# Purpose

The purpose of this document is to show the steps needed for constructing the Dealership Scenarios given in Project 1. Within this document I will explain the breakdown of each of the classes used along with their methods.

# Introduction

There are some things that need to be known prior to diving into this technical document. Below are some terms you should know:

* Array: An array is a collection of the same data types stored in indexes starting with 0 and going until the set limit minus one(ie int[]arr = new int[5] | {0,1,2,3,4}). Arrays can be one dimensional, having only one dimensional limit, or two dimensional, having two different dimensional limits
* ArrayList: Similar to an array, this stores same data types into indexes but unlike the array before they are one dimensional and have no set limit on creation(ie ArrayList<Car> carlist = new ArrayList<>()).To add to an arraylist you just need the name\_of\_the\_array.add(thing\_you\_are\_adding)
* File: File is a class that is called in order to make or retrieve information from a file stored in the program folder.
* Scanner: Scanner is a class used for taking inputs. These inputs can come from files or from the user. Scanners must be closed at the end of usage to not waste valuable resources(ie scannerName.close()).
* PrintWriter: PrintWriter is another class that uses the file class as a part of its methods. Printwriter uses similar commands to the System.out.print commands in order to write information from the program to some sort of output file. Just like the scanner these have to be closed after you are done with them, because not only do they waste resources, if you don’t close it, it is like closing an application without saving.
* For Loop: A for loop is a loop within the code that repeats a specific number of times before moving on. This is set within the conditions when it is declared(ie for(int x = 0; x< 10; x++)).
* While Loop: A while loop is a loop used when you want to repeat code until a specific condition is met, Usually you use a true or false statement to break the loop(ie while(repeat == true)). Until that condition is met the loop continues indefinitely.
* If-Else Loop:If and else loops are conditional loops that allow you to use the same code to go down different paths. If they are not put within one of the loops above they will skip all other paths automatically.
* UML: A UML diagram is a visual representation of the breakdown of a class. These diagrams start with the name of the class, followed by the variables in the class, and finish with the methods to be used in the class. These are a good starting point for making code as it helps you break down what you want to make for the code.

# UML Diagrams

| Car |
| --- |
| -double: price  -int: year  -String: make  -String: model |
| +Car(int year,double price, String make, String model)  +int getYear()  +double getPrice()  +String getMake()  +String getModel() |

| VendingMachine |
| --- |
| -Car[][] inventory |
| +VendingMachine(Car[][] carlist)  +String sortPrice()  +String sortYear()  +void addCar(int row,int column, Car car)  +void DisplayInventory()  +void Retrieve()  +void PrintInventory() |

# Story Breakdowns

For this program I needed to have it do certain tasks based on different stories. Below are the stories explained and how I broke them down into tasks.

#### Story 1:

In story one we needed to act as if we are the dealership owner and compile a full inventory list for our car vending machine.

1. Make a vending machine.
   1. Create a vending machine constructor that uses a 2D array.
2. Add vehicles to the vending machine.
   1. Populate the vending machine from a file.
3. Make sure that two slots aren’t the same for the vehicles.
   1. Make sure that the inputs don’t override each other.
4. Make the list viewable outside of the program.
   1. Make an output file for the program to write to.

#### Story 2:

In this scenario I am an employee trying to pull up the list of the inventory for the customer to look at. This one built off of the previous story and instead of sending the information to a file I needed to display it for the user.

1. Make a populated vending machine.
   1. Populate the vending machine like in story 1.
2. Format the information for display to the user.
   1. Set up a formatted output for filled and unfilled inventory slots.
3. Have an output to the console for the user to view the inventory.
   1. Similar to the output file in story 1 but instead send information to the console.

#### Story 3:

In story 3 we were tasked with sorting the inventory by two different elements of our Car object. One of them being Price and the other Year of the vehicle.

1. Populate the vending machine.
   1. Populate the vending machine like in story 1.
2. Develop a method for sorting the 2D array.
   1. Take the 2D array and add filled locations to an Array List.
   2. Then take the Array List and add those elements to a 1D array.
3. Take user input for which element to sort by(either price or year).
   1. Have one option be for price and the other for year
4. Display sorted array to user.
   1. If the user wants sorting by price, sort the array and display it in the console sorted by price.
   2. If the user wants sorting by year, sort the array and display it in the console sorted by year.

#### Story 4:

Story 4 was one where we need to retrieve a vehicle from a specific slot that the user requests so that they can test drive it. The story never asks us to remove the vehicle but I could have added it by taking the vehicle and putting it in a temp spot until the user returns with the vehicle.

1. Make a populated vending machine.
   1. Populate the vending machine like in story 1.
2. Gather user input for lot and spot positions.
   1. User scanner and system prompts to collect information from user
3. Check for the vehicle in location.
   1. Take the user inputs and search the array for that specific location
4. Return information of vehicle in location user requested or return that the location is empty.
   1. If a vehicle is there, output vehicle information.
   2. If the spot is empty let the user know that no vehicle was found.

#### Story 5:

Story 5 was a combination of all the other stories. You are making a dealerships inventory system for an employee to use to either add a new vehicle, display the inventory for a customer, retrieve a car for the customer, sort the available inventory by price or year for a customer, or exit the system.

1. Develop a menu to be displayed.
   1. Make a loop that works off of a user input and set up initial display and loop display text for the console.
2. Populate the vending machine for the dealership.
   1. Add vehicles to the vending machine from a file.
3. Take user input to navigate the menu.
   1. Add a scanner to take user input for the menu.
4. Make implementations of the other stories within the menu.
   1. Make an option for displaying inventory.
   2. Make an option to sort by price.
   3. Make an option to sort by year.
   4. Make an option to add to the inventory.
   5. Make an option to retrieve a vehicle.
5. Have it loop back to the menu until the user wishes to exit.
   1. Use a While loop with a condition that only goes off when user press option 6
6. Give an exit to the loop that displays a goodbye message.
   1. Choose a method for breaking the loop(boolean).
   2. Add a message before loop break condition to say goodbye.

# Method Breakdown

## Car methods

#### public Car(int year, double price, String make, String Model)

This is the Constructor method for what a car is in the code below we set the input values of the created car into that objects price, year, make, and model variable slots.

//Car constructor

public Car(int year, double price, String make, String model)

{

this.year = year;

this.price = price;

this.make = make;

this.model = model;

}

### Getters

Getters are methods used for retrieving information about the object in question. For the Car we have a total of four getters: Year, Price, Make, and Model.

#### getYear()

public int getYear()

{

return year;

}

#### getPrice()

public double getPrice()

{

return price;

}

#### getMake()

public String getMake()

{

return make;

}

#### getModel()

public String getModel()

{

return model;

}

## Vending Machine Methods

#### public VendingMachine(Car[][] carlist)

This is the constructor method for a car Vending Machine. All it requires is a two dimensional array for inventory control.

public VendingMachine(Car[][] carlist)

{

inventory = carlist;

}

#### sortPrice()

This method breaks down the two dimensional array into an array list which then is put into a one dimensional array for sorting portion of the method. During the sorting portion we used an insertion sort to sort the different objects by their price.

public String sortPrice()

{

ArrayList<Car> list = new ArrayList<>();

//loop for populating arrayList

for(int r = 0; r < inventory.length; r++)

{

for(int c = 0; c< inventory[0].length; c++)

{

if (inventory[r][c] !=null)

{

list.add(inventory[r][c]);

}

}

}

Car[] carlist = new Car[list.size()];

//loop for adding the objects from the array list into a 1d array for sorting

for(int z = 0; z< list.size();z++)

{

carlist[z] = list.get(z);

}

//sorting method used in both sorts to list the cheapest price or later the oldest car first.

int n = carlist.length;

for(int x = 1; x < n;x++)

{

double key = carlist[x].getPrice();

int y = x-1;

while(y >=0 && carlist[y].getPrice() > key)

{

carlist[y+1].price = carlist[y].getPrice();

y--;

}

carlist[y+1].price = key;

}

String sortedList = "";

//populate the string accordingly for return to main.

for(int car = 0; car < carlist.length; car++)

{

sortedList = sortedList+ "$"+carlist[car].getPrice()+" "+ carlist[car].getYear() + " " + carlist[car].getMake() + " "+carlist[car].getModel()+"\n";

}

System.***out***.println("");

//returns a string format of the array

return sortedList;

}

This is then compiled into a string that will be outputted to the main class for display to the user.

1)Add a Car

2)Display inventory

3)Retrieve Vehicle

4)Show Inventory(By Price)

5)Show Inventory(By Year)

6)Close System

4

$20000.0 2018 Toyota Corolla

$24000.0 2016 Honda Accord

$26000.0 2019 Ford Mustang

$28000.0 2021 Hyundai Ioniq5

$30000.0 2017 Nissan Altima

#### sortYear()

This method breaks down the two dimensional array in the same way as the sortPrice method but when sorting the one dimensional array it sorts by comparing each object’s year value rather than the price value.

public String sortYear()

{

ArrayList<Car> list = new ArrayList<>();

for(int r = 0; r < inventory.length; r++)

{

for(int c = 0; c< inventory[0].length; c++)

{

if (inventory[r][c] !=null)

{

list.add(inventory[r][c]);

}

}

}

Car[] carlist = new Car[list.size()];

for(int z = 0; z< list.size();z++)

{

carlist[z] = list.get(z);

}

int n = carlist.length;

for(int x = 1; x < n;x++)

{

int key = carlist[x].getYear();

int y = x-1;

while(y >=0 && carlist[y].getYear() > key)

{

carlist[y+1].year = carlist[y].getYear();

y--;

}

carlist[y+1].year = key;

}

String sortedList = "";

DecimalFormat money = new DecimalFormat("#.00");

for(int car = 0; car < carlist.length; car++)

{

sortedList = sortedList+ ""+carlist[car].getYear()+" $"+ money.format(carlist[car].getPrice()) + " " + carlist[car].getMake() + " "+carlist[car].getModel()+"\n";

}

System.***out***.println("");

return sortedList;

}

At the bottom you can see the return statement that sends the string that was formed back to the main class to be displayed.

1)Add a Car

2)Display inventory

3)Retrieve Vehicle

4)Show Inventory(By Price)

5)Show Inventory(By Year)

6)Close System

5

2016 $20000.00 Toyota Corolla

2017 $24000.00 Honda Accord

2018 $26000.00 Ford Mustang

2019 $28000.00 Hyundai Ioniq5

2021 $30000.00 Nissan Altima

#### addCar(int row, int column, Car car)

This method is used to add user inputted vehicles into the inventory array or not if the position selected is already filled.

public void addCar(int row, int column, Car car)

{

//establishes the variables inputted

Car vehicle = car;

int r = row;

int c = column;

//adds the vehicle to the spot if able to

if (inventory[r][c] == null)

{

inventory[r][c] = vehicle;

}

//message displayed if vehicle was added to an occupied spot.

else

{

System.***out***.println("That spot is already taken. Car not added to inventory.");

}

}

1)Add a Car

2)Display inventory

3)Retrieve Vehicle

4)Show Inventory(By Price)

5)Show Inventory(By Year)

6)Close System

1

What is the make of your car?

Marussia

What is the model of your car?

B2

What is the year of your car?

2012

What is the price of your car?

280000

Now look at the stock below:

[0][0]

Empty

Spot

-----------------------------------

[0][1]

Price: $20000.00

Year: 2016

Make: Toyota

Model: Corolla

-----------------------------------

[0][2]

Empty

Spot

-----------------------------------

[0][3]

Empty

Spot

-----------------------------------

[0][4]

Empty

Spot

-----------------------------------

[1][0]

Empty

Spot

-----------------------------------

[1][1]

Empty

Spot

-----------------------------------

[1][2]

Price: $24000.00

Year: 2017

Make: Honda

Model: Accord

-----------------------------------

[1][3]

Empty

Spot

-----------------------------------

[1][4]

Empty

Spot

-----------------------------------

[2][0]

Empty

Spot

-----------------------------------

[2][1]

Empty

Spot

-----------------------------------

[2][2]

Empty

Spot

-----------------------------------

[2][3]

Price: $26000.00

Year: 2018

Make: Ford

Model: Mustang

-----------------------------------

[2][4]

Empty

Spot

-----------------------------------

[3][0]

Price: $28000.00

Year: 2019

Make: Hyundai

Model: Ioniq5

-----------------------------------

[3][1]

Empty

Spot

-----------------------------------

[3][2]

Empty

Spot

-----------------------------------

[3][3]

Empty

Spot

-----------------------------------

[3][4]

Empty

Spot

-----------------------------------

[4][0]

Empty

Spot

-----------------------------------

[4][1]

Empty

Spot

-----------------------------------

[4][2]

Price: $30000.00

Year: 2021

Make: Nissan

Model: Altima

-----------------------------------

[4][3]

Empty

Spot

-----------------------------------

[4][4]

Empty

Spot

-----------------------------------

What lot would you like to story vehicle in:

0

What spot would you like to story vehicle in:

0

#### DisplayInventory()

The DisplayInventory method sends each index within the vending machine array(inventory) to the console for display to the user. It even displays the empty positions within the array.

public void DisplayInventory()

{

//loop for formatted display

for(int r = 0; r< inventory.length; r++)

{

for(int c = 0; c < inventory[0].length; c++)

{

if(inventory[r][c] == null)

{

System.***out***.printf("\n[%d][%d]\nEmpty\nSpot\n-----------------------------------",r,c);

}

else

{

System.***out***.printf("\n[%d][%d]\nPrice: $%.2f\nYear: %d\nMake: %s\nModel: %s\n-----------------------------------"

,r,c,inventory[r][c].getPrice(),inventory[r][c].getYear(),inventory[r][c].getMake(),inventory[r][c].getModel());

}

}

}

}

1)Add a Car

2)Display inventory

3)Retrieve Vehicle

4)Show Inventory(By Price)

5)Show Inventory(By Year)

6)Close System

2

[0][0]

Price: $280000.00

Year: 2012

Make: Marussia

Model: B2

-----------------------------------

[0][1]

Price: $20000.00

Year: 2016

Make: Toyota

Model: Corolla

-----------------------------------

[0][2]

Empty

Spot

-----------------------------------

[0][3]

Empty

Spot

-----------------------------------

[0][4]

Empty

Spot

-----------------------------------

[1][0]

Empty

Spot

-----------------------------------

[1][1]

Empty

Spot

-----------------------------------

[1][2]

Price: $24000.00

Year: 2017

Make: Honda

Model: Accord

-----------------------------------

[1][3]

Empty

Spot

-----------------------------------

[1][4]

Empty

Spot

-----------------------------------

[2][0]

Empty

Spot

-----------------------------------

[2][1]

Empty

Spot

-----------------------------------

[2][2]

Empty

Spot

-----------------------------------

[2][3]

Price: $26000.00

Year: 2018

Make: Ford

Model: Mustang

-----------------------------------

[2][4]

Empty

Spot

-----------------------------------

[3][0]

Price: $28000.00

Year: 2019

Make: Hyundai

Model: Ioniq5

-----------------------------------

[3][1]

Empty

Spot

-----------------------------------

[3][2]

Empty

Spot

-----------------------------------

[3][3]

Empty

Spot

-----------------------------------

[3][4]

Empty

Spot

-----------------------------------

[4][0]

Empty

Spot

-----------------------------------

[4][1]

Empty

Spot

-----------------------------------

[4][2]

Price: $30000.00

Year: 2021

Make: Nissan

Model: Altima

-----------------------------------

[4][3]

Empty

Spot

-----------------------------------

[4][4]

Empty

Spot

-----------------------------------

#### Retrieve()

The Retrieve method is used to look at a specific spot in the vending machine. This is used for grabbing a car for a test drive by a customer.It also uses a Thread.sleep counter in order to simulate a loading time for the retrieval of the vehicle.

public void Retrieve()

{

Scanner user = new Scanner(System.***in***);

System.***out***.println("\n\nWhat lot is the car you are looking for?");

int lot = user.nextInt();

System.***out***.println("What spot is the car on this lot?");

int spot = user.nextInt();

if(inventory[lot][spot] != null)

{

System.***out***.println("Retrieving Car");

try

{

//timer for retrieval to act like a real system

Thread.*sleep*(3000);

}

catch (InterruptedException e)

{

e.printStackTrace();

}

System.***out***.printf("\n\n%d $%.2f %s %s\n",inventory[lot][spot].getYear(),inventory[lot][spot].getPrice(),

inventory[lot][spot].getMake(),inventory[lot][spot].getModel());

}

else

{

System.***out***.println("The chosen spot seems to be empty.");

}

}

1)Add a Car

2)Display inventory

3)Retrieve Vehicle

4)Show Inventory(By Price)

5)Show Inventory(By Year)

6)Close System

3

What lot is the car you are looking for?

0

What spot is the car on this lot?

0

Retrieving Car

2012 $280000.00 Marussia B2

#### PrintInventory()

This method uses a PrintWriter and File in order to take the inventory two dimensional array and write it to an output file for viewing outside of the program itself.

public void PrintInventory()

{

File out = new File("inventory.txt");

try

{

PrintWriter writing = new PrintWriter(out);

for(int r = 0; r< inventory.length; r++)

{

for(int c = 0; c < inventory[0].length; c++)

{

if(inventory[r][c] == null)

{

writing.printf("\n[%d][%d]\nEmpty\nSpot\n-----------------------------------",r,c);

}

else

{

writing.printf("\n[%d][%d]\nPrice: $%.2f\nYear: %d\nMake: %s\nModel: %s\n-----------------------------------"

,r,c,inventory[r][c].getPrice(),inventory[r][c].getYear(),inventory[r][c].getMake(),inventory[r][c].getModel());

}

}

}

writing.close();

}

catch (FileNotFoundException e)

{

e.printStackTrace();

}

}

[0][0]

Price: $280000.00

Year: 2012

Make: Marussia

Model: B2

-----------------------------------

[0][1]

Price: $20000.00

Year: 2016

Make: Toyota

Model: Corolla

-----------------------------------

[0][2]

Empty

Spot

-----------------------------------

[0][3]

Empty

Spot

-----------------------------------

[0][4]

Empty

Spot

-----------------------------------

[1][0]

Empty

Spot

-----------------------------------

[1][1]

Empty

Spot

-----------------------------------

[1][2]

Price: $24000.00

Year: 2017

Make: Honda

Model: Accord

-----------------------------------

[1][3]

Empty

Spot

-----------------------------------

[1][4]

Empty

Spot

-----------------------------------

[2][0]

Empty

Spot

-----------------------------------

[2][1]

Empty

Spot

-----------------------------------

[2][2]

Empty

Spot

-----------------------------------

[2][3]

Price: $26000.00

Year: 2018

Make: Ford

Model: Mustang

-----------------------------------

[2][4]

Empty

Spot

-----------------------------------

[3][0]

Price: $28000.00

Year: 2019

Make: Hyundai

Model: Ioniq5

-----------------------------------

[3][1]

Empty

Spot

-----------------------------------

[3][2]

Empty

Spot

-----------------------------------

[3][3]

Empty

Spot

-----------------------------------

[3][4]

Empty

Spot

-----------------------------------

[4][0]

Empty

Spot

-----------------------------------

[4][1]

Empty

Spot

-----------------------------------

[4][2]

Price: $30000.00

Year: 2021

Make: Nissan

Model: Altima

-----------------------------------

[4][3]

Empty

Spot

-----------------------------------

[4][4]

Empty

Spot

-----------------------------------

# Main Program

The main portion of the program where everything is called from is the Main method. Within this method we populate the initial inventory using a file and a scanner to input the information.

public static void main(String [] args)

{

//empty 2d array for vending machine

Car[][] myInv = new Car[5][5];

//creation of vending machine

VendingMachine VM = new VendingMachine(myInv);

//Initial file for inventory of dealership

File input = new File("cars1.txt");

try

{

//auto scanner for the file to populate the array

Scanner entry = new Scanner(input);

while(entry.hasNext())

{

//positions in 2d array for entry

int row = entry.nextInt();

int col = entry.nextInt();

//creates a new car object for the array

Car myCar = new Car(entry.nextInt(),entry.nextDouble(),entry.next(),entry.next());

VM.addCar(row, col, myCar);

}

entry.close();

}

//statement that is run if file can't be found

catch (FileNotFoundException e)

{

e.printStackTrace();

}

After the initial array has been set up we head into the menu for the program that uses another scanner to take user inputs.

//menu scanner titled after story 5 scenario

Scanner Story5 = new Scanner(System.***in***);

//loop break condition

boolean loop = true;

System.***out***.println("Welcome to the inventory system of Mooty Motors.\nHow may I assist you today?");

//start of the loop

while(loop == true)

{

System.***out***.println("\n\n1)Add a Car\n2)Display inventory\n3)Retrieve Vehicle\n4)Show Inventory(By Price)\n5)Show Inventory(By Year)"

+ "\n6)Close System\n");

int answer = Story5.nextInt();

Welcome to the inventory system of Mooty Motors.

How may I assist you today?

1)Add a Car

2)Display inventory

3)Retrieve Vehicle

4)Show Inventory(By Price)

5)Show Inventory(By Year)

6)Close System

From these inputs you have multiple options for how the code will run. These if statements lead you down different paths until you select the close path(option 6). In option 1 we allow the user to add to the inventory that was established.

if(answer == 1)

{

System.***out***.println("What is the make of your car?");

String make = Story5.next();

System.***out***.println("What is the model of your car?");

String model = Story5.next();

System.***out***.println("What is the year of your car?");

int year = Story5.nextInt();

System.***out***.println("What is the price of your car?");

double price = Story5.nextDouble();

Car newCar = new Car(year,price,make,model);

System.***out***.println("Now look at the stock below: ");

VM.DisplayInventory();

System.***out***.println("\nWhat lot would you like to story vehicle in: ");

int row = Story5.nextInt();

System.***out***.println("What spot would you like to story vehicle in: ");

int col = Story5.nextInt();

if(VM.inventory[row][col] == null)

{

VM.addCar(row, col, newCar);

}

else

{

System.***out***.println("Car was not added. Now going back to beginning.");

}

}

In options 2 through 5 we have quick method calls within the other classes making the portion in the main method extremely short. Option 2 allows the user to see the entire vending machine inventory including empty slots. Option 3 allows the user to attempt retrieval of a car from a spot and will output a response accordingly. Options 4 and 5 are pretty similar with both of them sorting the actual inventory based off of either price or year of the cars.

//retrieves and displays the inventory of the vending machine

if(answer == 2)

{

VM.DisplayInventory();//Story 2 Scenario

}

//retrieve a vehicle for a customer

if(answer == 3)

{

VM.Retrieve();//Story 4 Scenario

}

//sorts the actual inventory by price

if(answer == 4)

{

System.***out***.print(VM.sortPrice());//Story 3 Scenario part 1

}

//sorts the actual inventory by year

if(answer == 5)

{

System.***out***.print(VM.sortYear());//Story 3 Scenario part 2

}

Finally we have option 6 of the menu which allows the user to exit the loop and have the program close.

//used to exit the loop

if(answer == 6)

{

System.***out***.println("Have a good rest of your day.\n\n---Goodbye---");

VM.PrintInventory();//Story 1 Scenario

loop = false;

}

}

Story5.close();

Have a good rest of your day.

---Goodbye---