```cpp
// FahrzeugAbleitungen.cpp : Definiert den Einstiegspunkt für die Konsolenanwendung.
//
#include "stdafx.h"
#include <iostream>
#include <sstream>
#include <string>
using namespace std;

// Aufzählungstyp für die Führerscheinklassen:
enum Fuehrerscheinklasse {keine, A, A1, A2, AM, B, BE, C1, C1E, C, CE, D1, D1E, D, DE, T, L}; // ? was gilt wofür, oh je...
//
// Datum muss als erste Klasse definiert werden, da die Basisklassen diese class verwenden ("hat ein Datum")
class Datum {
private:
	int Tag;
	int Monat;
	int Jahr;
public:
	Datum(int t, int m, int j) : Tag(t),Monat(m),Jahr(j) {}; // Konstruktor
	// Datum(int t, int m, int j) {Tag=t; Monat=m; Jahr=j;} hätte genau den gleichen Effekt
	string getDatumAsString();
	void coutDatumInOldGermanStyle() { cout << Tag << "." << Monat << "." << Jahr; };
	void coutDatumInNewGermanStyle() { cout << Jahr << "-" << Monat << "-" << Tag; };
};
string Datum::getDatumAsString() {
    ostringstream TempOut;
    TempOut << Tag << "." << Monat << "." << Jahr;
    return TempOut.str();
}
// ------------------------------------------------------------------------
// Person muss als nächstes kommen, da andere Klassen diese class verwenden)
class Person {
protected:  // damit abgeleitete Klassen darauf zugreifen können
	string Name;
	Datum Geburtstag;
public:
	Person(string n, int t, int m, int j) : Name(n),Geburtstag(t,m,j) {}
	string getName() {return Name;}
	void coutGeburtstag() {Geburtstag.coutDatumInOldGermanStyle();}
	string getGeburtstag() { return Geburtstag.getDatumAsString; }
	void changeName(string n) {Name = n;}
};

// wegen Lesereihenfolge jetzt vielleicht der Führerscheininhaber:
class Fuehrerscheininhaber : public Person {
// erbt von "Person" die Attribute "Name" und "Geburtstag", zusätzlich:
private:
	// string Fahrerlaubnis; // über den Typ kann man streiten, vielleicht besser ein enum-Aufzählungstyp?
```

```
49        Fuehrerscheinklasse Fahrerlaubnis;
50        // Datum Erstpruefung;    // hier nicht weiter benutzt....
51  public:
52        Fuehrerscheininhaber(string n, int t, int m, int j, Fuehrerscheinklasse f) :
53          Fahrerlaubnis(f), Person(n,t,m,j) {}
54        Fuehrerscheinklasse getFahrerlaubnis() {return Fahrerlaubnis;}
55        void entzieheFahrerlaubnis() {Fahrerlaubnis = keine;}
56        void erteileFahrerlaubnis(Fuehrerscheinklasse f) {Fahrerlaubnis = f;}
57  };
58
59  // Jetzt die ganzen Fahrzeuge. Zuerst die Basisklasse:
60  class Fahrzeug {
61  protected:
62      const string Marke;
63      const int Baujahr;
64      Person *Eigentuemer; // wir wollen keine neue Person kreieren, sondern eine
        bestehende Person referenzieren
65      static int AnzahlFahrzeuge;
66  public:
67      static int getAnzahl() { return AnzahlFahrzeuge; };
68      Fahrzeug(string m, int j, Person *p) : Marke(m),Baujahr(j) {Eigentuemer = p;
        AnzahlFahrzeuge++;} // public???
69      ~Fahrzeug()
        {AnzahlFahrzeuge--;}
              // public???
70      string getEigentuemer() {return Eigentuemer->getName();}
71      void verkaufe_an(Person *p) {
72          cout << "Das Fahrzeug " << Marke << " wurde soeben von " << Eigentuemer-
            >getName();
73          Eigentuemer = p;
74          cout <<  " an " << Eigentuemer->getName() << " verkauft.\n"; }
75      string getName() {return Marke;}
76  };
77  int Fahrzeug::AnzahlFahrzeuge = 0; // am besten direkt unter die Klasse schreiben,
     in der das static-Attribut deklariert wurde
78
79  // Jetzt die davon abgeleitete Klasse KFZ
80  class KFZ : public Fahrzeug {
81        Datum Erstzulassung;
82        string Kennzeichen;
83        Fuehrerscheinklasse benoetigteFahrerlaubnis;
84  public:
85        KFZ(int t, int m, int j, string kz, Fuehrerscheinklasse bf, string M, int Bj,
          Person *p) : // komplette Angaben...
86        //    Erstzulassung(t,m,j),Kennzeichen(kz),benoetigteFahrerlaubnis(bf),Marke
          (M),Baujahr(Bj),Eigentuemer(p) {} // protected
87        //    Erstzulassung(t,m,j),Kennzeichen(kz),benoetigteFahrerlaubnis(bf)
          {Marke=M; Baujahr=Bj; Eigentuemer=p;} // protected
88            Erstzulassung(t,m,j),Kennzeichen(kz),benoetigteFahrerlaubnis(bf),Fahrzeug
              (M,Bj,p) {}
89  //      KFZ(int t, int m, int j, string kz, string bf, Fahrzeug* F) : // bei
     Benutzung eines bereits existenten Fahrzeugs
90  //          Erstzulassung(t,m,j),Kennzeichen(kz),benoetigteFahrerlaubnis(bf),??? {}
91        string getKennzeichen() {return Kennzeichen;}
92        Datum getErstzulassung() {return Erstzulassung;}
93        Fuehrerscheinklasse getFahrerlaubnis() {return benoetigteFahrerlaubnis;}
```

```
 94 };
 95
 96 // Jetzt die von KFZ abgeleiteten Klassen:
 97 // (Um die Konstruktoren kümmern wir uns später, da gibt es noch Klärungsbedarf, s. u. ...)
 98 class PKW : public KFZ {
 99       bool istCabrio;
100 public:
101       PKW();
102 };
103 class LKW : public KFZ {
104       bool istMegaLiner;
105 public:
106       LKW();
107 };
108 class Motorrad : public KFZ {
109       bool hatBeiwagen;
110 public:
111       Motorrad();
112 };
113 class Bus : public KFZ {
114       int maxAnzahlFahrgaeste;
115 public:
116       Bus();
117 };
118 class Zugmaschine : public KFZ {
119       bool hatKabine;
120 public:
121       Zugmaschine();
122 };
123 // ######################### Hier beginnt das Hauptprogramm ############################
124 int main() {
125     // eine Person brauchen wir auf alle Fälle mal:
126     Person P1("Max Mustermann",4,4,1990);
127     cout << P1.getName() << "s Geburtstag ist am ";
128     P1.coutGeburtstag();
129     cout << endl << endl;
130
131     // gleich noch eine zweite Person:
132     Person* P2 = new Person("Peter Pan",3,3,1995);
133
134     // gleich noch zwei Führerscheininhaberinnen:
135     Fuehrerscheininhaber F1("Petra Panne",2,2,1992,B);
136     Fuehrerscheininhaber* F2 = new Fuehrerscheininhaber("Leia Organa",11,11,1111,A);
137
138     // wenn Fahrzeug und KFZ public-Konstruktoren haben, dann kann ich auch entsprechende Objekte erzeugen:
139     Fahrzeug meinFahrzeug("BobbyCar",2001,&P1);
140     cout << "Eigentuemer von " << meinFahrzeug.getName() << " ist " << meinFahrzeug.getEigentuemer() << endl;
141
142     KFZ meinKFZ(1,3,2012,"FN-XX 123","","KettCar",2009,P2);
143     // die Methoden "getName()" und "getEigentuemer()" sind von "Fahrzeug" geerbt:
144     cout << "Eigentuemer von " << meinKFZ.getName() << " ist " << meinKFZ.getEigentuemer() << endl;
```

```
145
146     cout << "Es gibt inzwischen " << Fahrzeug::getAnzahl() << " Fahrzeuge." << endl
        << endl;
147
148     // Kann man ein Fahrzeug/KFZ auch einem Führerscheininhaber zuordnen???
149     // ein Führerscheininhaber IST ja schließlich eine Person (Ist-Beziehung!)
150     Fahrzeug deinFahrzeug("Dreirad",2005,&F1);
151     cout << "Eigentuemer von " << deinFahrzeug.getName() << " ist " <<
        deinFahrzeug.getEigentuemer() << endl;
152
153     KFZ* deinKFZ = new KFZ(1,1,1889,"ABC999X",A,"Tretroller",1888,F2);
154     cout << "Eigentuemer von " << deinKFZ->getName() << " ist " << deinKFZ-
        >getEigentuemer() << endl;
155
156     cout << "Es gibt inzwischen " << Fahrzeug::getAnzahl() << " Fahrzeuge." << endl
        << endl;
157
158     // Was passiert, wenn Peter und Petra heiraten?
159     F1.changeName("Petra Pan");
160     cout << "Eigentuemer von " << deinFahrzeug.getName() << " ist " <<
        deinFahrzeug.getEigentuemer() << endl;
161
162     // Was für ein Fahrzeug fährt Prinzessin Leia? Das wäre im Fach "Datenbanken"
        kein Problem...
163     // Lösung: eine Element-Liste (Array, Vektor mit Indices) aller Fahrzeuge... ->
        später bei Bedarf
164
165     // Einen PKW anlegen -> Klärungsbedarf:
166     // Sollen Fahrzeug / KFZ weiterhin public sein? Dann könnte ich ein Fahrzeug /
        KFZ anlegen und bei PKW referenzieren.
167     // Oder soll es nur konkrete PKWs geben - Fahrzeug / KFZ werden dann "abstrakt"
        und stehen nicht für Objekte zur Verfügung.
168
169     system("pause");
170     return 0;
171 }
172
```