Used Car Dealership (UCD)

Team 4

Jarosław Rybak (Captain):	Iterator Pattern
Mohammad Ariz Haider:	Memento Pattern
Satar Hassni:	Singleton Pattern
Anthony Ferrara:	Observer Pattern
Mohammad U Uddin:	Factory Pattern
Fahim Ahmed:	Strategy Pattern

Project Description

A system capable to keeping, displaying, and editing an inventory of cars for the purpose of selling them.

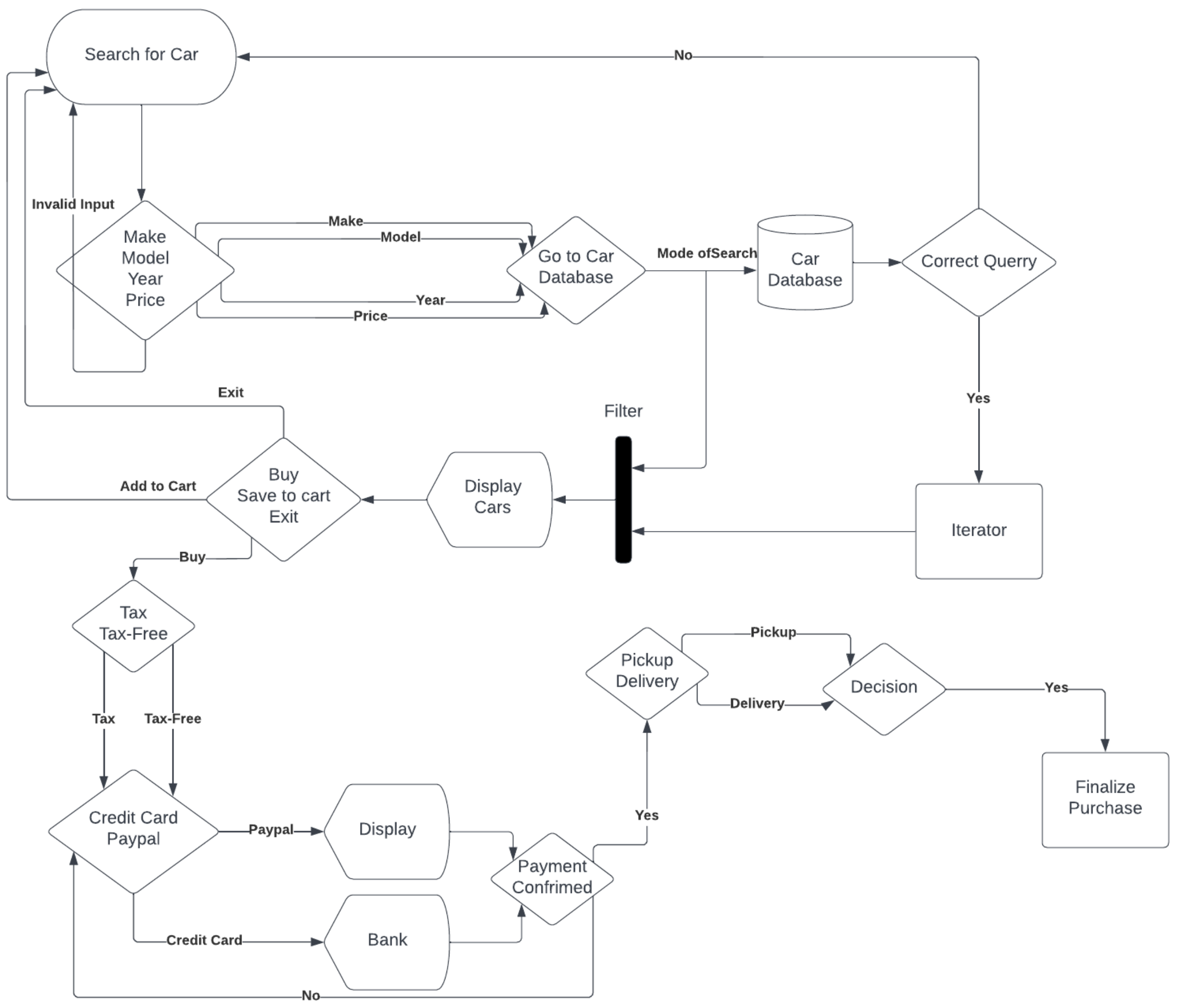
Project Requirements/Specifications

This system will allow users to effectively search cars from a database and proceed to a purchase. There are tax related options as well as multiple purchasing methods. Finally there are also acquisition options before the finalized purchase.

Use Case

- Actor: Sales representative or Customer.
- Preconditions: Cars in the database ready for sale.
- Triggers: Customer would like buy or view a car(s).
- Main Success scenario: Cars are requested by filter. Cars are returned for display. A car(s) is chosen for purchase. Payment method is verified. Acquisition method is chosen. Purchase is finalized.
- Alternative paths: Invalid filter is chosen. Return to search for cars.
- Alternative paths: Invalid querry is returned. Return to search for cars.
- Alternative paths: Exit or save to cart is chosen. Return to search for cars.
- Alternative paths: Payment is not confirmed. Return to payment options.
- Postcondition: Purchase is finalized.

Flow Chart

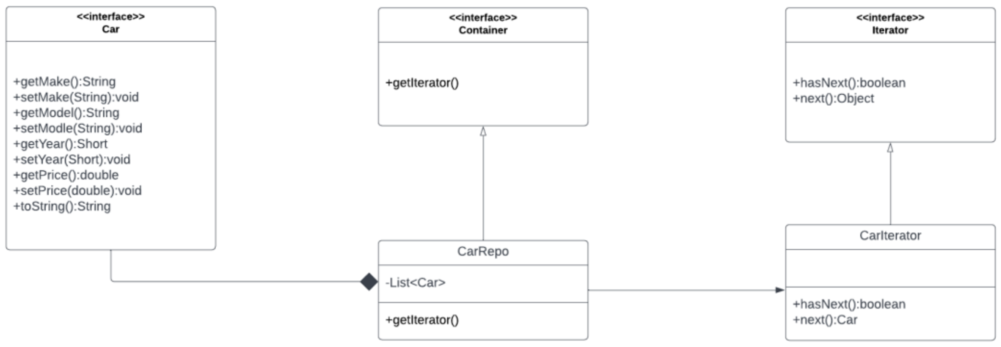


Search for Car
No
Invalid Input
Make
Model
Year
Price
Make
Model
Year
Price
Go to Car Database
Mode ofSearch
Car Database
Correct Querry
Yes
Exit
Filter
Add to Cart
Buy
Save to cart
Exit
Display Cars
Iterator
Buy
Tax
Tax-Free
Tax
Tax-Free
Pickup
Delivery
Pickup
Delivery
Decision
Yes
Finalize Purchase
Yes
Credit Card
Paypal
Paypal
Display
Payment Confrimed
Credit Card
Bank
No

Iterator Pattern

Jarosław Rybak

UML Diagram



Code

Iterato.java

```java
package IteratorPattern;

public interface Iterator
{
	public boolean hasNext();
	public Object next();
}
```

Container.java

```java
package IteratorPattern;

public interface Container
{
	public Iterator getIerator();
}
```

Car.java

```java
package IteratorPattern;

public interface Car
{
	public String getMake();
	public void setMake(String make);
	public String getModel();
	public void setModel(String model);
	public short getYear();
	public void setYear(short year);
	public double getPrice();
	public void setPrice(double price);
	public String toString();
}
```

CarRepo.java

```
package IteratorPattern;

import java.util.List;

public class CarRepo implements Container
{
	private  List<Car> Cars;

	public CarRepo()
	{
		DemoList DL =  new DemoList();
		Cars = DL.getCars(); //This would be replace by the real list of cars
	}

	//For unit testing
	public CarRepo(List<Car> cars)
	{
		Cars = cars;
	}

	@Override
	public Iterator getIerator()
	{
		return new CarIterator();
	}

	private class CarIterator implements Iterator
	{
		int index;

		@Override
		public boolean hasNext()
		{
			if(index < Cars.size())
			{
		     return true;
		 }
			return false;
		}

		@Override
		public Car next()
		{
			if(this.hasNext()){
		     return Cars.get(index++);
		   }
```

```
				return null;
			}
		}

		//For unit testing
		public List<Car> getCars()
		{
			return Cars;
		}

		//for unit testing
		public void setCars(List<Car> cars)
		{
			Cars = cars;
		}
}
```

Unit Test

<u>CarRepoTest.java</u>

```
package IteratorPattern;



import static org.junit.Assert.*;



import java.util.ArrayList;

import java.util.List;



import org.junit.After;

import org.junit.AfterClass;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.Test;



public class CarRepoTest

{

	private static DemoList DL;
```

```
}

private CarRepo CR;

private Iterator ier;


@BeforeClass

public static void setUpBeforeClass() throws Exception

{

        //Generating RNG list for testing.

        DL = new DemoList();

}


@AfterClass

public static void tearDownAfterClass() throws Exception

{


}


@Before

public void setUp() throws Exception

{

        //Making a clone for each test to not edit original list

        List<Car> ListClone =  new ArrayList<Car>();

        for(int i = 0;i<DL.getCars().size();i++)

        {

                ListClone.add(DL.getCars().get(i));

        }
```

```
                CR = new CarRepo(ListClone);


                ier = CR.getIerator();

        }


        @After

        public void tearDown() throws Exception

        {


        }


        //Testing Clone: Testing if editing List generator edits usable list

        @Test(expected=IndexOutOfBoundsException.class)

        public void testSetCars()

        {

                DL.getCars().add(DL.RNGCar());


                assertNull("Testing if editing original list will edit CarRepo list.",
CR.getCars().get(DL.getCars().size()));

        }


        //Testing Clone: testing if both lists are the same

        @Test

        public void test_Lists()

        {

                for(int i =0; i<DL.getCars().size()||i<CR.getCars().size();i++)
```

```
		{
			assertEquals("Testing Car: "+i+" If failed scrap all
tests",DL.getCars().get(i),CR.getCars().get(i));
		}
	}

	//Testing Iterator: Testing if 2 iterators are the same
	@Test
	public void testing_Diff_Iterators()
	{
		assertFalse("Testing if 2 Iterators are the same.",ier.equals(CR.getIerator()));
	}

	//Testing Iterator: Testing if 2 iterators give the same output
	@Test
	public void testing_Diff_output()
	{
		Iterator newIer = CR.getIerator();
		while(ier.hasNext()||newIer.hasNext())
		{
			assertEquals("Testing same list but diff iterator
output.",ier.next(),newIer.next());
		}
	}

	//Testing hasNext: Testing if the first hasNext returns true
```

```
@Test
public void hasNext_BaseCase()
{
	assertTrue("Teting hasNext base case",ier.hasNext());
}

//Testing hasNext: Testing if ending hasNext returns false
@Test
public void hasNext_OverFlow()
{
	while(ier.hasNext())
	{
		ier.next();
	}

	assertFalse("Testing hasNext overflow",ier.hasNext());
}

//Testing Next: Testing all Cars at once
@Test
public void next_compared_orginal_List()
{
	for(int i =0; i<DL.getCars().size()||i<CR.getCars().size();i++)
	{
		assertEquals("Testing Car: "+i,DL.getCars().get(i),ier.next());
```

```
                    }
          }
}
```

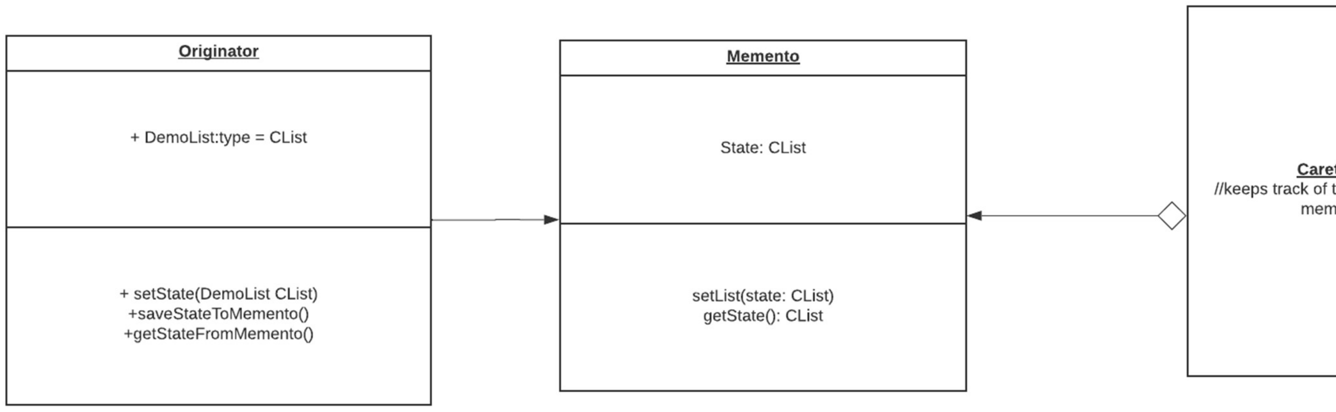
Component Test

Make 2 different instances of the Iterator. Use them separately for different purposes without one interfering with the other. Iterator 1 will filter Cars from “Make”: “Beta” and return them. Iterator 2 will return Cars in ascending order of year. Neither will know of the other nor have an effect on the others output.

MOHAMMAD ARIZ HAIDER

MEMENTO PATTERN

UML DIAGRAM: MEMENTO



REVELANT CODE TO MEMENTO PATTERN

Originator

```
package Memento;
public class Originator //Creates the save time
{
	DemoList CList; //CList = Car List

	public DemoList getState()
	{
		return CList;
	}

	public void setState(DemoList  CList) // saves a sate of the current list
	{
		this.CList = CList;
	}

	public Memento saveStateToMemento()
	{
		return new Memento(CList);
	}

	public void getStateFromMemento(Memento memento)// loads  or revert back saved
state
	{
		CList = memento.getState();
	}

}
```

CareTaker

```java
package Memento;
import java.util.ArrayList;
import java.util.List;

public class CareTaker //adds more save states as list is changed
{
	private     <Memento> CarList = new ArrayList<Memento>();
	private     <Memento> CopyCarLList = (    )(((ArrayList)CarList).clone());

	public void add(Memento mem)
	{
		CarList.add(mem); //
		System.out.println("List of Cars:" + CarList + "\n");
	}

	public Memento get(int index)
	{
		return CarList.get(index);
	}
}
```

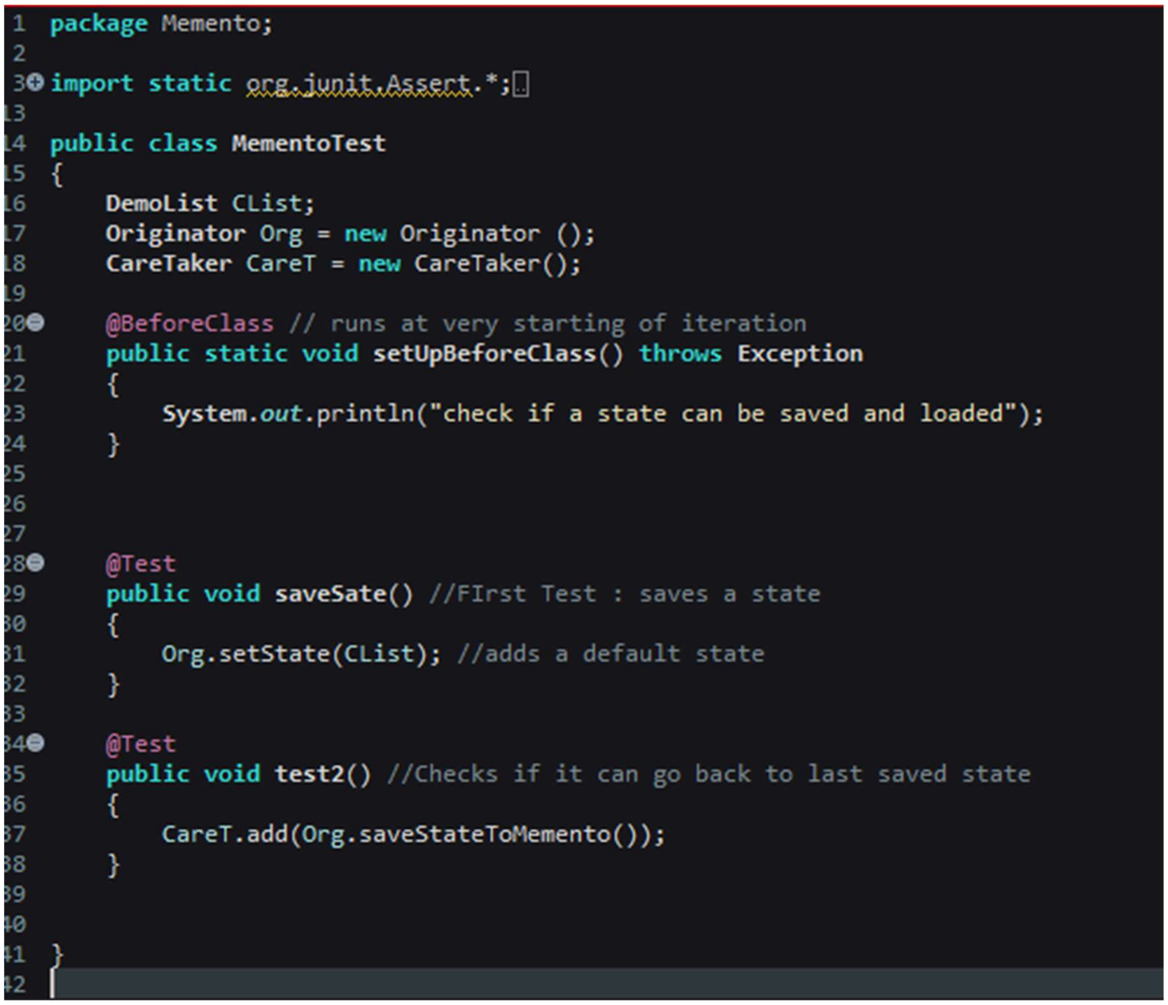
<u>Memento</u>

```java
package Memento;
public class Memento //save state
{
	private DemoList CList;

	public Memento (DemoList CList)
	{
		super();
		this.CList = CList;
	}
	public DemoList getState()
	{
		return CList;
	}
	public void setList(Memento memento)
	{
		CList = memento.getState();
	}

	public String toString()
	{
		return "Add the Cars to the List: " + CList;
	}
}
```

DemoList

```
package Memento;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;

public class DemoList //List to be added from somewhere
{
	private     <Car> Cars = new ArrayList<Car>();

	private String[] RNGMakeList = {"Alpha", "Beta", "Gamma", "Delta", "Epsilon",
"Sigma"};
	private String[] RNGModelList = {"SEDAN", "COUPE", "SPORTS CAR", "STATION
WAGON", "HATCHBACK", "CONVERTIBLE", "SPORT-UTILITY VEHICLE", "PICKUP TRUCK"};
	private short[] RNGYearList = {1989, 1994, 1999, 2005, 2006, 2009, 2011, 2015,
2020, 2022};
	private double[] RNGPriceList = {12300.99, 56300.99, 32500.00, 43600.50,
78900.00, 32200.00};

	public DemoList()
	{
		RNGList();
	}

	public     <Car> getCars()
	{
		return Cars;
	}

	public void setCars(     <Car> cars)
	{
		Cars = cars;
	}

	public DemoCar RNGCar()
	{

		String RNGMake = RNGMakeList[new Random().nextInt(RNGMakeList.length)];
		String RNGModel = RNGModelList[new
Random().nextInt(RNGModelList.length)];
		short RNGYear = RNGYearList[new Random().nextInt(RNGYearList.length)];
		double RNGPrice = RNGPriceList[new
Random().nextInt(RNGPriceList.length)];
		return new DemoCar(RNGMake, RNGModel, RNGYear, RNGPrice);
	}

	private void RNGList()
	{
		int index = 25;

		for(int i = 0; i<index; i++)
		{
			Cars.add(RNGCar());
		}
	}
```

UNIT TESTS



```
package Memento;

import static org.junit.Assert.*;

public class MementoTest
{
    DemoList CList;
    Originator Org = new Originator ();
    CareTaker CareT = new CareTaker();

    @BeforeClass // runs at very starting of iteration
    public static void setUpBeforeClass() throws Exception
    {
        System.out.println("check if a state can be saved and loaded");
    }



    @Test
    public void saveSate() //FIrst Test : saves a state
    {
        Org.setState(CList); //adds a default state
    }

    @Test
    public void test2() //Checks if it can go back to last saved state
    {
        CareT.add(Org.saveStateToMemento());
    }


}
```

COMPONENT TEST

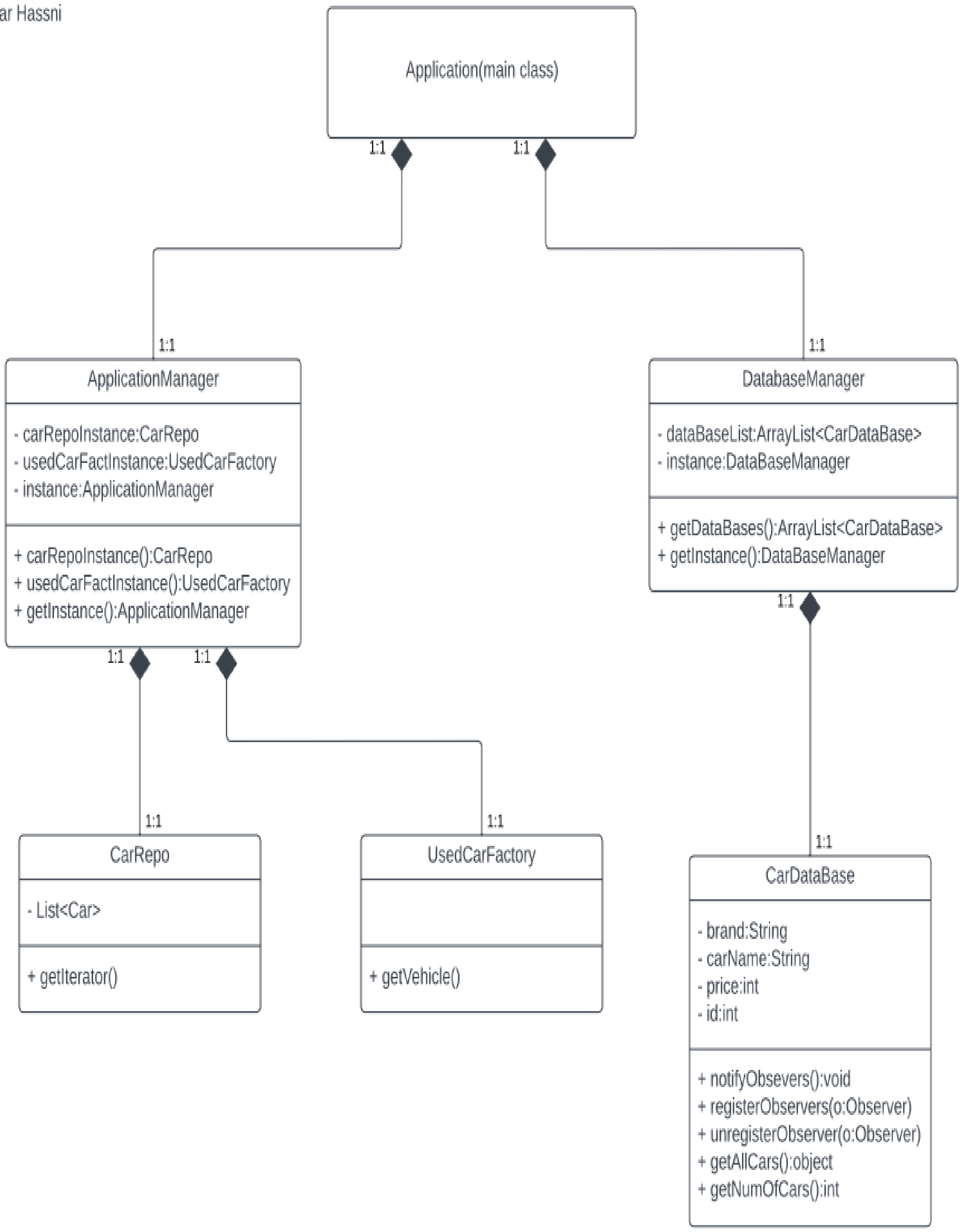
A list of cars exist for the user to buy (gets removed from the list) and sell, (gets added to the

car list). The List is first a default state (untouched) which exists in the Originator, it then is saved as a Memento (Save File), and the CareTaker keeps track of how many different save files are added or removed (keeps track of states). If a car is bought by accident and the list must be return to previous state, the CareTaker takes the saved state in the Memento which has the index no. of state before purchase and restores the list to that state.

Satar Hassni

Singleton Pattern

Singleton pattern by
Satar Hassni


Application(main class)
1:1
1:1
1:1
ApplicationManager
- carRepoInstance:CarRepo
- usedCarFactInstance:UsedCarFactory
- instance:ApplicationManager
+ carRepoInstance():CarRepo
+ usedCarFactInstance():UsedCarFactory
+ getInstance():ApplicationManager
1:1
DatabaseManager
- dataBaseList:ArrayList<CarDataBase>
- instance:DataBaseManager
+ getDataBases():ArrayList<CarDataBase>
+ getInstance():DataBaseManager
1:1
1:1
1:1
1:1
CarRepo
- List<Car>
+ getIterator()
1:1
UsedCarFactory
+ getVehicle()
1:1
1:1
CarDataBase
- brand:String
- carName:String
- price:int
- id:int
+ notifyObsevers():void
+ registerObservers(o:Observer)
+ unregisterObserver(o:Observer)
+ getAllCars():object
+ getNumOfCars():int

```
public class ApplicationManager {

    private CarRepo carRepoInstance;
    private UsedCarFactory usedCarFactory;
    private static ApplicationManager instance;

    public ApplicationManager() {
        this.carRepoInstance = new CarRepo();
        this.usedCarFactory = new UsedCarFactory();
    }

    public static ApplicationManager getInstance() {
        if (instance == null) {
            instance = new ApplicationManager();
        }
        return instance;
    }

    public CarRepo carRepoInstance() {
        return carRepoInstance;
    }

    public UsedCarFactory usedCarFactoryInstance() {
        return usedCarFactory;
    }
}
```

```
import java.util.ArrayList;

public class DataBaseManager {

    private List<CarDataBase> dataBaseList = new ArrayList<CarDataBase>();
    private static DataBaseManager instance;

    public DataBaseManager() {

    }

    public static DataBaseManager getInstance() {
        if (instance == null) {
            instance = new DataBaseManager();
        }
        return instance;
    }

    public List<CarDataBase> getDataBases() {
        return dataBaseList;
    }
}
```

Used Car Dealership

Strategy Pattern by Fahim Ahmed

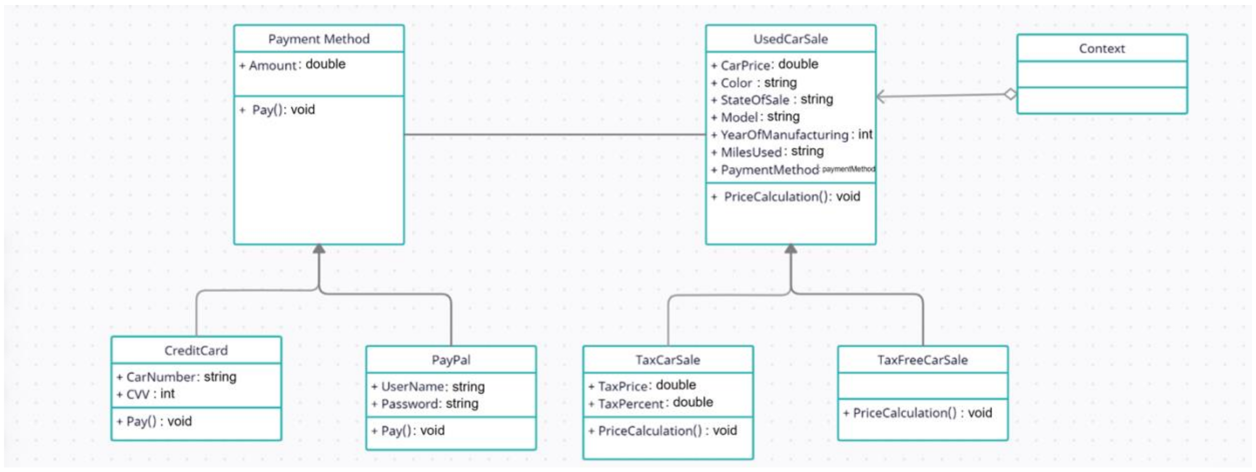
Strategy Pattern: Strategy design pattern is one of the behavioral design patterns. It is used when we have multiple algorithms for a specific task and client decides the actual implementation to be used at runtime. The Strategy Pattern Context class has multiple control strategies provided by the concrete strategy classes, or by the abstract strategy.

Summary:

In this project, Strategy pattern is used in Price calculation of car and also in payment method. We used two different strategies for price calculation. For certain states, buyer does not have to pay tax and vice versa. So, two concrete classes as “TaxCarSale” & “TaxFreeCarSale” is being extended from “UsedCarSale” class. Also in payment method, buyer has two different payment option as credit card & paypal which is also extended from “Payment Method” class.

For testing the code, Junit test and Component test is also implemented in this project.
In component test , positive and negative both testing is added.

UML Diagram:



Source File:

Main:

UsedCarSale.java

```
package project.usedcardealership;

public abstract class UsedCarSale {
   public double CarPrice = 0;
   public String Color = "Red";
   public int YearOfManufacturing = 2020;
   public String Model ="AB231";
   public String StateOfSale ="Ohio";
   public PaymentMethod PaymentType;

   public abstract void PriceCalculation();
}
```

UsedCarDealership.java

```
package project.usedcardealership;

public class UsedCarDealership {

   public static void main(String[] args) {
      //System.out.println("Hello World!");
// Unit Tests

      // Component Test
      RunComponentTest();

      //Negative Test Cases;
      RunNegativeTest();
   }
      public static void RunNegativeTest()
   {
      System.out.println("\nNegative Test\n------------------------");
      // Paypal Object Creation
      Paypal paypal = new Paypal();
      paypal.Amount= 1500;
      paypal.UserName = "UserX";
      paypal.Password = "Passsword1223";

      //CreditCard Object Creation
      CreditCard cc = new CreditCard();
```

```
    cc.Amount = 3000;
    cc.CVV = 233;
    cc.CardNumber="123465609386565";

    CreditCard cc2 = new CreditCard();
    cc2.Amount = 3000;
    cc2.CVV = 23;
    cc2.CardNumber="1234656909386565";
    cc2.pay();

    TaxCarSale car1 = new TaxCarSale();
    car1.CarPrice= 1500;
    car1.PaymentType = paypal;
    car1.TaxPercent= 0;
    car1.PriceCalculation();
    car1.PaymentType.pay();

    TaxFreeCarSale car2 = new TaxFreeCarSale();
    car2.CarPrice=3000;
    car2.PaymentType= cc;
    car2.PriceCalculation();
    car2.PaymentType.pay();
}
public static void RunComponentTest()
{
     System.out.println("\nComponent Test\n------------------------");
    // Paypal Object Creation
    Paypal paypal = new Paypal();
    paypal.Amount= 1500;
    paypal.UserName = "userX@abc.com";
    paypal.Password = "Passsword1223";

    //CreditCard Object Creation
    CreditCard cc = new CreditCard();
    cc.Amount = 3000;
    cc.CVV = 233;
    cc.CardNumber="1234656909386565";

    TaxCarSale car1 = new TaxCarSale();
    car1.CarPrice= 1500;
    car1.PaymentType = paypal;
    car1.TaxPercent= 5;
    car1.PriceCalculation();
    car1.PaymentType.pay();

    TaxFreeCarSale car2 = new TaxFreeCarSale();
```

```
        car2.CarPrice=3000;
        car2.PaymentType= cc;

        car2.PriceCalculation();
        car2.PaymentType.pay();


    }
}
```

PaymentMethod.java

```
package project.usedcardealership;


public abstract class PaymentMethod {
    public double Amount=0;
    public abstract void pay();
}
```

TaxCarSale.java

```
package project.usedcardealership;

public class TaxCarSale extends UsedCarSale{

   public double TaxPercent;
   public double TaxPrice;
   @Override
   public void PriceCalculation() {
      if(TaxPercent == 0)
      {
         System.out.println("Tax cannot be zero percent. Go For Tax Free Sales");
         return;
      }
      TaxPrice = CarPrice*TaxPercent/100;
      CarPrice = TaxPrice+CarPrice;

        System.out.println("Price To Be Paid With Tax: "+ CarPrice);

   }
}
```

TaxFreeCarSale.java

```
package project.usedcardealership;


public class TaxFreeCarSale extends UsedCarSale{

  @Override
  public void PriceCalculation() {
    System.out.println("Price To Be Paid for Tax Free Car Sale: "+ CarPrice);
  }
}
```

CreditCard.java

```
package project.usedcardealership;


public class CreditCard extends PaymentMethod{
   public int CVV;
   public String CardNumber;

      public void pay()
   {
      if(CardNumber.length() == 16){
         if(CVV <99 &&CVV>999){
      System.out.println(Amount+" Paid by Credit Card Number: "+CardNumber);
         }
         else{
            System.out.println("Invalid CVV "+ CVV);
         }
      }
      else{
               System.out.println("Invalid Card Number "+CardNumber);

      }
   }
}
```

Paypal.java

```
package project.usedcardealership;


public class Paypal extends PaymentMethod {
   public String UserName;
   public String Password;
```

```
    public void pay()
    {
        if(UserName.matches("^(.+)@(.+)$"))
        System.out.println(Amount+" Paid by PayPal User: "+UserName);

        else{
                    System.out.println("Username must be a valid email. Invalid Username:
"+UserName);

        }
    }
}
```

UsedCarDealership.java
// **Component Test**

```
package project.usedcardealership;

public class UsedCarDealership {

    public static void main(String[] args) {
        //System.out.println("Hello World!");
// Unit Tests

        // Component Test
        RunComponentTest();

        //Negative Test Cases;
        RunNegativeTest();
    }
        public static void RunNegativeTest()
    {
        System.out.println("\nNegative Test\n------------------------");
        // Paypal Object Creation
        Paypal paypal = new Paypal();
        paypal.Amount= 1500;
        paypal.UserName = "UserX";
        paypal.Password = "Passsword1223";

        //CreditCard Object Creation
        CreditCard cc = new CreditCard();
        cc.Amount = 3000;
        cc.CVV = 233;
        cc.CardNumber="123465609386565";

        CreditCard cc2 = new CreditCard();
        cc2.Amount = 3000;
        cc2.CVV = 23;
        cc2.CardNumber="1234656909386565";
        cc2.pay();

        TaxCarSale car1 = new TaxCarSale();
        car1.CarPrice= 1500;
        car1.PaymentType = paypal;
        car1.TaxPercent= 0;
        car1.PriceCalculation();
        car1.PaymentType.pay();

        TaxFreeCarSale car2 = new TaxFreeCarSale();
        car2.CarPrice=3000;
        car2.PaymentType= cc;
        car2.PriceCalculation();
        car2.PaymentType.pay();
    }
    public static void RunComponentTest()
    {
         System.out.println("\nComponent Test\n------------------------");
        // Paypal Object Creation
        Paypal paypal = new Paypal();
        paypal.Amount= 1500;
        paypal.UserName = "userX@abc.com";
        paypal.Password = "Passsword1223";

        //CreditCard Object Creation
        CreditCard cc = new CreditCard();
        cc.Amount = 3000;
        cc.CVV = 233;
        cc.CardNumber="1234656909386565";

        TaxCarSale car1 = new TaxCarSale();
        car1.CarPrice= 1500;
        car1.PaymentType = paypal;
        car1.TaxPercent= 5;
        car1.PriceCalculation();
        car1.PaymentType.pay();

        TaxFreeCarSale car2 = new TaxFreeCarSale();
        car2.CarPrice=3000;
        car2.PaymentType= cc;
```

```
        car2.PriceCalculation();
        car2.PaymentType.pay();


    }
}
```

Test:

TaxCarSaleTest.java

```
package project.usedcardealership;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author Fahim
 */
public class TaxCarSaleTest {

    public TaxCarSaleTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Before
    public void setUp() {
    }

    @After
    public void tearDown() {
    }
    /**
     * Test of PriceCalculation method, of class TaxCarSale.
     */
    @Test
    public void testPriceCalculation() {
         System.out.println("PriceCalculation");


        TaxCarSale instance = new TaxCarSale();
        instance.TaxPercent= 4;
        instance.CarPrice = 1000;
        instance.Color= "White";
        instance.PriceCalculation();
    }}
```

TaxFreeCarSaleTest.java

```
package project.usedcardealership;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author Fahim
 */
public class TaxFreeCarSaleTest {

    public TaxFreeCarSaleTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Before
    public void setUp() {
    }
    @After
    public void tearDown() {
    }

    /**
     * Test of PriceCalculation method, of class TaxFreeCarSale.
     */
    @Test
    public void testPriceCalculation() {
     System.out.println("PriceCalculation");

        TaxFreeCarSale instance = new TaxFreeCarSale();
        instance.PriceCalculation();
    }

}
```

PaypalTest.java

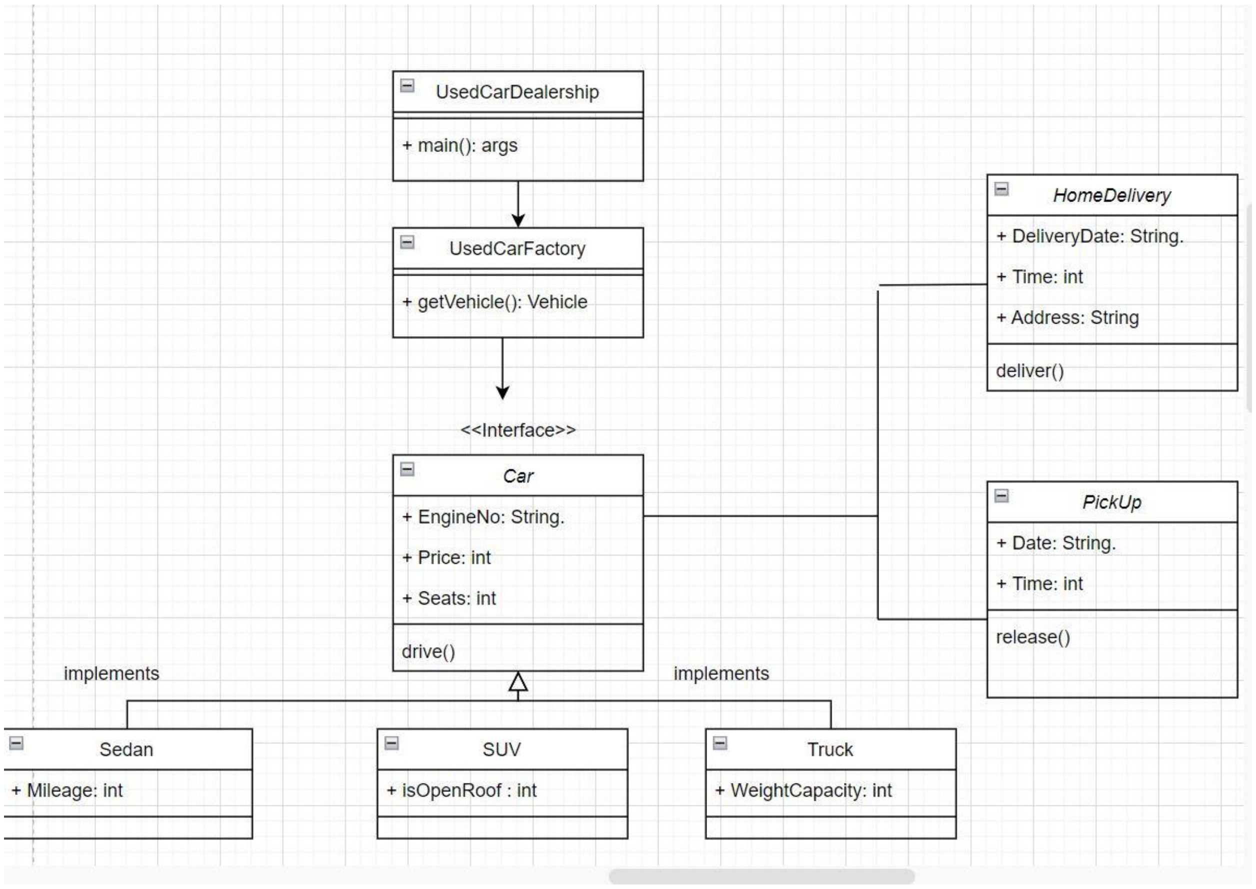
```
package project.usedcardealership;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author Fahim
 */
public class PaypalTest {

    public PaypalTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }
    @Before
    public void setUp() {
    }

    @After
    public void tearDown() {
    }

    /**
     * Test of pay method, of class Paypal.
     */
    @Test
    public void testPay() {
            System.out.println("pay");
        Paypal instance = new Paypal();
        instance.UserName = "user1@abc.com";
        instance.Password = "Pass";
        instance.Amount = 2000;
        instance.pay();
    }

}
```

CreditCardTest.java

```
package project.usedcardealership;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author Fahim
 */
public class CreditCardTest {

    public CreditCardTest() {
    }
    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Before
    public void setUp() {
    }

    @After
    public void tearDown() {
    }

    /**
     * Test of pay method, of class CreditCard.
     */
    @Test
    public void testPay() {
        System.out.println("pay");
        CreditCard instance = new CreditCard();

        instance.CVV = 123;
        instance.CardNumber = "1111222244448888";
        instance.Amount = 2000;
        instance.pay();

        // TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }

}
```

Mohammad Uddin
CSCI 370
Group:4
Used Car Dealership : Factory Pattern

Factory Pattern UML Diagram



Code,

UsedCarFactory.java

```
public class UsedCarFactory {

        public Car getVehicle(String typeOfCar) {
```

```
		if(typeOfCar == null || typeOfCar.isEmpty()) return null;

		switch(typeOfCar) {
		case "Sedan":
			return new Sedan();
		case "SUV":
			return new SUV();
		case "Truck":
			return new Truck();
		default:
			throw new IllegalArgumentException("Unknown type of 
car -> "+typeOfCar);
		}


	}

}
```

UsedCarDealership.java

```
public class UsedCarDealership {

	public static void main(String[] args) {

		UsedCarFactory carFactory = new UsedCarFactory();
		Car newCar=carFactory.getVehicle("SUV");
		newCar.drive();
	}
}
```

Truck.java

```
public class Truck implements Car{

	int weightCapacity=500;

	@Override
	public void drive() {
		System.out.println("Driving a Truck now with 
WeightCapacity: "+weightCapacity);

	}
}
```

SUV.java

```
public class SUV implements Car{
        String isOpenRoof="No";

        @Override
        public void drive() {
                System.out.println("Driving a SUV now with Open Roof 
Status: "+isOpenRoof );

        }



}
```

Sedan.java

```
public class Sedan implements Car{

        int milleage=190;
        @Override
        public void drive() {
                System.out.println("Driving a sedan now with mileage : 
"+milleage+" kms." );

        }
}
```

PickUp.java

```
public class PickUp {

        String date;
        int time;

        public String getDate() {
                return date;
        }
        public void setDate(String date) {
                this.date = date;
        }
        public int getTime() {
                return time;
        }
        public void setTime(int time) {
                this.time = time;
        }
        public PickUp(String date, int time) {
                super();
                this.date = date;
                this.time = time;
        }
}
```

HomeDelivery.java

```
public class HomeDelivery {

        String Address;
        int time;
        String deliveryDate;

        public String getAddress() {
                return Address;
        }

        public void setAddress(String address) {
                Address = address;
        }

        public int getTime() {
                return time;
        }

        public void setTime(int time) {
                this.time = time;
        }

        public String getDeliveryDate() {
                return deliveryDate;
        }

        public void setDeliveryDate(String deliveryDate) {
                this.deliveryDate = deliveryDate;
        }

        public HomeDelivery(String address, int time, String deliveryDate)
{
                super();
                Address = address;
                this.time = time;
                this.deliveryDate = deliveryDate;
        }

        boolean deliverVehicle(Car car,String address) {
                if(address.isEmpty())
                        return false;
                else
                        System.out.println("Delivering car now>.");
                return true;

        }
}
```

Unit Test

CarTest.java

```
import static org.junit.jupiter.api.Assertions.*;

import java.io.ByteArrayOutputStream;
import java.io.PrintStream;

import org.junit.Before;
import org.junit.jupiter.api.Test;

class CarTest {



    final ByteArrayOutputStream outContent = new ByteArrayOutputStream();

        @Before
        public void setUpStreams() {
            System.setOut(new PrintStream(outContent));
        }

        @Test
        public void CarDelivery() {
               HomeDelivery hmd=new HomeDelivery(null, 0, null);
               assertNotNull(hmd.deliverVehicle(null, "South LA, 3445"));

        }

        @Test
        public void TestCarDealership(){

               System.out.println("Test Car object Creation");
               UsedCarFactory obj=new UsedCarFactory();
               assertNotNull(obj.getVehicle("Truck"));

        }

}
```

Car.java

```
public interface Car {
        String engineNo="";
        int price=0;
        int seats=0;


        void drive();

}
```

Component Test

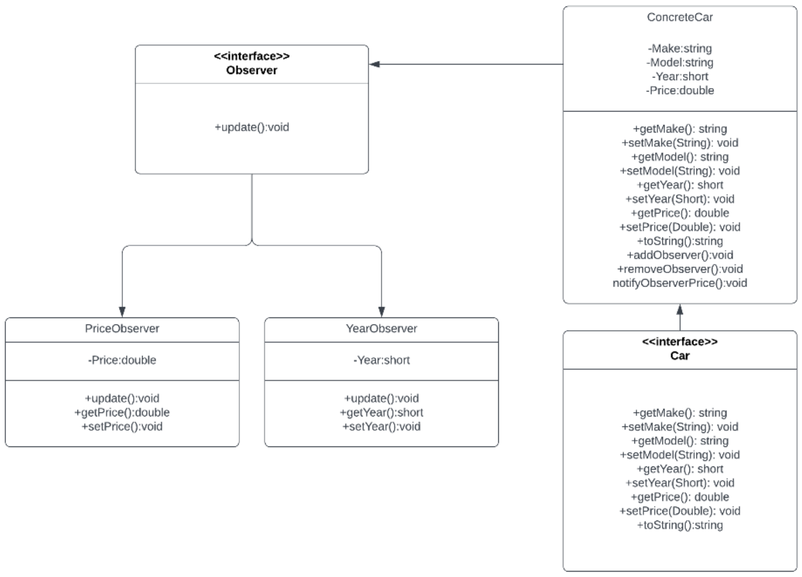
Make a new car instance by giving its type to the factory class which will then implement a switch case deciding which instance to initialize. Then when car is initialized create a new delivery or pick up option by sending in the parameters to the appropriate class constructor which will confirm its order.

Anthony Ferrara

Observer Pattern

Team 4: Used Car Dealership

UML Diagram



Code

Car.java

```
package main;
public interface Car
{
    public String getMake();
    public void setMake(String make);
    public String getModel();
    public void setModel(String model);
    public short getYear();
    public void setYear(short year);
    public double getPrice();
    public void setPrice(double price);
    public String toString();
}
```

ConcreteCar.java

```java
package main;
import java.util.List;
import java.util.ArrayList;

public class ConcreteCar implements Car
{
    private String Make;
    private String Model;
    private short Year;
    private double Price;
    private List<Observer> observers = new ArrayList<>();


    public ConcreteCar(String make, String model, short year, double price)
    {
        Make = make;
        Model = model;
        Year = year;
        Price = price;
    }

    @Override
    public String getMake()
    {
        return this.Make;
    }

    @Override
    public void setMake(String make)
    {
        this.Make = make;

    }

    @Override
    public String getModel()
    {
        return this.Model;
    }

    @Override
    public void setModel(String model)
    {
        this.Model = model;
    }

    @Override
    public short getYear()
    {
        return this.Year;
    }

    @Override
    public void setYear(short year)
    {
        this.Year = year;
    }

    @Override
    public double getPrice()
    {
        return this.Price;
    }

    @Override
    public void setPrice(double price)
    {
        this.Price = price;
    }

    public String toString()
    {
        return "Make: "+Make+"\nModel: "+Model+"\nYear: "+Year+"\nPrice:
"+Price;
    }
    public void addObserver(Observer observer) {
        this.observers.add(observer);
    }

    public void removeObserver(Observer observer) {
        this.observers.remove(observer);
    }

    public void notifyObserversPrice(double price) {
        this.Price = price;
        for (Observer observer : this.observers) {
            observer.update(this.Price);
        }
    }
    public void notifyObserversYear(short year) {
        this.Year = year;
        for (Observer observer : this.observers) {
             observer.update(this.Year);
            }
        }
    }
```

Observer.java

```
package main;

public interface Observer {
    public void update(Object o);
}
```

PriceObserver.java

```
package main;

public class PriceObserver implements Observer{
    private double Price;
    @Override
    public void update(Object Price) {
        this.setPrice((Double) Price);
    }
    public double getPrice() {
        return this.Price;
    }
    public void setPrice(double price)
    {
        this.Price = price;
    }
}
```

YearObserver.java

```
package main;

public class YearObserver implements Observer{
    private short Year;
    @Override
    public void update(Object Year) {
        this.setYear((short) Year);
    }
    public double getYear() {
        return this.Year;
    }
    public void setYear(short year)
    {
        this.Year = year;
    }
}
```

Unit Tests

CarTests.java

```
import main.*;
import org.junit.jupiter.api.*;

public class CarTests {
    @Test
    public void doesReturnPrice(){
        ConcreteCar subject = new ConcreteCar("Ford", "Focus", (short) 2020,
25000);
        PriceObserver observer = new PriceObserver();
        subject.addObserver(observer);
        subject.notifyObserversPrice(23000);
        Assertions.assertEquals(23000, observer.getPrice());
    }
    @Test
    public void doesReturnYear(){
        ConcreteCar subject = new ConcreteCar("Ford", "Focus", (short) 2020,
25000);
        YearObserver observer = new YearObserver();
        subject.addObserver(observer);
        subject.notifyObserversYear((short) 2021);
        Assertions.assertEquals(2021, observer.getYear());

    }
}
```

Component Test

Create two observer classes, one that observes the car’s year of production, and one that observes the car’s price. When these values change, they are passed to the observers via the ConcreteCar class. Add an instance of either observer to the array list located within the ConcreteCar class. Then, use the getter methods located within each observer class, (priceObserver and yearObserver) to retrieve the information.