Probleemstelling Portfolio Opdracht

Dealership Fortiono

Mehmet Emin Ozturk

20129947 (@student.hhs.nl)

DH-SE-3

Autodealer Fortiono is een gerespecteerde autodealer die al jarenlang een probleem heeft met het onderscheiden van allerlei voertuigen binnenin zijn systeem.

Op dit moment wordt alle data bijgehouden doormiddel van een Excel bestand, dit gaat regelmatig fout omdat de medewerkers of verkeerde data gebruiken of soms helemaal geen data.

Doormiddel van “Fortiono Administratie” wordt er gebruik gemaakt van een dynamische applicatie die ervoor zorgt dat de correcte data altijd wordt getoond op het juiste moment. Zo kan een medewerker sneller inzien welke dealer welke voertuig heeft verkocht. Ook wordt het gemakkelijker om alle voertuigen te inventariseren.

De volgende functionaliteiten zitten in de applicatie:

* Je kan vanuit een dealership een dealer toevoegen.
* Je kan nieuwe auto’s aanmaken.
* Je kan nieuwe scooters aanmaken
* Je kan auto’s inzien
* Je kan scooters inzien
* Testcases moeten werken.

Inhoud

[UML 3](#_Toc67746296)

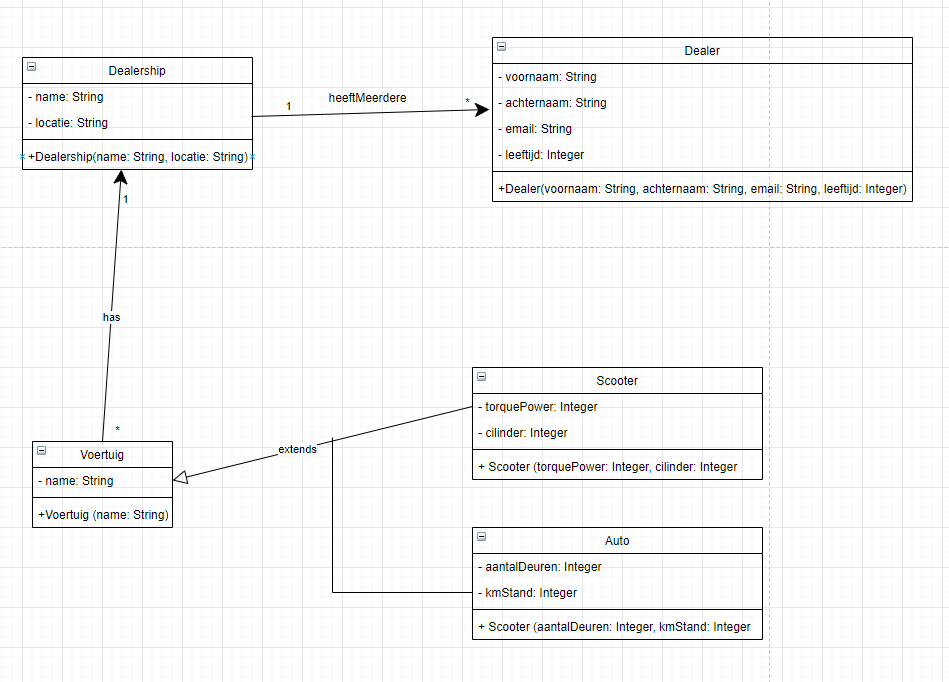
[Code 4](#_Toc67746297)

[Test-cases / J-Unit testing 9](#_Toc67746298)

[Resultaat J-unit testing 9](#_Toc67746299)

[Beoordelingsformulier 10](#_Toc67746300)

# UML



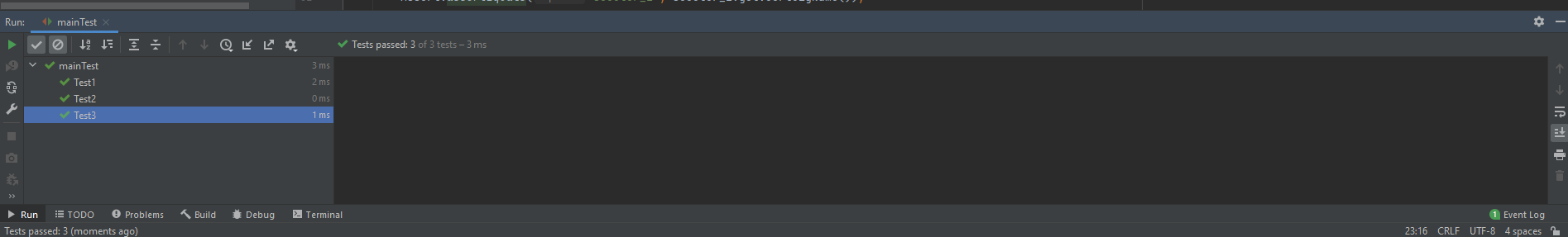
# Code

import javax.xml.namespace.QName;  
import java.util.ArrayList;  
  
public class Main {  
  
 public static void main(String[] args) {  
  
 // Auto voor test  
 Auto auto1 = new Auto("BMW", 120000, 2);  
 auto1.Opslag();  
  
 System.*out*.println("-------------------------------------------");  
  
 Auto auto2 = new Auto("Audi", 89000, 4);  
 auto2.Opslag();  
  
 System.*out*.println("-------------------------------------------");  
  
 // Scooter voor test  
 Scooter scooter1 = new Scooter("Vespa", 50, 200);  
 scooter1.Opslag();  
  
 System.*out*.println("-------------------------------------------");  
  
 //Dealership locatie uitprinten  
 Dealership dlship1 = new Dealership("Fontiona", "Kosovo");  
 dlship1.dLocatie();  
  
 System.*out*.println("-------------------------------------------");  
  
 //Dealer uitprinten  
 Dealer dealer1 = new Dealer("Mehmet", "Ozturk", "20129947@student.hhs.nl", 20);  
 dealer1.dDealer();  
  
 System.*out*.println("----------------------TESTING TESTING TESTING---------------------");  
  
 dlship1.addDealer(dealer1);  
 dlship1.getDealers();  
  
 Scooter scooby = new Scooter("Scooby",20,10);  
  
 scooby.startEngine();  
  
 System.*out*.println("-------------------------------------------");  
  
 }  
}  
  
class Dealer {  
 private String voornaam;  
 private String achternaam;  
 private String email;  
 private Integer leeftijd;  
  
 public Dealer(String voornaam, String achternaam, String email, Integer leeftijd) {  
 this.voornaam = voornaam;  
 this.achternaam = achternaam;  
 this.email = email;  
 this.leeftijd = leeftijd;  
 }  
  
 public void dDealer(){  
 System.*out*.println("Dealer heet: " + this.getVoornaam() + " " + this.getAchternaam());  
 System.*out*.println("De/het/it email van de/het/it dealer is: " + this.getEmail());  
 }  
  
 public String getVoornaam() {  
 return voornaam;  
 }  
  
 public void setVoornaam(String voornaam) {  
 this.voornaam = voornaam;  
 }  
  
 public String getAchternaam() {  
 return achternaam;  
 }  
  
 public void setAchternaam(String achternaam) {  
 this.achternaam = achternaam;  
 }  
  
 public String getEmail() {  
 return email;  
 }  
  
 public void setEmail(String email) {  
 this.email = email;  
 }  
  
 public Integer getLeeftijd() {  
 return leeftijd;  
 }  
  
 public void setLeeftijd(Integer leeftijd) {  
 this.leeftijd = leeftijd;  
 }  
}  
  
class Dealership {  
 private String naam;  
 private String locatie;  
 private ArrayList<Dealer> hasA = new ArrayList<>();  
  
 public Dealership(String naam, String locatie) {  
 this.naam = naam;  
 this.locatie = locatie;  
 }  
  
 public void addDealer(Dealer dealer) {  
 hasA.add(dealer);  
 }  
  
 public void getDealers() {  
 for (Dealer m : hasA) {  
 System.*out*.println(m.getVoornaam() + " " + m.getAchternaam());  
 }  
 }  
  
 public ArrayList<Dealer> getHasA() {  
 return hasA;  
 }  
  
 public void dLocatie(){  
 System.*out*.println("Locatie is: " + this.getLocatie());  
 System.*out*.println("En de naam van deze franchise is " + this.getNaam());  
 }  
  
 public String getNaam() {  
 return naam;  
 }  
  
 public void setNaam(String naam) {  
 this.naam = naam;  
 }  
  
 public String getLocatie() {  
 return locatie;  
 }  
  
 public void setLocatie(String locatie) {  
 this.locatie = locatie;  
 }  
  
}  
  
abstract class Voertuig{  
 private String name;  
 private Dealership has;  
  
 public abstract void Opslag();  
  
 public Voertuig(String name) {  
 this.name = name;  
 }  
  
 public String getVoertuigName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public Dealership getHas() {  
 return has;  
 }  
  
 public void setHas(Dealership has) {  
 this.has = has;  
 }  
  
 public void startEngine() {  
 System.*out*.println("starting engine....");  
 }  
}  
  
class Scooter extends Voertuig {  
 private Integer torquePower;  
 private Integer cilinder;  
  
 public Scooter(String voertuigName, Integer torquePower, Integer cilinder) {  
 super(voertuigName);  
 this.cilinder = cilinder;  
 this.torquePower = torquePower;  
 }  
  
 @Override  
 public void startEngine() {  
 System.*out*.println(this.getVoertuigName() + " starts engine");  
 }  
  
 public Integer getTorquePower() {  
 return torquePower;  
 }  
  
 public void setTorquePower(Integer torquePower) {  
 this.torquePower = torquePower;  
 }  
  
 public Integer getCilinder() {  
 return cilinder;  
 }  
  
 public void setCilinder(Integer cilinder) {  
 this.cilinder = cilinder;  
 }  
  
 @Override  
 public void Opslag() {  
 System.*out*.println("Scooter heeft merk "+ this.getVoertuigName() + ".");  
 System.*out*.println("Heeft zoveel " + this.getCilinder() + " cilinders,");  
 System.*out*.println("en heeft zoveel " + this.getTorquePower() + " torque.");  
 }  
  
}  
  
class Auto extends Voertuig {  
 private Integer kmStand;  
 private Integer aantalDeuren;  
  
 public Auto(String name, Integer kmStand, Integer aantalDeuren) {  
 super(name);  
 this.aantalDeuren = aantalDeuren;  
 this.kmStand = kmStand;  
 }  
  
 @Override  
 public void Opslag() {  
 System.*out*.println("Auto heeft merk "+ this.getVoertuigName());  
 System.*out*.println("Heeft als kilometerstand: " + this.getKmStand());  
 System.*out*.println("En heeft zoveel deuren: " + this.getAantalDeuren());  
 }  
  
 public Integer getKmStand() {  
 return kmStand;  
 }  
  
 public void setKmStand(Integer kmStand) {  
 this.kmStand = kmStand;  
 }  
  
 public Integer getAantalDeuren() {  
 return aantalDeuren;  
 }  
  
 public void setAantalDeuren(Integer aantalDeuren) {  
 this.aantalDeuren = aantalDeuren;  
 }  
  
}

# Test-cases / J-Unit testing

import org.junit.Assert;  
import org.junit.Test;  
  
public class mainTest {  
  
 // Test of je dealers kan toevoegen aan een dealership.  
 @Test  
 public void Test1() {  
 Dealer dealer\_1 = new Dealer("test","test","dd@test.nl",18);  
 Dealership dealership\_1 = new Dealership("dealership\_1","locatie\_1");  
 dealership\_1.addDealer(dealer\_1);  
  
 // Dealer wordt toegevoegd, waardoor dealership 1 dealer heeft namelijk .get(0) en hierop wordt de test uitegevoerd.  
 Assert.*assertEquals*(dealer\_1, dealership\_1.getHasA().get(0));  
 }  
  
 // Test of je een dealer kan specificeren.  
 @Test  
 public void Test2() {  
 Dealer dealer\_1 = new Dealer("test","test","dd@test.nl",18);  
 Dealer dealer\_2 = new Dealer("test2","test2","dd@test.nl2",19);  
  
 Assert.*assertEquals*("test", dealer\_1.getVoornaam());  
 Assert.*assertEquals*("test2", dealer\_2.getVoornaam());  
 }  
  
 // Test of method overriding van extended abstract class werkt.  
 @Test  
 public void Test3() {  
 // scooter de voertuigNaam komt vanuit abstract class Voertuig en hierbij testen we of het werkt.  
 Scooter scooter\_1 = new Scooter("scooter\_1",20,10);  
 Assert.*assertEquals*("scooter\_1", scooter\_1.getVoertuigName());  
  
 }  
}

# Resultaat J-unit testing



# Beoordelingsformulier

**Beoordelingscriteria en instructies**

1. De (gehele) portfolio opdracht wordt met een voldoende afgesloten bij 55 of meer punten. Er wordt geen goed toegekend (het is of voldoende of onvoldoende).

* Aantal behaalde punten >= 55 = voldoende
* Aantal behaalde punten < 55 = onvoldoende

1. Het onderdeel wordt in zijn geheel als goed of fout beoordeeld. Het is dus NIET mogelijk een deel van de punten te krijgen voor eenn onderdeel/ criteria.
2. De student laat zijn werk controleren tijdens de individuele assessment.
3. De student mag voorbeelden NIET overnemen van het lesmateriaal (dit wordt gezien als fraude).
4. Als code wel werkt maar op essentiële punten onder de maat is (ontbreken van constanten, gebruiken van onjuiste variabele- en methodenamen), worden geen punten toegekend. De examinator maakt dan kenbaar waarom het onvoldoende is.

|  |  |  |  |
| --- | --- | --- | --- |
| **Criteria #** | **Criteria / indicator behaald als** | **Punten** | **Onvoldoende / Voldoende** |
|  | Je een beknopte probleemstelling (max. 200 woorden) hebt geformuleerd. | 5 | Voldoende |
|  | Je een UML klassendiagram hebt bijgevoegd met minimaal 5 klassen. | 25 | Voldoende |
|  | Je overerving en polymorphism hebt toegepast. | 10 | Voldoende |
|  | Je gebruik gemaakt hebt van abstract classes en/of interfaces. | 10 | Voldoende |
|  | Je method overriding hebt toegepast. | 10 | Voldoende |
|  | Je minimaal 3 zinvolle JUnit testen hebt. | 20 | Voldoende |
|  | Je aan kunt tonen in IntelliJ dat het programma werkt (zowel het programma als de JUnit testen). | 5 | Voldoende |
|  | Je composiet en/of aggregaat relatie hebt verwerkt in de opdracht. | 15 | Onvoldoende |
| **Totaal** | | **100** | **85** |