Source datei "Aufgabenblatt_02.cpp"

```cpp
#include "stdafx.h"
#include <iostream>
#include <string>
#include <vector>
using namespace std;

// **************  Aufgaben 1-3 ******************
class person
{
private:
    int alter;
    string name;
public:
    person() { }; // Standardkonstruktor
    person(string n, int a) // allgemeiner Konstruktor
        :name(n), alter(a) {} // Initialisierung über Liste (siehe vorn)
    void set_alter(int zahl) { alter = zahl; };
    int get_alter(void) { return alter; };
    void set_name(string n) { name = n; };
    string get_name(void) { return name; };
    virtual void drucke_attribute()
    {
        cout << name << "\n";
        cout << alter << "\n";
    };

    virtual string gen_key() = 0; //rein virtuell
};

class student : public person
{
private:
    int matrnummer;
public:
    student() {}; //Standardkonstruktor
    student(string n, int a, int m) //Allgemeiner Konstruktor
        : person(n, a), matrnummer(m) // Initial. über Liste mit Allg.-Konstr. der Elternklasse
    { };
    int get_matrnummer() { return matrnummer; }
    void set_matrnummer(int zeichenkette) { matrnummer = zeichenkette; }
    virtual void drucke_attribute() override
    {
        person::drucke_attribute();
        cout << matrnummer << "\n";
    };

    virtual string gen_key() override
    {
        return "Student_" + get_name();
    };
};

class dozent : public person
{
private:
    string fachgebiet;
public:
    dozent() {}; //Standardkonstruktor
    dozent(string n, int a, string f) //Allgemeiner Konstruktor
        :person(n, a), fachgebiet(f) // Initial. über Liste mit Allg.-Konstr. der Elternklasse
    { };
    string get_fachgebiet() { return fachgebiet; };
    void set_fachgebiet(string f) { fachgebiet = f; };
    virtual void drucke_attribute() override
    {
        person::drucke_attribute();
        cout << fachgebiet << "\n";
    };

    virtual string gen_key() override
    {
        return "Dozent_" + get_name() + fachgebiet;
    };

};
```

```
// ************  Aufgabe 4 ******************
class Person {
    string Familienname;
    string Vorname;
};

class Mitarbeiter : public Person {
    int Mitarbeiternummer;
    // abstrakte Klasse: muss rein virtuelle Methode besitzen
};

// Klasse Bestellung hier nur deklarieren, nicht definieren, da sie für Klasse Kunde und Produkt bekannt sein
class Bestellung;

class Kunde : public Person {
    int Kundennummer;
    Bestellung* Kundenbestellung;
    // abstrakte Klasse: muss rein virtuelle Methode besitzen
};

class Privatkunde : public Kunde {
    string Privatadresse;
};

class Geschäftskunde : public Kunde {
    string Firmenadresse;
};

class Produkt {
    string Produktname;
    int Produkt_ID;
    Bestellung* zugehörige_Bestellung;
    // abstrakte Klasse: muss rein virtuelle Methode besitzen
};

class Downloadprodukt : public Produkt {
    int Speichergröße;
};

class Hardwareprodukt : public Produkt {
    float Produktgewicht;
};

class Bestellung {
    int Bestellnummer;
    vector<Produkt*> Bestellliste;
    Kunde* derKunde;
};


class Versandfirma {
    vector<Mitarbeiter*> Mitarbeiterliste;
    vector<Kunde*> Kundenliste;
    vector<Produkt*> Produktkatalog;
    vector<Bestellung> Alle_Bestellungen;
};

// *********** Aufgabe 1-3
int main(void)
{


    student s1("Meyer", 23, 1234);
    dozent d1("Matthes", 41, "C++");

    cout << "Aufruf ueber Objekt \n";
    s1.drucke_attribute();
    d1.drucke_attribute();

    cout << "Aufruf ueber typspezifische Zeiger \n";
    student *s_ptr = &s1;
    dozent  *d_ptr = &d1;

    s_ptr->drucke_attribute();
    d_ptr->drucke_attribute();

    cout << "Aufruf ueber allg. Personen-Zeiger \n";
    person *p_ptr;

    p_ptr = s_ptr;
    p_ptr->drucke_attribute();
```

```
    cout << "Key: " << p_ptr->gen_key() << "\n";

    p_ptr = d_ptr;
    p_ptr->drucke_attribute();
    cout << "Key: " << p_ptr->gen_key() << "\n";


    vector<person*> Liste = { &s1, &d1 };

    cout << endl << "Liste: " << endl;
    for (auto p : Liste)
    {
        p->drucke_attribute();
        cout << p->gen_key() << endl << endl;
    }


    system("pause");
    return 0;
}
```