Objektorientierte Systeme 1 - SWB2 & TIB2
Labor 3

Aufgabe 1: Ein Fuhrpark - Teil 2

Erweitern Sie die Klasse `Fahrzeug` aus der Hausaufgabe 3 folgendermaßen:

a) Ergänzen Sie die Klasse `Fahrzeug` um eine Klassenvariable `summeKm`, die die Kilometerleistung aller Fahrzeuge aufsummiert.
b) Die Klassenmethode `getSummeKm` liefert den Wert von `summeKm` zurück.
c) Der Destruktor passt die Klassenvariable `summeKm` entsprechend an, da das Fahrzeug den Fuhrpark verlassen hat.

Erweitern Sie Ihr Hauptprogramm, um die erweiterte Klasse – insbesondere den Destruktor – zu testen.

Aufgabe 2: Eine eigene String-Klasse - Teil 3

Das Arbeiten mit konstanten Objekten erfordert konstante Instanzmethoden.

Ergänzen Sie Ihre Klasse `MyString` aus Hausaufgabe 3 so, dass das folgende Hauptprogramm funktioniert.

```
#include "MyString.hpp"

int main() {
    const MyString cs("Ein konstanter String");
    MyString s(cs);
    s.assign(cs);
    s.append(cs);
    cout << cs.c_str() << endl;
    cout << cs.size() << endl;
    cout << cs.capacity() << endl;
    cout << boolalpha << cs.empty() << endl;
    s = cs + cs;
    cout << (cs == cs) << endl;
    s = cs;
    cout << cs << endl;
    s.at(1) = 'X';
    s[2] = 'Y';
    cout << s << endl;
    return 0;
}
```

Aufgabe 3: Grafische Objekte - Teil 2

Erweitern Sie Ihre Klassen zu den grafischen Objekten aus Labor 2 in folgender Weise.

a) Deklarieren Sie alle Instanzmethoden soweit möglich als konstant.

b) Ergänzen Sie die Klassen `Point`, `Circle` und `Polygonline` um je eine Funktion `toString()`. Die Funktion `toString()` gibt als String zurück, was bisher die `print`-Methode auf dem Bildschirm ausgegeben hat. D.h. die Programmzeilen `o.print();` und `cout << o.toString() << endl;` für ein Point-, Circle- und Polygonline-Objekt `o` sollen das gleiche Ergebnis liefern. Nutzen Sie Stringstreams wie in der Vorlesung gezeigt.

c) Schreiben Sie einen Einleseoperator >>, der von einem Stringstream einen Punkt einliest. Folgender Programmausschnitt muss funktionieren:

```
istringstream is(" (1.1, 2.2) ist ein Punkt.");
Point p;
is >> p;
```

d) Ergänzen Sie die Klasse `Point`, um einen Konvertierkonstruktor, der einen String, z.B. `(1.1, 2.2)`, als Parameter nimmt und die Koordinaten des neuen Objektes entsprechend setzt. Im Prinzip handelt es sich um die Umkehrung der Funktion `toString()`.

e) Ergänzen Sie die Klassen `Circle` und `Polygonline`, um je einen Konvertierkonstruktor, der einen String als Parameter nimmt und die Attributwerte des neuen Objektes entsprechend setzt. Testen Sie die Konstruktoren mit dem folgenden Hauptprogramm.

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

// Hauptprogramm
int main(void) {
  string str1 ("(1.1,2.1)");
  string str2 ("<(5.5, 6.6), 10.1>");
  string str3 ("|(1.1,1.2) - (2.1, 2.2) -(3.1,3.2) |");
  Point p (str1);
  Circle c (str2);
  Polygonline l (str3);

  cout << p.toString() << endl;
  cout << c.toString() << endl;
  c.move(1.0,2.0);
  cout << c.toString() << endl;
  cout << l.toString() << endl;
  return 0;
}
```

f) Modifizieren Sie Ihr Hauptprogramm aus dem vorherigen Punkt so, dass ein String zur Konvertierung von der Standardeingabe gelesen und dann in das entsprechende grafische Objekt konvertiert wird.

Aufgabe 4: Grafische Objekte - Teil 3

Ergänzen Sie das Programm aus der vorherigen Aufgabe um die folgenden Operatoren:

a) Definieren Sie je einen binären Operator << als Freundfunktion der Klassen der grafischen Objekte `Point`, `Circle` und `Polygonline` um z.B. einen Punkt `p1` direkt mit

`cout << p1 << endl;` ausgeben zu können. Der linke Operand und der Rückgabewert ist vom Typ **`ostream`**`&`.

b) Definieren Sie die binären Operatoren + und − und den unären Operator −, die Punkte koordinatenweise addieren, subtrahieren bzw. negieren und das Ergebnis als einen neuen Punkt zurückliefern, ohne die Operanden zu verändern. Definieren Sie diese Operatoren nicht als globale Funktionen oder Freundfunktionen. Beachten Sie, dass eine Verkettung von Operatoren möglich sein soll `(p1 + p2 + p3)`.

c) Überladen Sie den Operator +, so dass auch Additionen wie `p1 + 3.5` und `3.5 + p1` durchgeführt werden können. Dabei wird der Wert vom Typ **`double`** zu jedem der beiden Koordinaten des Punkts hinzuaddiert.

d) Definieren Sie den unären Operator ++ sowohl als Präfix- als auch als Postfixoperatoren für die Klasse `Point`. Analog zu ++ für den Typ **`int`** soll der Operator ++ bei Punkten beide Koordinaten inkrementieren. Beachten Sie, dass `p1 = p2++;` und `p1 = ++p2;` unterschiedliche Werte an `p1` liefern.

e) Definieren Sie einen binären Operator +, der einen Punkt an einen Linienzug anhängt. Wenn `l` ein Objekt der Klasse `Polygonline` ist und `p` ein `Point`, dann fügt `l+p` den Punkt `p` an den Linienzug `l` an und liefert den Linienzug `l` zurück.

f) Definieren Sie einen binären Operatoren +, der einen Linienzug an einen anderen Linienzug anhängt. Wenn `l1` und `l2` Objekte der Klasse `Polygonline` sind, dann hängt `l1 + l2` den Linienzug `l2` an `l1` an und liefert den Linienzug `l1` zurück.

Zum Testen nutzen Sie das folgende Hauptprogramm:

```
#include <iostream>
#include <string>
#include "Circle.hpp"
#include "Polygonline.hpp"
using namespace std;

// Hauptprogramm
int main(void)
{
    Point p1(0,0);
    const Point p2(2,2);
    const Point p3(3,3);
    Circle c(p1, 1.1);
    cout << "Circle c: " << c << endl;
    p1 = p1 + 0.5;
    p1 = 0.5 + p1;
    cout << "p1: " << p1 << endl;
    cout << "p2: " << p2 << endl;
    cout << "p3: " << p3 << endl;
    Point p4 = p1 + p2 - p3 + 4.0;
    cout << "p4: " << p4 << endl;
    p1 = -p4;
    cout << "p1: " << p1 << endl;
    cout << "p4: " << p4 << endl;
    Point p5 = p1++;
    cout << "p5: " << p5 << endl;
```

```
    cout << "p1: " << p1 << endl;
    p5 = ++++++++p1;
    cout << "p5: " << p5 << endl;
    cout << "p1: " << p1 << endl;
    cout << "p2: " << p2 << endl;
    cout << "p3: " << p3 << endl;
    cout << "p4: " << p4 << endl;
    Polygonline l1;
    cout << "l1: " << l1 << endl;
    (l1 + p1) + p2;
    cout << "l1: " << l1 << endl;
    const Polygonline l2(p4);
    l1 + l2;
    cout << "l1: " << l1 << endl;
    cout << "l2: " << l2 << endl;
    return 0;
}
```