# PRO192: OOP in Java
# Assignment

## Learning outcome:

Upon successful completion of this assignment, you will have demonstrated the abilities to:

- Uses streams to read and write data from/to different types of sources/targets
- Identify classes, objects, members of a class, and relationships among them needed for a specific problem
- Explain the concept and demonstrates the use of Polymorphism, Encapsulation, Abstraction, and Inheritance in java

- Understand and implement a complete program using a collection framework
- Describe to your instructor what you have learned in completing this workshop.

## Scenario

The car showroom, named Minh Trang BMW, has a list of BMW cars. BMW brands are stored in a text file, named **brands.txt,** and cars are stored in a text file, named **cars.txt** as following:

| **File brands.txt** | **Description** |
|---|---|
| B7-2018, BMW 730Li (2018), Harman Kardon: 3.749<br>B7-MS, BMW 730Li M Sport, Harman Kardon: 4.319<br>B7-MS20, BMW 730Li M Sport (2020), Harman Kardon: 4.369<br>B7-PE, BMW 730Li Pure Excellence, Harman Kardon: 4.929<br>B5-18, BMW 530i (2018), Alpine: 2.599<br>B7019, BMW 530i (2019) , Alpine: 2.729<br>BX4-18, BMW X4 xDrive20i (2018), Sony: 2.799<br>BX4-17, BMW X4 xDrive20i (2019) , Sony: 2.899<br>B3-GT18, BMW 320i GT (2018), Bose: 1,799<br>B3-S19, BMW 320i Sportline (2019), Bose: 1.899<br>B5-X19, BMW X5 xDrive40i XLine (2019), Bose: 4.199<br>B5-X20, BMW X5 xDrive40i XLine (2020), Bose: 4.239 | Information in a line:<br><ID, brand name, sound brand: price> |

| **File cars.txt** | **Description** |
|---|---|
| C01, B7-2018, red, F12345, E12345<br>C02, B7-2018, black, F12346, E12346<br>C03, B7-MS, orange, F12347, E12347<br>C04, B7-MS20, white, F12348, E12348<br>C05, B7-PE, pink, F12349, E12349<br>C06, B5-18, pink, F12350, E12350<br>C07, B5-X20, grey, F12351, E12351 | Information of a line:<br><ID, brand ID, color, frame ID, engine ID> |

## Problem requirements

The manager of the showroom needs a Java console application in which operations must be supported:

1- List all brands
2- Add a new brand
3- Search a brand based on its ID
4- Update a brand
5- Save brands to the file, named brands.txt
6- List all cars in ascending order of brand names
7- List cars based on a part of an input brand name
8- Add a car
9- Remove a car based on its ID
10- Update a car based on its ID
11- Save cars to file, named cars.txt

**Constraints**

1- **Constraints on brands**:
   a. Brand ID can not be duplicated.
   b. The brand name can not be blank.
   c. The sound manufacturer can not be blank.
   d. The price must be a positive real number.
2- **Constraints on cars**:
   a. Car ID can not be duplicated.
   b. Brand ID must have existed and it must be inputted using a menu.
   c. Color can not be blank.
   d. Frame ID can not be blank and must be in the "F00000" format and can not be duplicated.
   e. Engine ID can not be blank and must be in the "E00000" format and can not be duplicated.

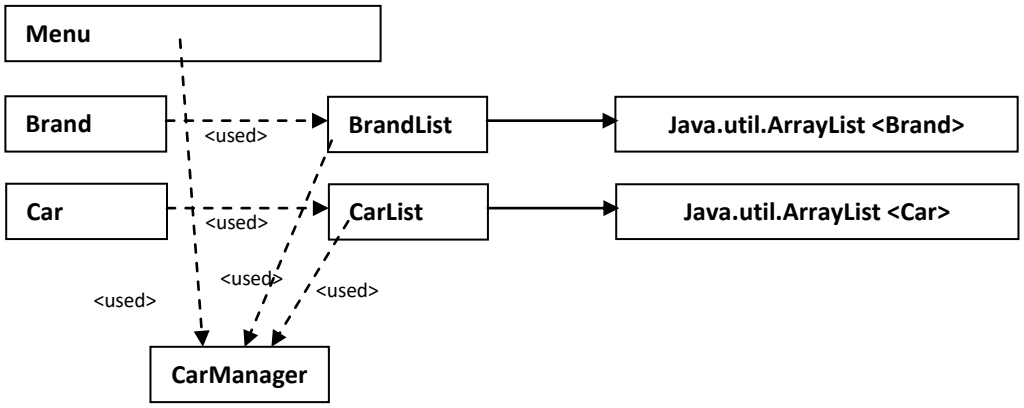
## Analysis

From the problem description, main concepts and their details are identified:

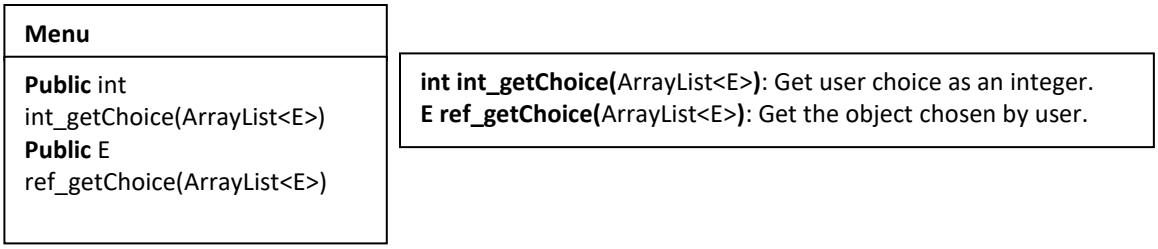
| Concept | Detail |
|---|---|
| Brand | Brand ID, brand name, sound brand, price |
| List of brands | |
| Car | Car ID, brand ID, color, frame ID, engine ID |
| List of cars | |
| Menu | A list of objects |
| Program | A menu, a list of brands, a list of cars |

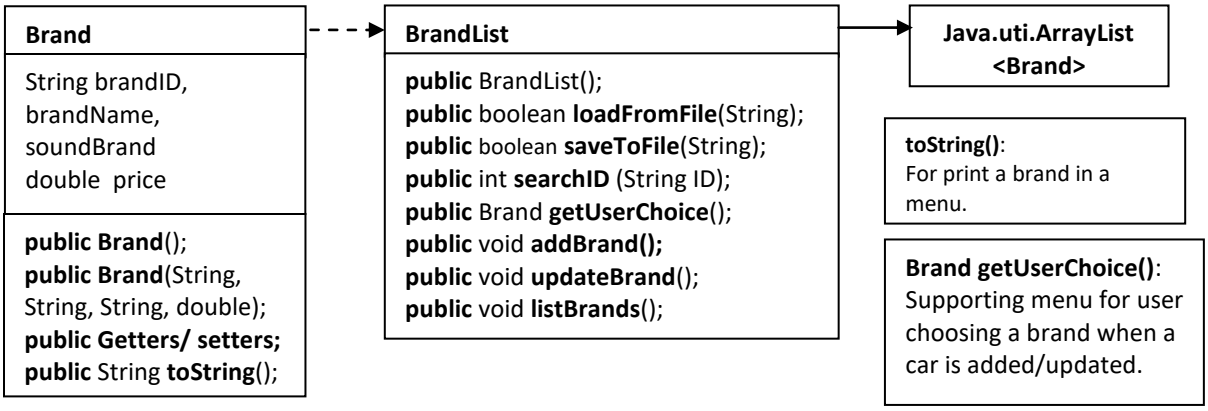
# Design

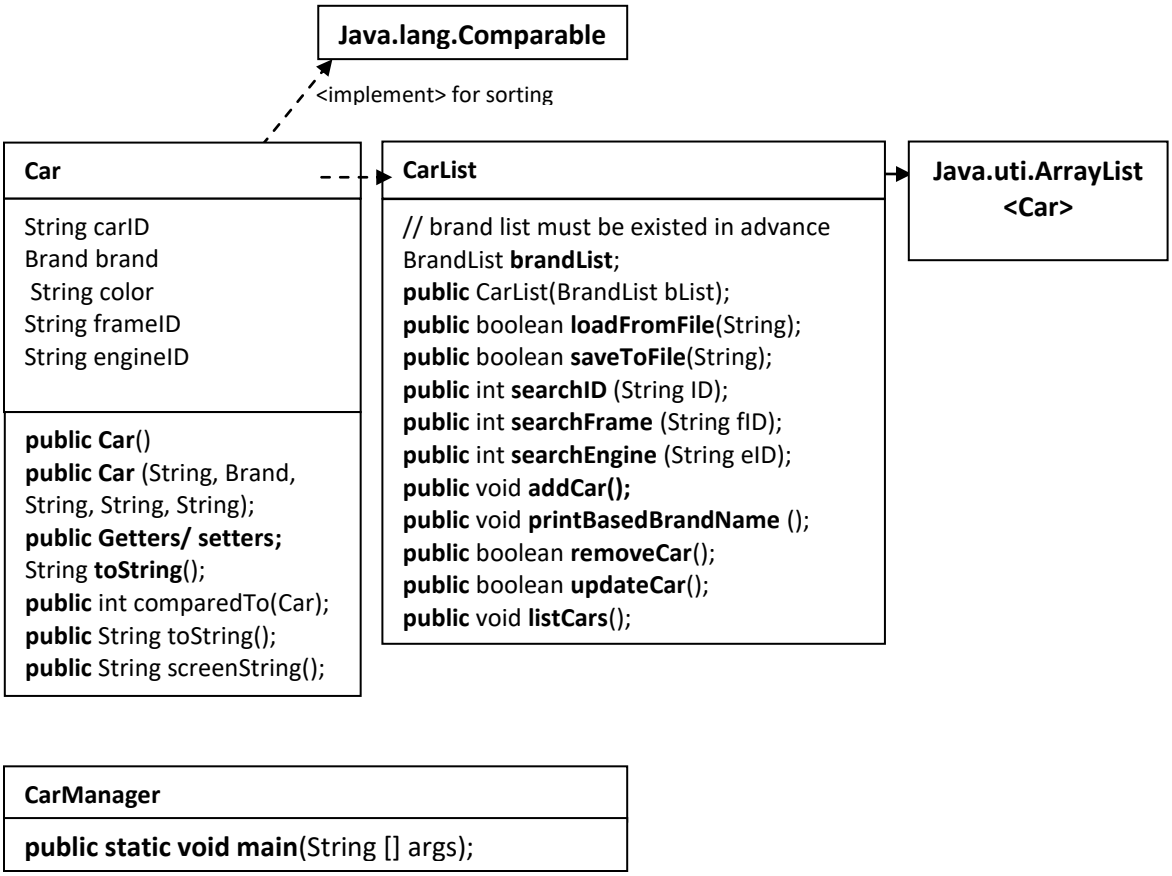
## 1- Class Design outline




## 2- Class Design in Details



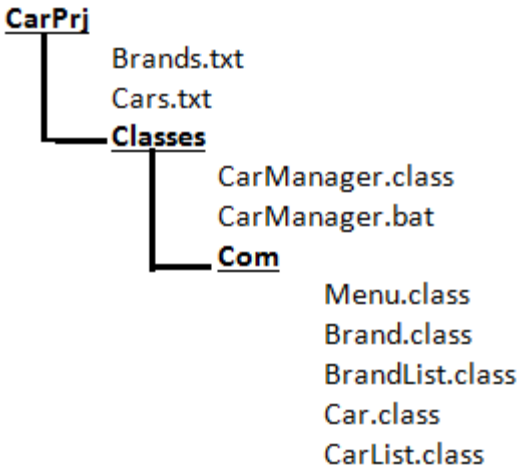






**3- Software Structure design**

Project name: CarPrj

Package structure:



## 4- Main Algorithm

| Class: Menu | Description and main idea of an algorithm |
|---|---|
| int int_getChoice(ArrayList<E> options) | Get user choice as an integer:<br>int response;<br>N = size of the list options;<br>For i= 0 .. N<br>Print out (i+1) + options.get(i);<br>Print out " Please choose an option 1..N:"<br>Receive response;<br>Return response; |
| E ref_getChoice(ArrayList<E> options) | Get user choice as an object in the list:<br>int response;<br>int N = size of the list options;<br>do {<br>response = int_getChoice(options)<br>}<br>While ( response<0 \|\| response>N);<br>Return options.get(response-1); |

| Class: Brand | Description and main idea of an algorithm |
|---|---|
| **public** String **toString**() | Return a string in the template: < brandID, brandName, soundBrand: price> |

| Class: BrandList | Description and main idea of an algorithm |
|---|---|
| **public** boolean **loadFromFile**(String filename) | File f = new File (filename);<br>If (f does not exist) exit the program;<br>Else {<br>Open file in text format for reading line-by-line;<br>While ( a line is read from file) {<br>Split the read line into parts;<br>Create a brand from input data(parts);<br>Add the brand to this list;<br>}<br>Close the file;<br>}<br>Return true; |
| **public** boolean **saveToFile**(String filename); | Open the file based on the filename to write data in line-by-line text format;<br>For each brand in the list {<br>Write the brand to file + "\n";<br>}<br>Close the file;<br>Return true; |

| | |
|---|---|
| **public** int **searchID** (String bID); | Search a brand based on brand ID. Return the existence position(int)<br>N= size of the list;<br>For I = 0 … N-1<br>  If (this.get(i).brandID == bID) return I;<br>Return -1; |
| **public** Brand **getUserChoice**(); | Transform the list to a menu, the user will choose a brand from this menu.<br>Menu mnu = new Menu();<br>Return (Brand)mnu. ref_getChoice(this); |
| **public** void **addBrand()** | Add a new Brand to the list.<br>Receive String ID, constraint. Input ID can not exist in the list<br>Receive String brandName. The brand name is not blank<br>Receive String soundBrand. The sound brand is not blank<br>Receive double price. Price >0<br>Create a new brand from inputted data;<br>Add a new brand to the list. |
| **public** void **updateBrand**(); | Update brand_name, sound_brand, price of an existed brand.<br>Receive brandID;<br>Pos = **searchID** (brandID);<br>if pos<0 print out “Not found!”;<br>Else{<br>  Receive String brandName. The brand name is not blank<br>  Receive String soundBrand. The sound brand is not blank<br>  Receive double price. Price >0<br>  Update new brandName, new sound brand, new price to the pos(th) brand.<br>} |
| **public** void **listBrands**(); | N = size of the list;<br>For I = 0.. N-1<br>  Print out this.get(i); |

| **Class: Car** | **Description and main idea of an algorithm** |
|---|---|
| **public** int comparedTo(Car c); | Used in the operation opf listing cars in ascending order of brand names.<br>int d = this.brand.brandName.compareTo(c.brand.brandName);<br>if (d!=0) return d;<br>// they are in the same brand, comparing based on their ID<br>return this.carID.compareTo(c.carID); |

| **Class: CarList** | **Description and main idea of an algorithm** |
|---|---|
| **public CarList**(BrandList bList); | Initialize a list based on the existed brand list;<br>brandList=bList; |
| **public** String **toString**(); | Associating fields to a string for writing a car to file<br>Return format < carID, brand.brandID, color, frameID, engineID> |
| **public** String **screenString**(); | Associating fields to a string for outputting a car to screen<br>Return format < brand, “\n”, car_ID, color, frameID, engineID> |
| **public** boolean **loadFromFile**(String filename); | File f = new File (filename);<br>If (f doesn’t exist) return false; |

| | |
|---|---|
| | Else {<br>Open file in text format for reading line-by-line;<br>While ( a line is read from file) {<br>Split the read line into parts;<br>Extract parts to carID, brandID, color, frameID, engineID<br>int pos= brandList.searchID(brandID);<br>Brand b = brandList.get(pos);<br>Create new car with data above;<br>Add new car to the list;<br>}<br>Close the file;<br>Return true;<br>} |
| **public** boolean **saveToFile**(String); | Open the file based on the filename to write data in line-by-line in text format;<br>For each car in the list {<br>Write the car to file + "\n";<br>}<br>Close the file;<br>Return true; |
| **public** int **searchID** (String carID); | Search a car based on car ID. Return the existed position(int)<br>N= size of the list;<br>For I = 0 … N-1<br>If (this.get(i).carID == carID) return I;<br>Return -1; |
| **public** int **searchFrame** (String fID); | Search a car by its frame ID. Use in checking frames are not duplicated.<br>N= size of the list;<br>For I = 0 … N-1<br>If (this.get(i).frameID == fID) return I;<br>Return -1; |
| **public** int **searchEngine** (String eID); | Search a car by its engine ID. Use in checking engines are not duplicated.<br>N= size of the list;<br>For I = 0 … N-1<br>If (this.get(i).engineID == eID) return I;<br>Return -1; |
| **public** void **addCar();** | Receive carID, carID must be not duplicated<br>Create a menu for choosing a brand;<br>Band b = (Brand)menu. ref_getChoice(brandList);<br>Receive color, color can not be blank<br>Receive frameID. It must be in the "F0000" and not be duplicated<br>Receive engineID. It must be in the "E0000" format and not be duplicated<br>Create a new car with inputted data;<br>Add a new car to the list |
| **public** void **printBasedBrandName** (); | Receive aPartOfBrandName;<br>N = size of the list;<br>Int count = 0; |

| | |
|---|---|
| | For I = 0.. N-1 {<br>  Car c = this.get(i);<br>  If (aPartOfBrandName is a sub-string of c.brand.brandName) {<br>  Print out c.screenString();<br>  count++;<br>}<br>If (count==0) print out “No car is detected!”; |
| **public** boolean **removeCar**(); | Remove a car based on it’s ID<br>Receive removedID;<br>Int pos = searchID(removedID);<br>If (pos<0) {<br>  print out “Not found!”<br>  return false;<br>}<br>Else{<br>  Remove (pos);<br>}<br>Return true; |
| **public** boolean **updateCar**(); | Update a car based on it’s ID<br>Receive updatedID;<br>Int pos = searchID(updatedID);<br>If (pos<0) {<br>  print out “Not found!”<br>  return false;<br>}<br>Else{<br>  Create a menu for choosing a brand;<br>  Band b = (Brand)menu. ref_getChoice(brandList);<br>  Receive color, color can not be blank<br>  Receive frameID. It must be in the “F0000” and not be duplicated<br>  Receive engineID. It must be in the “E0000” format and not be duplicated<br><br>  Update brand, color, Frame ID, machine ID for the pos(th) car.<br>}<br>Return true; |
| **public** void **listCars**(); | Listing cars in ascending order of brand names.<br>Sorting cars // Collection.sort(this);<br>N = size of the list;<br>For i=0 … N-1 {<br>  Car c = this.get(i);<br>  Print out c.screenString();<br>} |

| **Class: CarManager** | **Description and main idea of an algorithm** |
|---|---|
| Public static void main(String[] args) | Main program<br>Create ArrayList ops of strings containing options of the program; |

| | |
|---|---|
| | Create an empty brandList;<br>Load brands from the file brands.txt to brandList;<br>Create an empty carList using brandList;<br>Load cars from the file cars.txt to carList;<br>Int choice;<br>Create a menu;<br>Do{<br>  Choice = menu.int_getChoice(ops);<br>  Switch (choice) {<br>  Case 1: brandList.listBrands(); break;<br>  ……<br>  }<br>}<br>While (choice>0 && choice <=ops.size()); |

**5- Test cases**

| Test | Option | Objective | Requirements |
|---|---|---|---|
| 1 | 1 | List all brands<br>[0.5 point] | All brands in the file must be shown correctly |
| 2 | 2 | Add a new brand<br>[1 point] | • Brand ID can not be duplicated.<br>• Brand name can not be blank.<br>• Sound manufacturer can not be blank.<br>• Price must be a positive real number.<br>Add:<br>B7-MS2, BMW 730Li M, Alpine: 4.050 |
| 3 | 3 | Search a brand based on it's ID [1 point] | Test 2 cases:<br>- BrandID = B5-30 → Not found<br>- BrandID = B5-18 → Brand result is shown. |
| 4 | 4 | Update a brand<br>[1 point] | • Brand name can not be blank.<br>• Sound manufacturer can not be blank.<br>• Price must be a positive real number.<br>Update: B7-MS, BMW 730Li M Sport, Harman Kardon: 4.319<br>To: B7-MS, BMW 730Li MS, Sony: 4.150 |
| 5 | 5 | Save brands to the file, named brands.txt<br>[1 point] | This operation must be successful.<br>Open the file to check its content. |
| 6 | 6 | List all cars in ascending order of brand names<br>[1 point] | All cars in the file must be shown in ascending order of brand names and their ID. |
| 7 | 7 | List cars based on a part of an input brand name<br>[1 point] | Input: "960": No result<br>Input: "730": All cars of the brand 730 must be shown. |
| 8 | 8 | Add a car | - Car ID can not be duplicated (C05, C08) |

| | | | |
|---|---|---|---|
| | | [1 point] | - Brand ID must have existed and it must be inputted using a menu. Choose B5-18<br>- Color can not be blank.( black/ yellow)<br>- Frame ID can not be blank and must be in the "F00000" format and can not be duplicated (K0123/ F12345/ F12352).<br>- Engine ID can not be blank and must be in the "E00000" format and can not be duplicated (M0123/ M12345/ E12352)<br><br>Add: C08, B5-18, yellow, F12352, E12352 → successful |
| 9 | 9 | Remove a car based on its ID<br>[1 point] | C10 → Not found.<br>C06 → Remove successfully. |
| 10 | 10 | Update a car based on its ID<br>[1 point] | ID = C10 → Not found.<br>ID= C03<br>Update: C03, B7-MS, orange, F12347, E12347<br>To: C03, B5-18, brown, F99999, E99999<br>- Brand ID must have existed and it must be inputted using a menu. Choose B5-18<br>- Color can not be blank. ( black/ brown)<br>- Frame ID can not be blank and must be in the "F00000" format and can not be duplicated (K0123/ F12345/ F99999).<br>- Engine ID can not be blank and must be in the "E00000" format and can not be duplicated (M0123/ M12345/ E99999) |
| 11 | 11 | Save cars to file<br>[0.5 point] | This operation must be successful.<br>Open the file to check its content. |