Objektorientierte Systeme 1 - SWB2 & TIB2
Hausaufgabe 3

Aufgabe 1: Eine Klasse, die Objekte zählt und identifiziert

Schreiben Sie eine Klasse `ObjectCounter`, die jedem ihrer Objekte eine eindeutige unveränderliche Nummer zuweist und sich merkt, wie viele Objekte der Klasse es gibt. Dazu soll die Klasse folgende Eigenschaften haben:

a) Die konstante Instanzvariable `id` (Identifikator) merkt sich die eindeutige Nummer des Objektes.
b) Die Klassenvariable `maxId` merkt sich die größte bisher vergebene eindeutige Nummer.
c) Die Klassenvariable `number` speichert, wie viele Objekte der Klasse es gibt.
d) Schreiben Sie die Methoden `getId`, `getMaxId` und `getNumber`, um die Variablen auszulesen.
e) Schreiben Sie einen Konstruktor und einen Destruktor.

Schreiben Sie ein Hauptprogramm, um Ihre Klasse zu testen.

Aufgabe 2: Klassenvariablen

Die Klasse `Kunde` soll folgende Attribute besitzen:

```
char *name;
char *ort;
int *alter;
double umsatz;
int transaktion;
int id;
static int anzahl;
```

Das Attribut `umsatz` gibt an, welcher Umsatz in Euro in einer Transaktion (d.h. einem Einkauf) gemacht wird. Die Variable `id` gibt die Kundennummer an. Sie wird in Abhängigkeit der statischen Variablen `anzahl` gesetzt. Die statische Variable `anzahl` wird automatisch inkrementiert, wenn ein Objekt der Klasse `Kunde` angelegt wird. Verfassen Sie

a) einen Konstruktor mit Default-Parametern für `name`, `ort` und `alter`.
b) einen Destruktor
c) einen Kopierkonstruktor (der eine tiefe (!) Kopie erstellt)
d) die Methode `kaufe(double u)`, mit welcher `umsatz` um `u` erhöht und gleichzeitig `transaktion` inkrementiert wird.
e) eine statische Methode `getAnzahl()`, mit welcher die statische Variable `anzahl` zurückgegeben werden kann.
f) eine `print()`-Methode, mit welcher z.B. folgender Text ausgegeben werden kann:

```
Kunde Meier aus Esslingen (ID = 4, 28 Jahre) hatte 3
Transaktion(en) und 230 Euro Unsatz
```

Testen Sie die Klasse Kunde mit mehreren Objekten.

Aufgabe 3: Eine eigene String-Klasse - Teil 2

Erweitern Sie Ihr Programm aus Hausaufgabe 2, indem Sie Operatoren für Ihre String-Klasse `MyString` schreiben.

Folgende Operatoren sollen definiert werden:

a) Der binäre Operator + liefert einen `MyString` zurück, der aus der Konkatenation (Aneinanderreihung) der beiden Operanden besteht.
b) Der Vergleichsoperator == gibt nur dann **`true`** zurück, wenn die in den beiden `MyStrings` gespeicherten Strings identisch sind.
c) Der Operator `[]` greift auf den C-String im `MyString` zu, wie auf einem Array üblich.
d) Der Zuweisungsoperator = weist dem linken Operanden den String des rechten Operanden zu. Achten Sie darauf, dass der Operator kaskadierbar sein muss, d.h. `str1 = str2 = str3;` muss auch korrekt funktionieren.
e) Der Ausgabeoperator << gibt den gespeicherten C-String an einen Stream, z.B. an `cout`.

Testen Sie die Operatoren mit dem folgenden Programm:

```
#include <iostream>
#include "MyString.hpp"
using namespace std;

int main() {
    MyString s1("Eins ");
    MyString s2("Zwei ");
    MyString s3("Drei ");
    MyString s4("NIX");
    s4 = s1 + s2 + s3;
    s2 = s1;
    cout << "Sind s1 und s2 gleich? "
         << boolalpha << (s1 == s2) << endl;
    cout << s1 << endl;
    cout << s2 << endl;
    cout << s3 << endl;
    cout << s4 << endl;
    return 0;
}
```

Aufgabe 4: Freundfunktionen

Die Klasse Duck enthält private Datenelemente, die über Freundfunktionen manipuliert werden sollen. Ergänzen Sie die fehlenden Teile und machen Sie sich deren Sinn klar. Was ist der wesentliche Unterschied zwischen einer Freundfunktion und einer Methode?

```
#include <iostream>
#include <string>
using namespace std;

class Duck {
    friend void zaster (/* HIER */, long n);
    friend void kontoauszug (const Duck p);
```

```
    friend void panzerknacker (/* HIER */, long n);
public:
    Duck ();
private:
    long bankkonto;
    string waehrung; //Symbol für Währung
    static long familienbesitz;
};

long Duck::familienbesitz = 0;

Duck::Duck () {
    bankkonto = 0;
    waehrung = "$";
}

void zaster (/* HIER */, long money) {
    /* HIER */ // bankkonto um money erhöhen
    /* HIER */ // familienbesitz um money erhöhen
    cout << "Hier ist zaster, zaster eine "
            "Freund-Funktion der Klasse Duck" << endl;
    cout << "Die Einnahme ist " << money << " " << p.waehrung << endl;
};

void panzerknacker (/* HIER */, long money) {
    /* HIER */ //bankkonto um money erniedrigen
    /* HIER */ //familenbesitz um money erniedrigen
    cout << "panzerknacker eine Freund-Funktion ?!" << endl;
    cout << "Die Panzerknacker waren da" << endl;
    cout << "Sie haben " << money << " " << p.waehrung
         << " geraubt" << endl;
};

void kontoauszug (Duck p) {
    cout << "Kontoauszug: " << p.bankkonto << " "
         << p.waehrung << endl;
    cout << "Familienbesitz: " << p.familienbesitz
         << " " << p.waehrung << endl;
}

int main (void) {
    Duck dagobert;
    cout << "dagobert bekommt Geld" << endl;
    zaster(dagobert, 1000000);
    Duck daisy;
    cout << "daisy bekommt Geld" << endl;
    zaster(daisy, 100);
```

```
    cout << "Der Besitz von dagobert ist ";
    kontoauszug (dagobert);
    cout << endl;
    cout << "Der Besitz von daisy ist ";
    kontoauszug (daisy);
    cout << endl;
    cout << "armer dagobert" << endl;
    panzerknacker (dagobert, 1000000);
    cout << "Der Besitz von dagobert ist ";
    kontoauszug (dagobert);
    return 0;
}
```

Aufgabe 5: Ein Fuhrpark - Teil 1

Schreiben Sie eine Klasse `Fahrzeug`. Die Klasse hat folgende Eigenschaften.

a) Jedes Fahrzeug hat eine Instanzvariable `kz` (Kennzeichen) vom Typ `MyString`, die das Kennzeichen des Fahrzeugs speichert.
b) Die konstante Instanzvariable `vin` (Vehicle Identification Number) vom Typ **`int`** hält die eindeutige Seriennummer des Fahrzeugs.
c) Sorgen sie dafür, dass jedes Fahrzeug automatisch eine eindeutige Seriennummer erhält.
d) Die Instanzvariable `km` gibt an, wie viele Kilometer das Fahrzeug bisher gefahren ist.
e) Der Konstruktor nimmt einen C-String als Parameter, der direkt an den Konstruktor des `MyStrings` übergeben wird, um das Kennzeichen zu initialisieren.
f) Schreiben Sie eine Instanzmethode `fahren(`**`double`**` km)`, die ein Fahrzeug die übergebene Anzahl von Kilometern fahren lässt. Dadurch erhöht sich der Kilometerstand des Fahrzeugs entsprechend.
g) Der Operator `<<` gibt die Informationen zu einem Fahrzeug `f` an einen Stream (z.B. `cout << f << endl;`). Die Ausgabe des Hauptprogramms unten kann zum Beispiel so aussehen:

```
Kz = ES - H 123  VIN = 1  km = 0.0
Kz = ES - H 123  VIN = 1  km = 101.5
Kz = ES - M 4711  VIN = 2  km = 10.6
```

Testen Sie Ihr Programm mit dem folgenden Hauptprogramm.

```
int main() {
    Fahrzeug f1("ES - H 123");
    cout << f1 << endl;
    f1.fahren(101.5);
    cout << f1 << endl;
    Fahrzeug f2("ES - M 4711");
    f2.fahren(10.57);
    cout << f2 << endl;
    return 0;
}
```