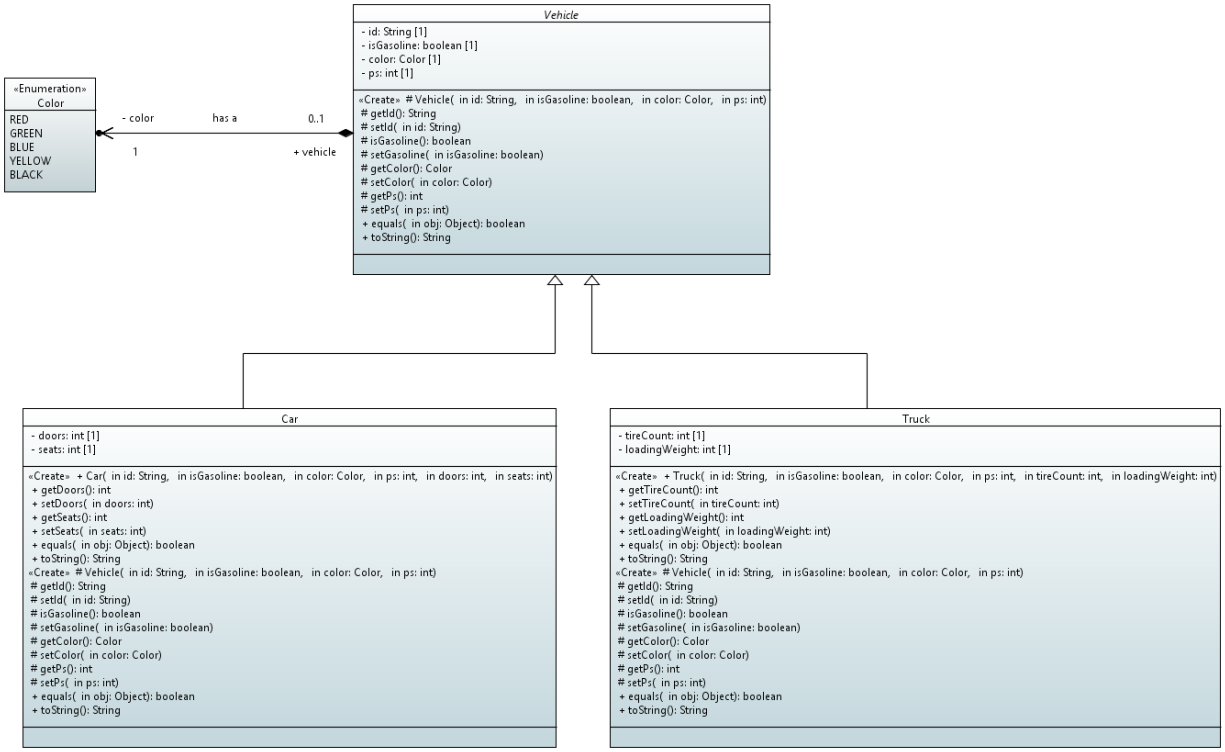
Aufgabe 8.1 - Fahrzeuge

Aufgabe 8.1.1

Erstellen Sie ein Enum. Dieses soll die Konstanten „RED“, „GREEN“, „BLUE“, „YELLOW“ und „BLACK“ annehmen können

Aufgabe 8.1.2

Gegeben ist folgendes schematisches Klassendiagramm. Implementieren Sie die jeweiligen Klassen in Eclipse. Beachten Sie, dass es sich bei der Klasse „Fahrzeug“ um eine abstrakte Klasse handelt. Die Klassen „PKW“ und „LKW“ erben von der Klasse Fahrzeug. Die Attribute entnehmen Sie passend vom schematischen Klassendiagramm. Das Attribut Farbe soll von Aufgabe 15.1 übernommen werden.



Aufgabe 8.1.3

A) Implementieren Sie nun ein Main-Programm, welches eine beliebige Anzahl an PKW-Instanzen beinhaltet. Erstellen Sie in der Klasse PKW eine *equals*-Methode, die zwei Autos mite8inander vergleichen soll. Zwei Autos sollen gleich sein, wenn die PS-Zahl, die Farbe, die Anzahl der Sitze und der Motortyp übereinstimmen. Testen Sie Ihre Implementierung, indem Sie in der main-Methode mehrere Autos miteinander vergleichen und sich das Ergebnis ausgeben lassen.

B) Implementieren Sie nun ein Main-Programm, welches eine Liste von LKWs beinhaltet. Erstellen Sie in der Klasse LKW eine *equals*-Methode, die zwei LKWs miteinander vergleichen soll. Zwei LKWS sollen gleich sein, wenn die Anzahl der Ladegewicht übereinstimmen. Testen

Sie Ihre Implementierung, indem Sie in der main-Methode mehrere LKWs in der Liste speichern und Sie mit einem weiteren LKW vergleichen. Nutzen Sie dazu die *contains*-Methode der Liste.

C) Implementieren Sie nun ein Main-Programm, welches eine Liste von Fahrzeugen beinhaltet. Diese Liste beinhaltet alle Fahrzeuge die als gestohlen gemeldet wurden. Wenn die Polizei ein Fahrzeug untersucht, werden alle Fahrzeuge mir der Liste der gestohlenen Wägen überprüft. Erstellen Sie in der Klasse Fahrzeug eine *equals*-Methode, die zwei Fahrzeuge miteinander vergleicht. Die Unterklassen können diese Methode mithilfe von „*super*“ aufrufen. Testen Sie Ihre Implementierung, indem Sie in der main-Methode mehrere Fahrzeuge(PKWs und LKWs) in der Liste speichern und Sie mit einem weiteren Fahrzeugt vergleichen.

Aufgabe 8.2* – Getränkeautomat

Aufgabe 8.2.1 (Getränkeautomat v2.0)

Schreiben Sie ein Java-Programm, das einen Getränkeautomaten simuliert. Der Automat kennt die drei verschiedenen Getränkesorten, Cola, Wasser und Limo. Eine Flasche Cola kostet 80 Cent, eine Flasche Wasser 60 Cent und eine Flasche Limo 70 Cent. Der Automat akzeptiert als Einwurf die Münzen, 10-Cent, 20-Cent, 50-Cent und 1€. Der Automat ist mit jeweils drei Flaschen pro Sorte bestückt. Das Programm soll den Benutzer zunächst wie folgt nach der Getränkesorte fragen:

```
Wählen Sie das Getränk aus
1=Cola
2=Wasser
3=Limo
```

Bei dieser Frage soll berücksichtigt werden, dass nur noch vorhandene Sorten angeboten werden. Ist beispielsweise kein Wasser mehr im Automaten, soll sich die Frage nach der Sorte so verkürzen:

```
Wählen Sie das Getränk aus
1=Cola
3=Limo
```

Nach der Wahl des Getränks, soll der Benutzer die Münzen, mit denen er bezahlen will, eingeben. Hat der Benutzer sich beispielsweise für eine Flasche Cola entschieden, wird ihm erst der Preis genannt und dann das Geld verlangt.

```
Das gewählte Getränk kostet 80 Cent.
Bitte werfen Sie eine Münze ein(10=10-Cent, 20=20-Cent, 50=50-
Cent, 1=1€)
```

Ihr Programm soll den Benutzer solange nach Münzen fragen, bis der Preis des Getränks gedeckt ist. Falls eine ungültige Eingabe erfolgt, soll es eine entsprechende Fehlermeldung

ausgeben. Sobald ausreichend Münzen eingeworfen worden sind, soll in etwa folgende Meldung ausgegeben werden

```
Danke. Entnehmen Sie Ihre Flasche Wasser.
```

Danach wiederholt sich das Programm und soll den Benutzer wieder nach der Getränkesorte fragen. Der Automat verfügt nicht über die Möglichkeit Wechselgeld auszugeben. Wird beispielswiese die Flasche Cola mit einer 1€-Münze bezahlt, wird die Flasche ausgegeben, aber es gibt keine 20 Cent zurück.

Aufgabe 8.2.2 - Wechselgeld

Erweitern Sie das Programm aus Aufgabe 1 so, dass auch Wechselgeld zurückgegeben wird. Gehen Sie dabei davon aus, dass der Automat zu Beginn über folgende Münzen verfügt

Münze	Stückzahl
10-Cent	10
20-Cent	5

Der Automat soll das Wechselgeld in möglichst wenig Münzen ausgeben. Beim obigen Beispiel der Cola-Flasche, die mit 1€ bezahlt wurde, soll er als Wechselgeld eine 20-Cent Münze zurückgeben und nicht zwei 10-Cent Münzen. Der Automat gibt als Wechselgeld immer 10-Cent- oder 20-Cent-Münzen zurück. Er gibt **keine** 50-Cent- und auch **keine** 1€-Münze zurück!

Die Münzen, mit denen Flaschen bezahlt werden, sollen ebenfalls als Wechselgeld verwendet werden. Wurde z. B. eine Wasserlasche mit sechs 10 Cent-Münzen bezahlt, stehen dem Automaten diese sechs Münzen auch als Wechselgeld zur Verfügung.

Aufgabe 8.3 – Arbeitnehmerclone

8.3 Arbeitnehmerclone

Schreiben Sie ein Java-Programm, welches einen Arbeitnehmer kopiert.

Folgendes ist dabei zu beachten.

- Ein `Department` ist die Abteilung für die ein Arbeitnehmer arbeitet hat zwei Attribute: int id, String name (z.B. {1, “Feldspieler”} oder {2, “Leitung”}
- Ein `Employee` hat drei Attribute: int employeeId, String employeeName, Department department (z.b. {1, “Thomas Müller”, {Department: {1, “Feldspieler”}}} oder {2, “Karl-Heinz Rummenigge”, {Department: {2, “Leitung”}}}

- Aufgabe 8.3.1. Klassen

Erstellen Sie den Code für die oben genannten Klassen. Getter und Setter nicht vergessen!

Aufgabe 8.3.2. Flache Kopie

Eine *Flache Kopie* beschreibt das Kopieren der einfachen Attribute einer Klasse. D.h. sobald eine andere Klasse als Variable in eurer Klasse auftaucht, wird dies nur referenziert statt kopiert. Im obigen Beispiel wird mit einer flachen Kopie zwar ein neues `Employee` Objekt erzeugt, aber nur auf das alte `Department` Objekt referenziert.

Erstelle eine flache Kopie von Employee mit Hilfe der `clone` Methode von Java. Geben Sie anschließend die Object ID von folgenden Objekte aus, um den Unterschied zu verstehen:

- Object ID des alten `Employee` Objektes (System.out.println(yourObject))
- Object ID des neuen `Employee` Objektes
- Object ID des alten `Department` Objektes
- Object ID des neuen `Department` Objektes

Aufgabe 8.3.3. Tiefe Kopie

Eine *Tiefe Kopie* beschreibt das Kopieren aller Attribute einer Klasse. D.h. alle Objekte werden nicht nur referenziert, sondern kopiert. Im Beispiel wird mit einer tiefen Kopie nicht nur ein neues `Employee` Objekt erzeugt, sondern ebenso ein neues `Department` Objekt.

Erstelle eine tiefe Kopie von Employee mit Hilfe der `clone` Methode von Java. Das bedeutet, dass die Clone Methode implementiert werden muss!

Geben Sie anschließend die Object ID von folgenden Objekte aus, um den Unterschied zu verstehen:

- Object ID des alten `Employee` Objektes (System.out.println(yourObject))
- Object ID des neuen `Employee` Objektes
- Object ID des alten `Department` Objektes
- Object ID des neuen `Department` Objektes

Aufgabe 8.4 - Sortieren einer Liste an Personen

Gegeben sind die folgenden zwei Java-Klassen Person und Beruf. Modifizieren Sie beide Klassen, so dass eine Liste `list` von Personen durch einen Aufruf von `Collections.sort(list)` wie folgt sortiert wird:
1. Sortierkriterium: Aufsteigend nach dem Gehalt(salary), das dem Beruf der Person entspricht.
2. Sortierkriterium: Absteigend nach der Größe (size) der Person.

Fügen Sie beiden Klassen einen standardmäßigen Konstruktor hinzu. Testen Sie ihr Programm mit mindesten vier Personen, welche Sie der zu sortierenden Liste hinzufügen.

```
public class Person {
    private String name;
    private int size;
    private Job job;

    //Konstruktor hinzufügen



    public String getName(){
        return this.name;
    }
    public int getSize(){
        return this.size;
    }
    public Job getJob(){
        return this.job;
    }

}

public class Job {
    private String name;
    private double salary;

    //Konstruktor hinzufügen

    public String getName(){
        return this.name;
    }

    public double getSalary(){
        return this.salary;
    }

}
```