1 Navigationssystem

Für die Firma NavTech soll eine Navigator-App implementiert werden. Die Applikation soll in der Lage sein die benötigte Zeit einer Route von Orten zu berechnen. Je nachdem welches Reisemittel gewählt wird, berechnet sich die benötigte Zeit unterschiedlich. Um zukünftig weitere Reiseoptionen zur Auswahl stellen zu können, soll das **Strategie-Muster** verwendet werden. Die Route soll durch ein Array von *Ort*-Objekten der Strategie übergeben werden. Die Route wird in aufsteigender Reihenfolge des Arrays abgearbeitet. Ein Ort muss einen *Namen*, *X*- und *Y*-Koordinaten besitzen. Dabei sind die Koordinaten als 2D-GPS Position zu verstehen. Für jedes der drei Attribute müssen Getter- und Setter-Methoden angegeben werden.

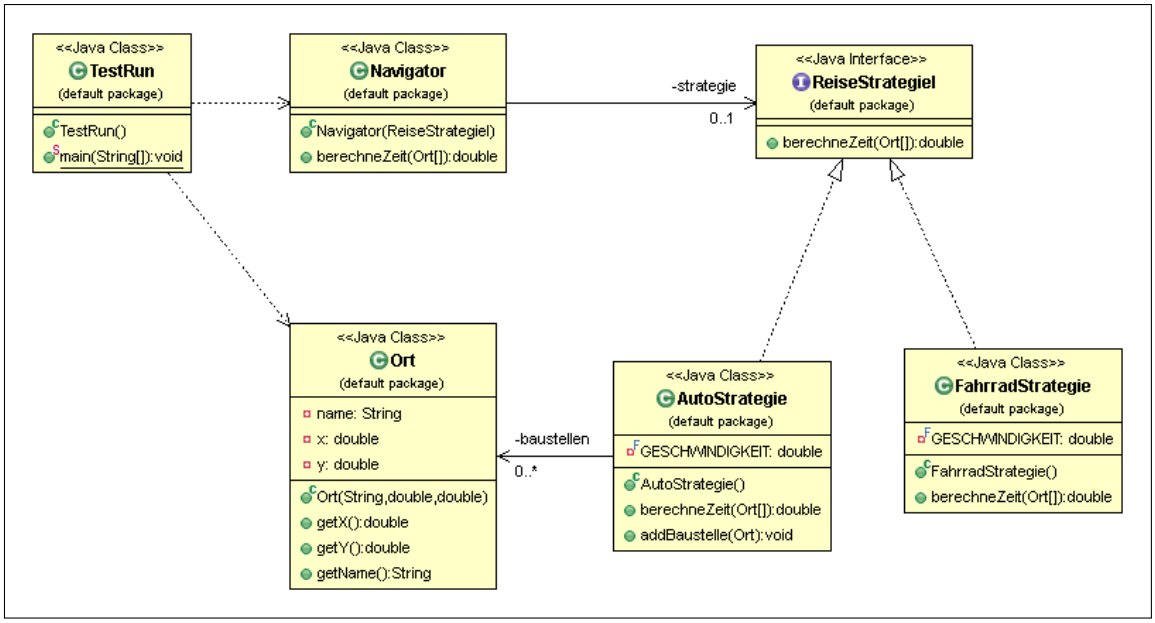
Eine *ReiseStrategie* besitzt die Methode *berechneZeit(route : Ort[])*. Die Methode nimmt ein Array von Orten entgegen und gibt die Zeit als *double*-Wert zurück. Die Strategie *FahrradStrategie* implementiert das *ReiseStrategie* Interface. Zusätzlich besitzt die Klasse ein konstantes Attribut namens *GESCHWINDIGKEIT*, welches auf den Wert 10.0 (für 10 km/h) gesetzt wird. Zur Berechnung der Zeit soll die Route durchlaufen werden und Paarweise die Strecken als Euklidische Distanz aufsummiert werden. Die benötigte Zeit wird berechnet indem die zurückgelegte Strecke durch die Geschwindigkeit geteilt wird.

Die Strategie *AutoStrategie* implementiert ebenfalls das *ReiseStrategie* Interface. Dabei trifft alles was zu der FahrradStrategie erläutert wurde ebenfalls auf die AutoStrategie zu. Die Strategie wird jedoch um das Attribut *baustellen* ergänzt und die Geschwindigkeit wird auf 30.5 gesetzt. Beim *baustellen*-Attribut handelt es sich um eine Liste von *Ort*-Objekten. Ist ein Ort aus der Route in der Liste der Baustellen enthalten, wird die benötigte Zeit um 15 Minuten verlängert. Die AutoStrategie besitzt zudem eine Methode namens *addBaustelle(ort : Ort)*, welche einen Ort zu der Liste der Baustellen hinzufügt.

Aufgaben

1. Zeichnen Sie das entsprechende UML-Diagramm zu der angegebenen Beschreibung.

Lösung:



2. Implementieren Sie das UML-Diagramm aus Aufgabe 1.

Lösung:

Ort.java

```
public class Ort {

    private String name;
    private double x;
    private double y;

    public Ort(String name, double x, double y) {
        this.name = name;
        this.x = x;
        this.y = y;
    }

    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }

    public String getName() {
        return name;
    }
}
```

ReiseStrategieI.java

```
public interface ReiseStrategieI {

    public double berechneZeit(Ort[] route);

}
```

FahrradStrategie.java

```
public class FahrradStrategie implements ReiseStrategieI {

    private final double GESCHWINDIGKEIT = 10.0;

    @Override
    public double berechneZeit(Ort[] route) {

        double strecke = 0.0;
        Ort vorgeanger = null;
        for(Ort o : route) {
            if(vorgeanger != null) {

                strecke = Math.sqrt( Math.pow(vorgeanger.getX() -
                ↪  o.getX(), 2) +  Math.pow(vorgeanger.getY() -
                ↪  o.getY(), 2));

            }
            vorgeanger = o;
        }

        return (strecke / GESCHWINDIGKEIT)*60;
    }
}
```

AutoStrategie.java

```
import java.util.ArrayList;

public class AutoStrategie implements ReiseStrategieI {

    private final double GESCHWINDIGKEIT = 30.5; // km pro h
    private ArrayList<Ort> baustellen = new ArrayList<>();

    @Override
    public double berechneZeit(Ort[] route) {
```

```
        double strecke = 0.0;
        double stauZeit = 0.0;
        Ort vorgeanger = null;
        for(Ort o : route) {
            if(vorgeanger != null) {

                strecke = Math.sqrt( Math.pow(vorgeanger.getX() -
                ↪  o.getX(), 2) +  Math.pow(vorgeanger.getY() -
                ↪  o.getY(), 2));

                if(baustellen.contains(o)) {
                    stauZeit += 15;
                }

            }
            vorgeanger = o;
        }

        return (strecke / GESCHWINDIGKEIT)*60 + stauZeit;
    }

    public void addBaustelle(Ort o) {
        baustellen.add(o);
    }
}
```

Navigator.java

```
public class Navigator {

    private ReiseStrategieI strategie;

    public Navigator(ReiseStrategieI strategie) {
        this.strategie = strategie;
    }

    public double berechneZeit(Ort[] route) {
        return strategie.berechneZeit(route);
    }
}
```

3. Testen sie ihre Implementierung indem sie folgende Route verwenden: Bochum (12, 15) → Wattenscheid (25, 18) → Essen (30, 20) → Mülheim (35, 25). Testen Sie mit

dem Auto und mit dem Fahrrad.

Lösung:

TestRun.java

```
public class TestRun {

    public static void main(String[] args) {

        Ort[] route = new Ort[4];
        route[0] = new Ort("Bochum", 12.0, 15.0);
        route[1] = new Ort("Wattenscheid", 25.0, 18.0);
        route[2] = new Ort("Essen", 30.0, 20.0);
        route[3] = new Ort("Mülheim", 35.0, 25.0);

        //AutoStrategie strategie = new AutoStrategie();
        //strategie.addBaustelle(route[2]);

        FahrradStrategie strategie = new FahrradStrategie();

        Navigator nav = new Navigator(strategie);

        double zeit = nav.berechneZeit(route);
        System.out.println(zeit);
    }

}
```

4. Was ist zu beachten beim Aufruf von *addBaustelle* in der *AutoStrategie*?

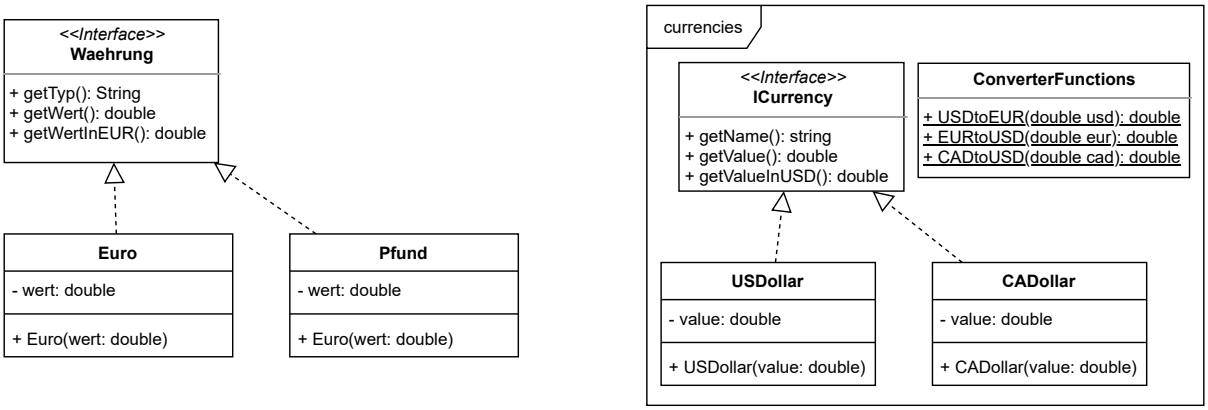
Lösung:

Überschreibt die Klasse Ort nicht die `equals()`-Funktion, dann muss das übergebene Ort-Objekt die selbe Adresse haben wie ein Objekt aus der Route.

2 Währungs-Umrechnung

Eine Bank verwendet für das Verwalten von Geld-Transfers eine Umrechnungssoftware. Die Software unterstützt bereits die Europäischen Währungen Euro und Pfund. Alle Währungen werden als Interface `Waehrung` im Softwaresystem angelegt. In Zukunft sollen mit der Software auch andere Währungen interpretiert werden. Eine externe amerikanische Firma hat eine Bibliothek für Dollar mit dem Interface `Currency` eingeführt. Diese Bibliothek beinhaltet auch eine Hilfsklasse zum Umrechnen von Dollar in andere Währungen. Nun soll diese Bibliothek in die Software eingebunden werden. Folgende Klassen sind bereits gegeben:

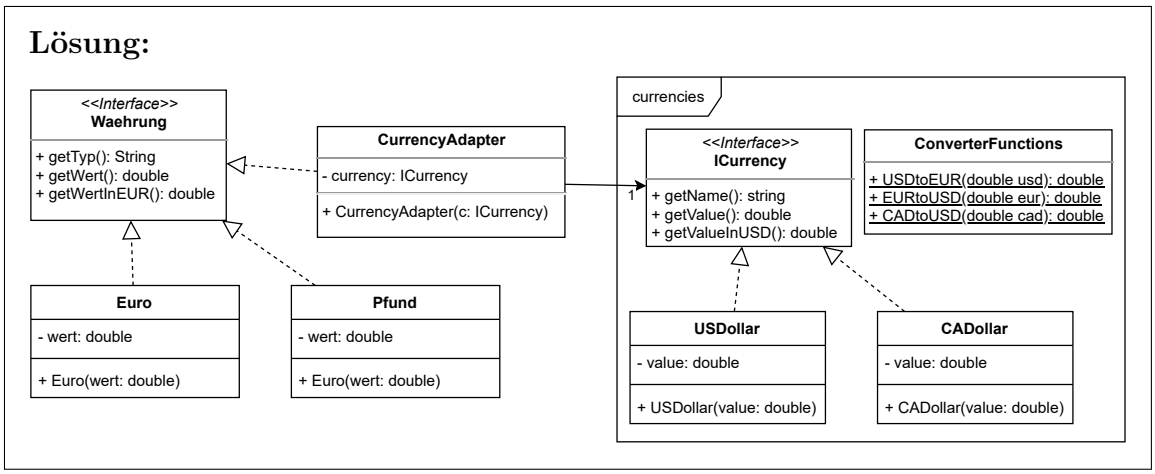
static function(): 函数名下面有下划线



Aufgaben

1. Erweitern Sie das UML-Diagramm um ein Adaptermuster, welches die externen Klassen `USDollar` und `CADollar` unter dem Interface `Waehrung` verfügbar machen.

Lösung:



2. Implementieren Sie alle in Aufgabenteil 1) neu hinzugefügten Klassen.

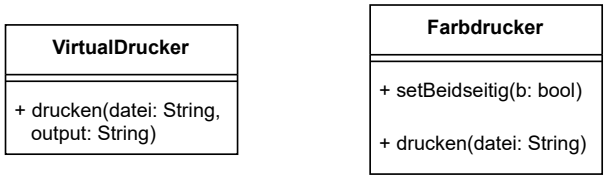
Lösung:

```
public class CurrencyAdapter implements Waehrung {
    private ICurrency currency;

    public CurrencyAdapter(ICurrency currency) {
        this.currency = currency;
    }

    public String getTyp() {
        return currency.getName();
    }

    public double getWert() {
        return currency.getValue();
    }

    public double getWertInEUR() {
        return
        ↪  ConverterFunctions.USDtoEUR(currency.getValueInUSD());
    }
}
```

3 Druckverwaltung

In der Firma ComIT arbeiten Programmierer, Sekretäre, Auszubildende und der Vorstand. Für den Austausch von Dokumenten stehen ein Farbdrucker und ein spezieller Virtual-Drucker zur Verfügung. Ein firmeneigenes Druckzentrum ist über das Netzwerk verbunden, sodass jeder Mitarbeiter ein Befehl zum Drucken senden kann.

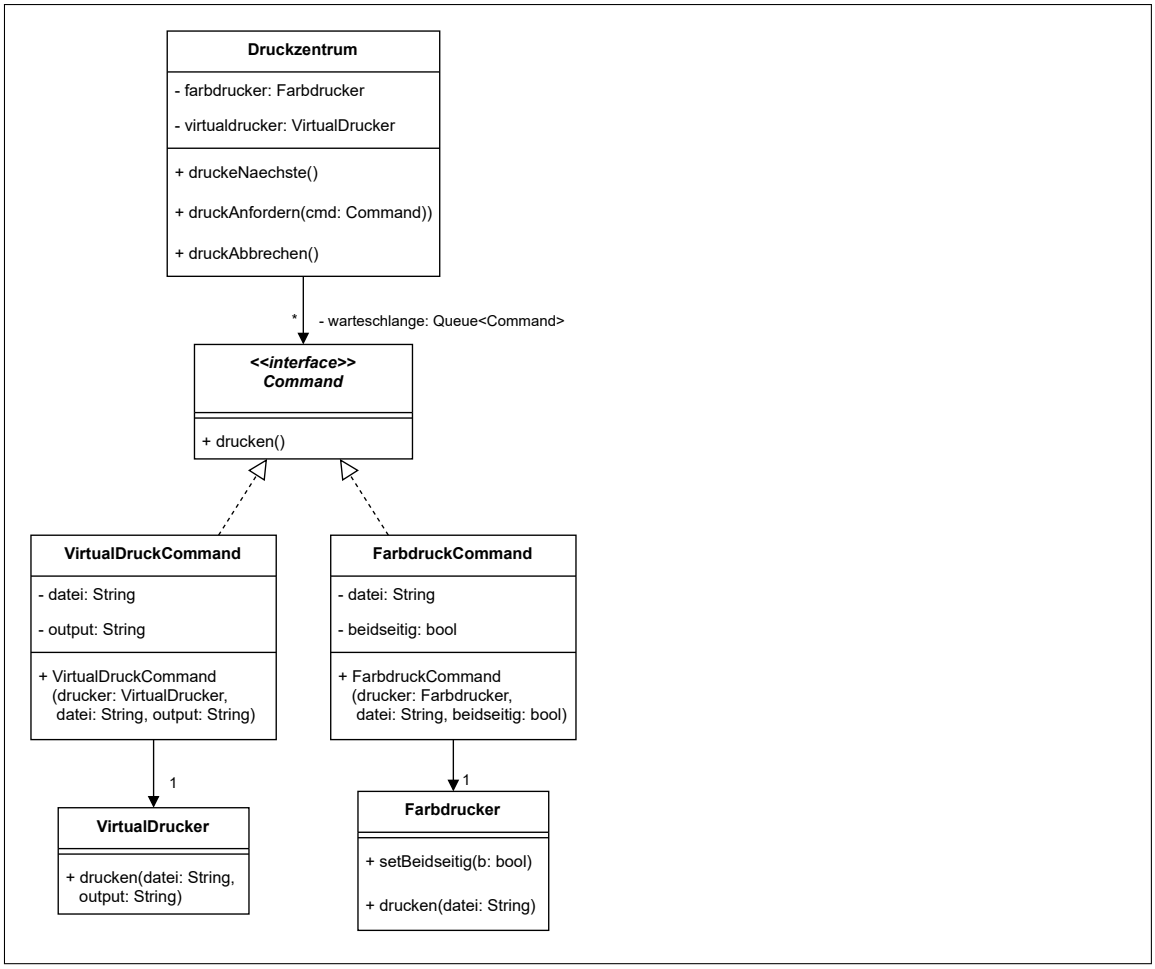
Der Druckvorgang wird vom Zentrum selbst ausgelöst. Das zu druckende Dokument wird als Text (String-Typ) einem Druckbefehl zugeordnet. Der Farbdrucker kann nach Wunsch beidseitig drucken. Der Virtual-Drucker erstellt eine PDF-Datei unter einem angegebenen Pfad. Dokumente werden nicht sofort gedruckt, sondern landen in einer gemeinsamen Warteschlange im Druckzentrum (`druckAnfordern()`-Methode), sodass der letzte Druckvorgang noch abgebrochen werden kann (`druckAbbrechen()`-Methode). Ein Aufruf von `druckNaechste()` druckt das vorderste Dokument in der Warteschlange.



Aufgaben

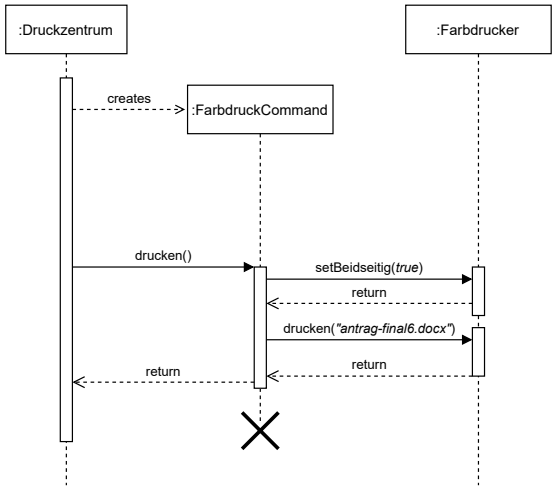
1. Erstellen Sie anhand der gegebenen Informationen ein UML-Klassendiagramm, welches das Druckzentrum und die Drucker modelliert. Verwenden Sie dabei das Command-Entwurfsmuster.

Lösung:



2. Ein Mitarbeiter möchte ein Dokument “antrag-final6.docx” beidseitig per Farbdrucker drucken. Zeigen Sie mithilfe eines Sequenzdiagramms einen beispielhaften Systemablauf.

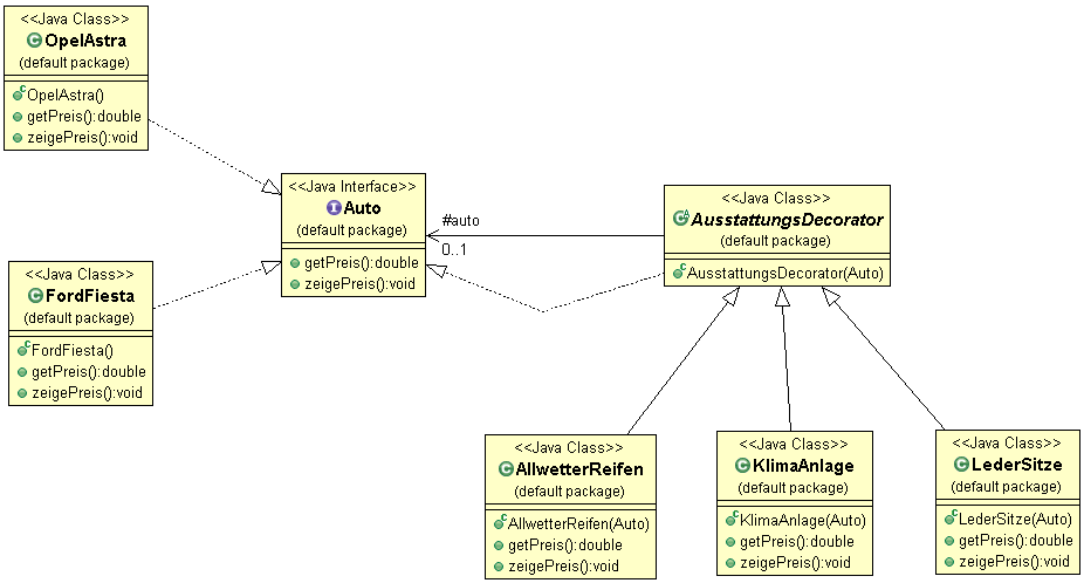
Lösung:



4 Autoaustatter

Für die Automobilindustrie sind Anpassungen der Ausstattung eines Fahrzeugs nach Kundenwünschen wichtig. Angenommen ein Autohaus verkauft zwei unterschiedliche Wagentypen. Ein Opel Astra hat einen Grundpreis von 7500 Euro und ein Ford Fiesta einen Grundpreis von 3500 Euro.

Um den Preis für die zusätzlichen Ausstattungen zu verändern soll das Decorator-Muster verwendet werden. Es werden drei Ausstattungen angeboten. Eine Klimaanlage kostet 550 Euro. Für 725 Euro bekommt man Allwetterreifen dazu. Ledersitze werden für 144,99 Euro gehandelt.



Aufgaben

1. Implementieren Sie das angegebene UML-Diagramm. Berücksichtigen Sie dabei die Anforderungen an die Ausstattung.

Lösung:

Auto.java

```
public interface Auto {

    public double getPreis();
    public void zeigePreis();

}
```

FordFiesta.java

```java
public class FordFiesta implements Auto{

    @Override
    public double getPreis() {
        return 7200;
    }

    @Override
    public void zeigePreis() {
        System.out.println("Grundpreis f r " +
        ↪  this.getClass().getSimpleName() + ": " +
        ↪  this.getPreis() + " Euro");
    }

}
```

OpelAstra.java

```java
public class OpelAstra implements Auto{

    @Override
    public double getPreis() {
        return 3500;
    }

    @Override
    public void zeigePreis() {
        System.out.println("Grundpreis für " +
        ↪  this.getClass().getSimpleName() + ": " +
        ↪  this.getPreis() + " Euro");
    }

}
```

AusstattungsDecorator.java

```java
public abstract class AusstattungsDecorator implements Auto{

    protected Auto auto = null;

    public AusstattungsDecorator(Auto auto) {
        this.auto = auto;
    }

}
```

AllwetterReifen.java

```
public class AllwetterReifen extends AusstattungsDecorator{

    public AllwetterReifen(Auto auto) {
        super(auto);
    }

    @Override
    public double getPreis() {
        return this.auto.getPreis() + 750;
    }

    @Override
    public void zeigePreis() {
        auto.zeigePreis();
        System.out.println("Dazu allwetter Reifen: " +
        ↪  this.getPreis() + " Euro");
    }



}
```

KlimaAnlage.java

```
public class KlimaAnlage extends AusstattungsDecorator{

    public KlimaAnlage(Auto auto) {
        super(auto);
    }

    @Override
    public double getPreis() {
        return this.auto.getPreis() + 550;
    }

    @Override
    public void zeigePreis() {
        auto.zeigePreis();
        System.out.println("Dazu Klima Anlage: " + this.getPreis()
        ↪  + " Euro");
    }
```

```
}
```

LederSitze.java

```
public class LederSitze extends AusstattungsDecorator{

    public LederSitze(Auto auto) {
        super(auto);
    }

    @Override
    public double getPreis() {
        return this.auto.getPreis() + 125.5;
    }

    @Override
    public void zeigePreis() {
        auto.zeigePreis();
        System.out.println("Dazu Ledersitze: " + this.getPreis() +
        ↪  " Euro");
    }



}
```

2. Testen Sie ihre Implementierung indem Sie sich den Preis eines Opel Astra mit Allwetterreifen und Klimaanlage ausgeben lassen.

Lösung:

TestRun.java

```
public class TestRun {

    public static void main(String[] args) {

        Auto ford = new FordFiesta();
        Auto opel = new OpelAstra();
```

```
        KlimaAnlage anlage = new KlimaAnlage(ford);
        LederSitze sitze = new LederSitze(anlage);
        Navigator navi = new Navigator(sitze);
        navi.zeigePreis();

        KlimaAnlage anlage2 = new KlimaAnlage(opel);
        AllwetterReifen reifen = new AllwetterReifen(anlage2);
        reifen.zeigePreis();


    }

}
```

5 Buchverwaltung

Für eine Bücherei soll eine Verwaltungssoftware entworfen werden. Die Bücher sollen mit Titel, Autorennamen und Veröffentlichungsjahr gespeichert werden. Außerdem kann jedes Buch ausgeliehen werden. Bei einer Ausleihe sollen das Buch, das Ausleihdatum, und der Name des Ausleihenden für jede Ausleihe gespeichert werden. Außerdem soll in jeder Ausleihe markiert werden, ob das Buch zurückgegeben wurde. Die Software soll über ein GUI-Fenster verfügen, über welches man ein Buch ausleihen oder zurückgeben kann. Wird ein Buch ausgeliehen oder zurückgegeben, soll das GUI-Fenster die Änderung sofort darstellen.

Aufgaben

1. Entwerfen Sie ein UML-Klassendiagramm für das Programm. Nutzen Sie dabei das MVC-Muster.

Lösung: