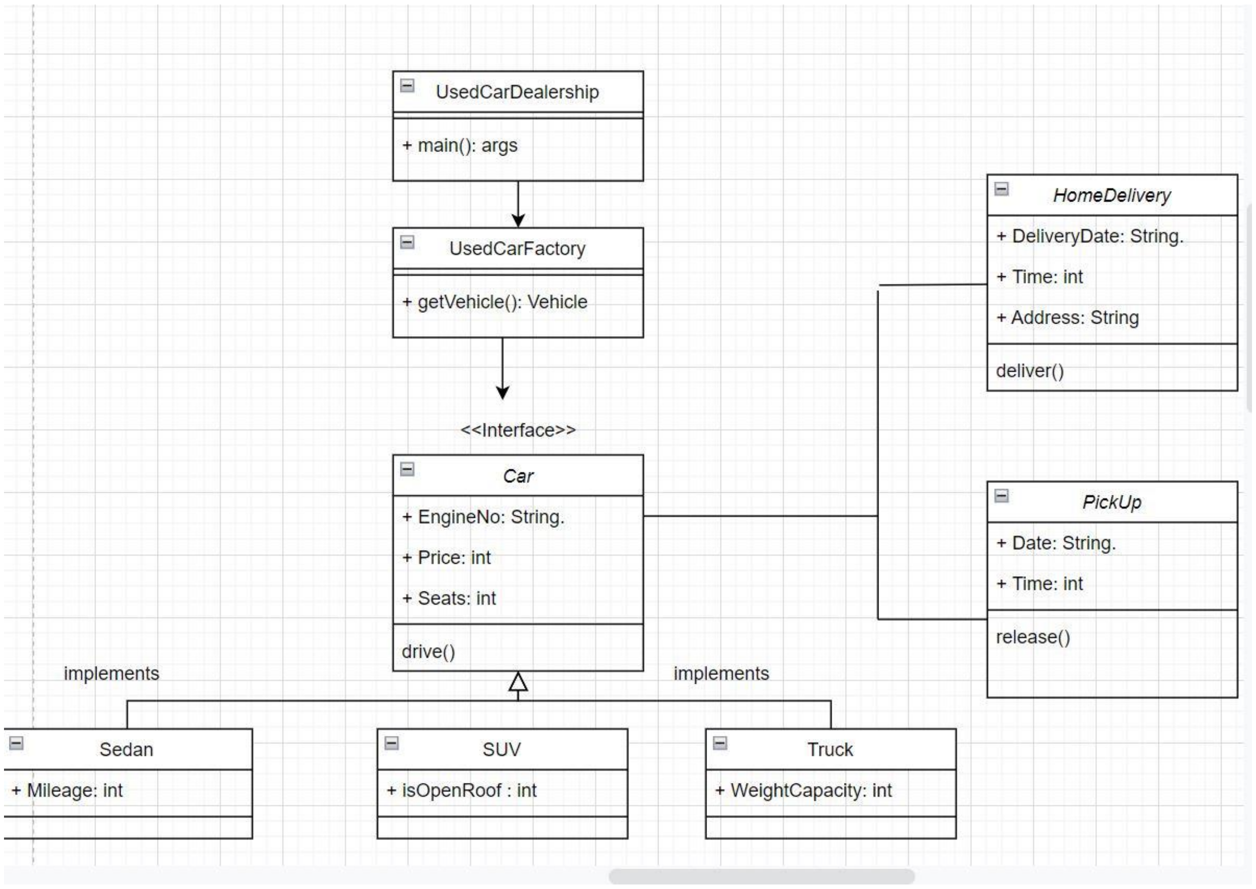
Mohammad Uddin
CSCI 370
Group:4

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.