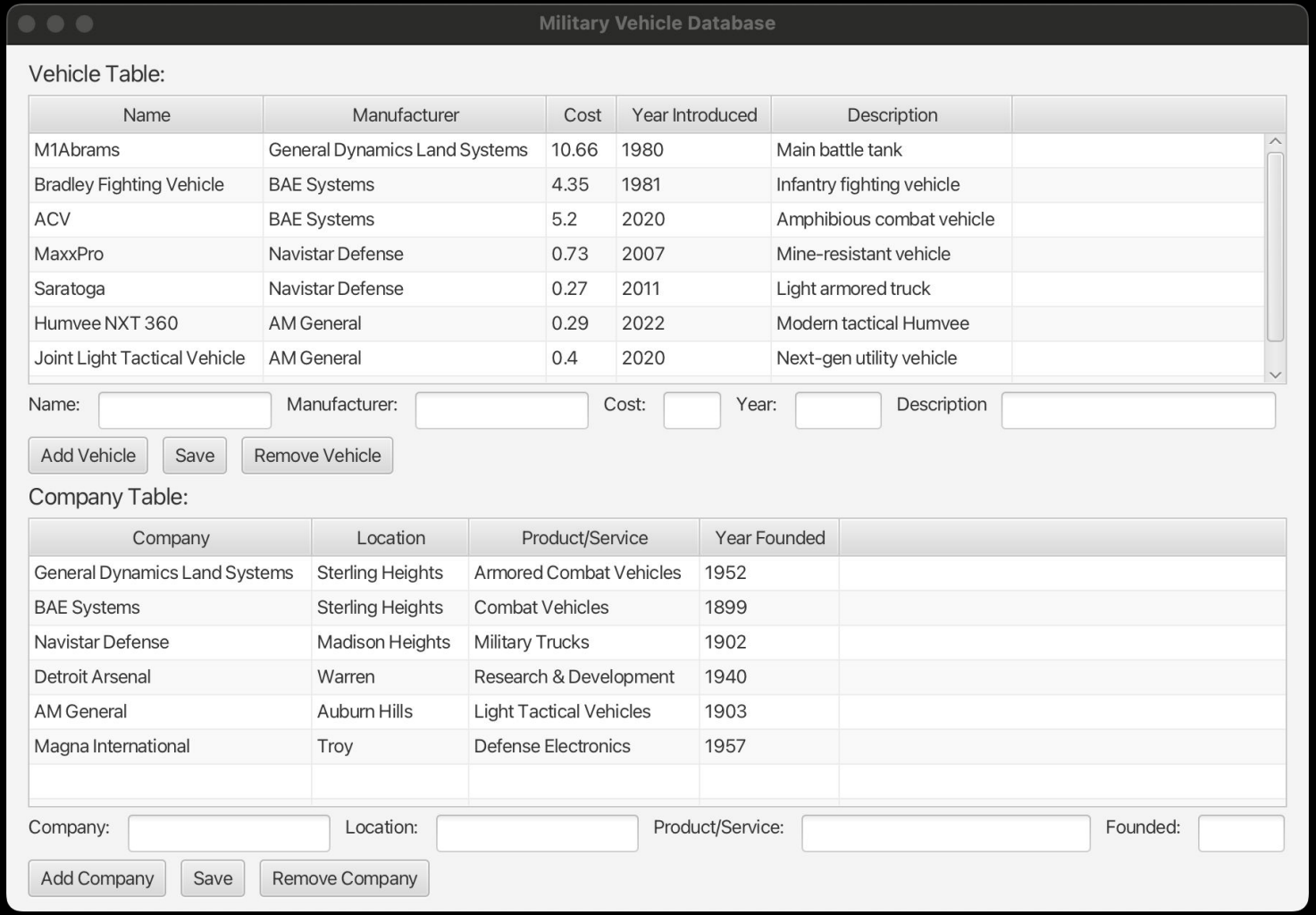
Personal Goals:

- Gain exposure to a new industry
- Research local career opportunities
- Build and showcase Java skills

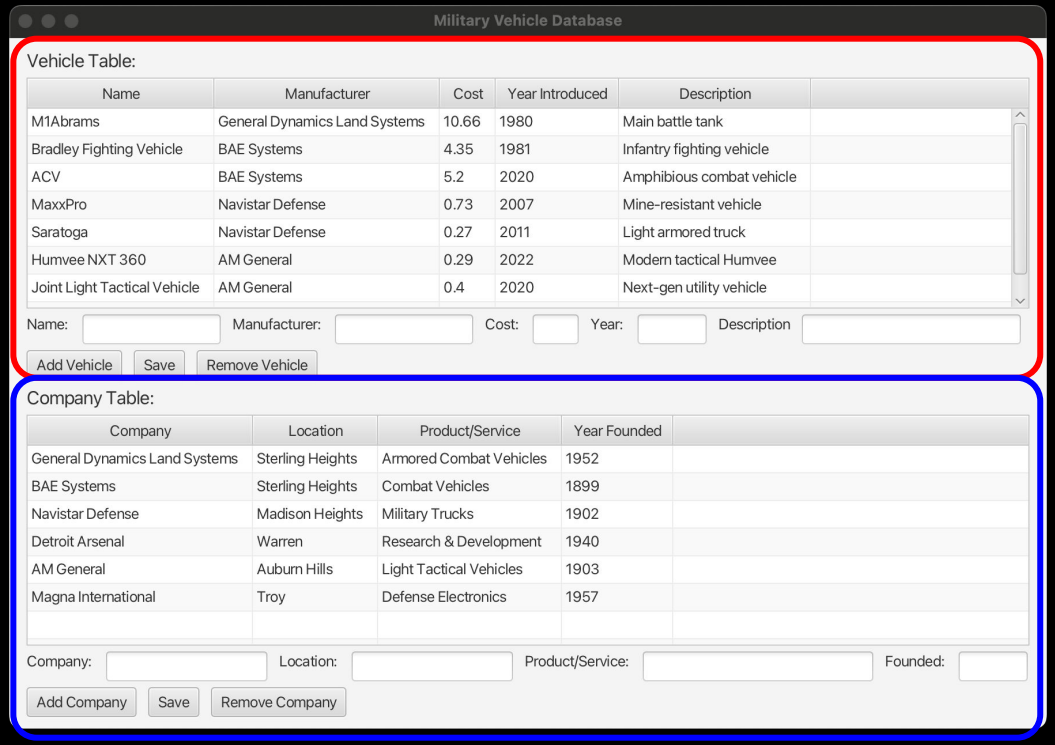
Problem Statement:

Create a Military Vehicle Database to manage data on local defense manufacturers and their vehicles.

UI


Military Vehicle Database
Vehicle Table:
Name
Manufacturer
Cost
Year Introduced
Description
M1Abrams
General Dynamics Land Systems
10.66
1980
Main battle tank
Bradley Fighting Vehicle
BAE Systems
4.35
1981
Infantry fighting vehicle
ACV
BAE Systems
5.2
2020
Amphibious combat vehicle
MaxxPro
Navistar Defense
0.73
2007
Mine-resistant vehicle
Saratoga
Navistar Defense
0.27
2011
Light armored truck
Humvee NXT 360
AM General
0.29
2022
Modern tactical Humvee
Joint Light Tactical Vehicle
AM General
0.4
2020
Next-gen utility vehicle
Name:
Manufacturer:
Cost:
Year:
Description
Add Vehicle
Save
Remove Vehicle
Company Table:
Company
Location
Product/Service
Year Founded
General Dynamics Land Systems
Sterling Heights
Armored Combat Vehicles
1952
BAE Systems
Sterling Heights
Combat Vehicles
1899
Navistar Defense
Madison Heights
Military Trucks
1902
Detroit Arsenal
Warren
Research & Development
1940
AM General
Auburn Hills
Light Tactical Vehicles
1903
Magna International
Troy
Defense Electronics
1957
Company:
Location:
Product/Service:
Founded:
Add Company
Save
Remove Company

Data Management UI



The UI is divided into two sections for structured data management:

Vehicle Table:

Field	Description:
Name	Common name of the vehicle
Manufacturer	Company that produces the vehicle
Cost	Estimated unit cost (in million USD)
Year Introduced	Year the vehicle entered service
Description	Vehicle type or operational role

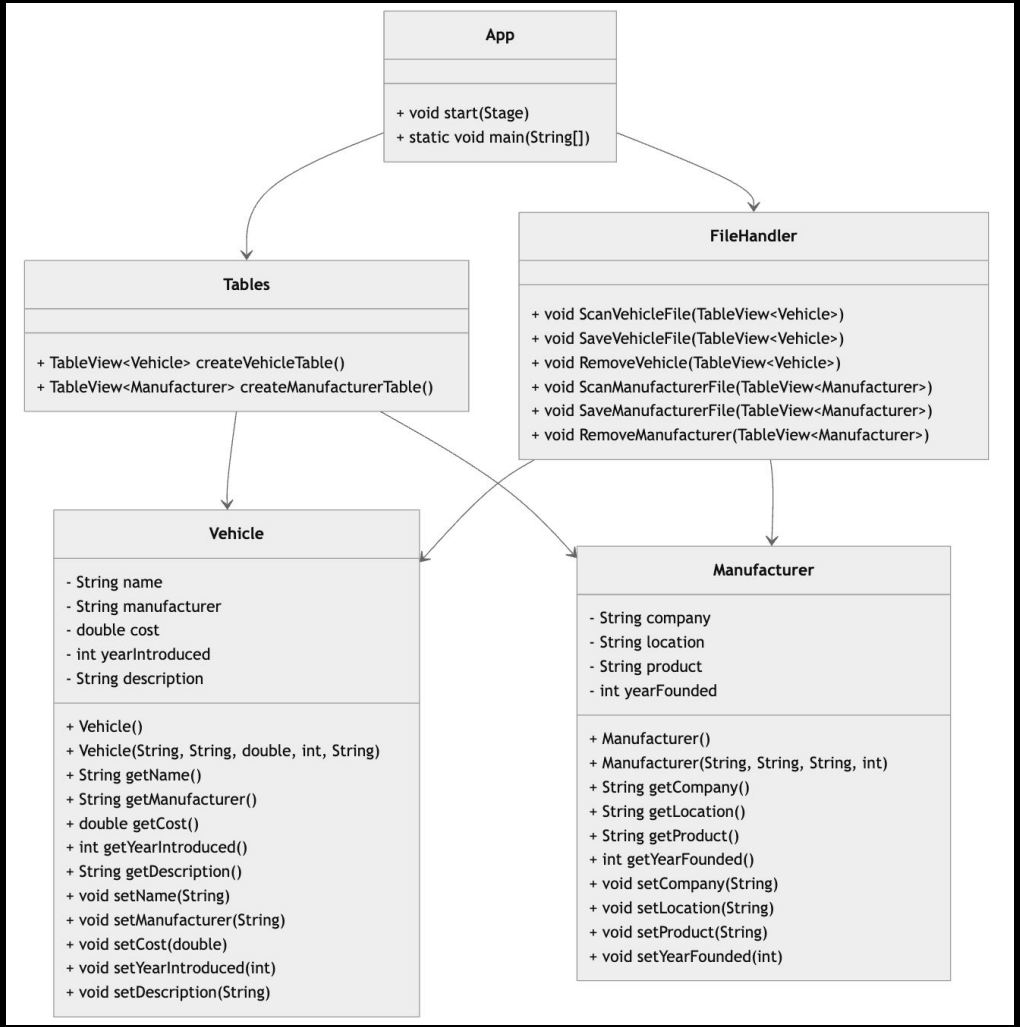
Company Table:

Field	Description:
Company	Full name of the manufacturer or contractor
Location	City of primary operations or major facility
Product/Service	Main product or service provided
Year Founded	Year the company was established

Button Functions

1. **Add**: Enter details in the text fields, then click “Add Vehicle” or “Add Company” to submit the entry.
2. **Save:** Click “Save” to write all current table entries to the text file.
3. **Remove**: Select a row, then click “Remove Vehicle” or “Remove Company” to delete the entry.

UML


App
+ void start(Stage)
+ static void main(String[])
Tables
+ TableView<Vehicle> createVehicleTable()
+ TableView<Manufacturer> createManufacturerTable()
FileHandler
+ void ScanVehicleFile(TableView<Vehicle>)
+ void SaveVehicleFile(TableView<Vehicle>)
+ void RemoveVehicle(TableView<Vehicle>)
+ void ScanManufacturerFile(TableView<Manufacturer>)
+ void SaveManufacturerFile(TableView<Manufacturer>)
+ void RemoveManufacturer(TableView<Manufacturer>)
Vehicle
- String name
- String manufacturer
- double cost
- int yearIntroduced
- String description
+ Vehicle()
+ Vehicle(String, String, double, int, String)
+ String getName()
+ String getManufacturer()
+ double getCost()
+ int getYearIntroduced()
+ String getDescription()
+ void setName(String)
+ void setManufacturer(String)
+ void setCost(double)
+ void setYearIntroduced(int)
+ void setDescription(String)
Manufacturer
- String company
- String location
- String product
- int yearFounded
+ Manufacturer()
+ Manufacturer(String, String, String, int)
+ String getCompany()
+ String getLocation()
+ String getProduct()
+ int getYearFounded()
+ void setCompany(String)
+ void setLocation(String)
+ void setProduct(String)
+ void setYearFounded(int)

Class Breakdown

Class Categories:

1. **GUI**
   - `App.java` : Manages layout, event handling, and application launch.
2. **Utility**
   - `Tables.java` : builds and returns table views.
   - `FileHandling.java` : Handles loading and saving data to text files.
3. **Data**
   - `Vehicle.java` : Represents a military vehicle entry.
   - `Manufacturer.java` : Represents a company or contractor.

Tables.java

```java
public static TableView<Vehicle> createVehicleTable(){

    // Create a TableView
    TableView<Vehicle> table = new TableView<>();

    // Create columns and cell factory for each
    TableColumn nameColumn = new TableColumn<Vehicle,String>("Name");
    nameColumn.setCellValueFactory(new PropertyValueFactory<Vehicle,String>("name"));

    TableColumn manufacturerColumn = new TableColumn<Vehicle,String>("Manufacturer");
    manufacturerColumn.setCellValueFactory(new PropertyValueFactory<Vehicle,String>("manufacturer"));

    TableColumn costColumn = new TableColumn<Vehicle,String>("Cost");
    costColumn.setCellValueFactory(new PropertyValueFactory<Vehicle,String>("cost"));

    TableColumn<Vehicle, Integer> yearColumn = new TableColumn<>("Year Introduced");
    yearColumn.setCellValueFactory(new PropertyValueFactory<>("yearIntroduced"));

    TableColumn<Vehicle, String> descriptionColumn = new TableColumn<>("Description");
    descriptionColumn.setCellValueFactory(new PropertyValueFactory<>("description"));

    // Add columns to table
    table.getColumns().addAll(nameColumn, manufacturerColumn, costColumn, yearColumn, descriptionColumn);

    return table;

}
```

Constructs `TableView` instances for vehicle and company data.

- Defines table columns for object fields like name, cost, and year.
- Connects columns to field values using ‘setCellValueFactory’.
- Returns the fully configured ‘TableView’ for use in the UI.

FileHandler.java

```java
// Scanner method to populate the table with info from the VehicleInfo.txt file
public static void ScanVehicleFile( TableView<Vehicle> table) {
    File infoFile = new File("VehicleInfo.txt");    // Create a file object for the tex
    try {
        Scanner scanner = new Scanner(infoFile);        // Scan from that file
        while (scanner.hasNextLine()) {                 // While there is info in the f

            String line = scanner.nextLine();           // Scan line by line
            String[] vehicleEntry = line.split(",");    // Create vehicleEntry array by
            String name = vehicleEntry[0];              // Takes the 0th index of array
            String manufacturer = vehicleEntry[1];              // Takes the 1st index a
            double cost = Double.parseDouble(vehicleEntry[2]);  // Takes the 2nd index a
            int year = Integer.parseInt(vehicleEntry[3]);       // Takes the 3rd index a
            String description = vehicleEntry[4];               // Takes the 4th index a
            //************************************* Can add more entries here later

            Vehicle vehicle = new Vehicle(name, manufacturer, cost, year, description);
            table.getItems().add(vehicle);                              // and adds it to t
        }
        scanner.close();                                // Close the scanner to prevent

    } catch (FileNotFoundException ex) {                // Catch portion of our try-cat
        ex.printStackTrace();
    }
}
```

(Not pictured: SaveVehicleFile() and RemoveVehicle())

Handles file input/output using simple `.txt` files with CSV formatting.

- Loads data into tables from `VehicleInfo.txt` and `ManufacturerInfo.txt`.
- Saves table data back to the same files.
- Removes entries based on selected row in the table.

Enables persistence between application runs!

Reflections

Achievements:

- First time designing and implementing a complete java program.
- Gained hands-on experience with GUI design and file I/O integration.

Lessons Learned:

- Thorough initial UML helps avoid design conflicts.
- Consistent, simple commenting makes revisiting code much easier.

Improvements:

- Add in-table editing for faster updates.
- Use inheritance for vehicle subtypes (e.g. `GroundVehicle -> LightTransport`)
- Upgrade to a more advanced database.

Demo + Questions