Vererbung

AUFGABE 4.1 (Vererbung Figur)

Ziel dieser Übung ist es, das Konzept der Vererbung anhand einer Hierarchie von geometrischen Figuren zu veranschaulichen.

(a) Definieren Sie in der Datei `Figuren.cpp` zunächst die Klassen `Rechteck` und `Kreis`. Versehen Sie sie mit passenden Attributen (Länge, Breite, Radius) und einer Methode `oberflaeche()`.

(b) Anschließend fügen Sie eine Klasse `buntesRechteck` hinzu. Diese Klasse muss nur ein weiteres Attribut, Farbe (vom Typ unsigned int) haben.

(c) Um diese Klassen etwas realistischer (und brauchbarer) zu machen, fügen Sie jeder Klasse einen Konstruktor hinzu (falls noch nicht geschehen).

(d) Testen Sie mit dieser vereinfachten `main`:

```
int main() {
      buntesRechteck recht(4.3, 12.5, 4);
      cout << recht.get_laenge() << endl;
      return 0;
}
```

(e) Fahren Sie fort, indem Sie die folgenden Klassen definieren:

- Die Klasse `Figur`, die zwei Attribute `x` und `y` (die den Mittelpunkt der Figur darstellen), eine Methode `anzeigen(ostream&)`, die einfach die Koordinaten des Mittelpunkts anzeigt, und einen Konstruktor, der die Koordinaten des Mittelpunkts übernimmt, enthält.
- Lassen Sie die Klassen `Kreis` und `Rechteck` von der Klasse `Figur` erben.

Testen Sie mit dieser `main`:

```
int main() {
      buntesRechteck recht(4.3, 12.5, 4);
      cout << recht.get_laenge() << endl;
      recht.anzeigen(cout);
      cout << endl;

      Kreis kr;
      kr.set_mitte(2.3, 4.5);
      kr.set_radius(12.2);
      kr.anzeigen(cout);
      cout << endl;
      return 0;
}
```

(f) Um interessanter zu sein, sollten die Möglichkeiten der Klasse `Figur` auf die Konstruktoren der Klassen `Rechteck` und `Kreis` übertragen werden, und die Methode `anzeigen` sollte eventuell erweitert werden. Sie sollte auch den Radius anzeigen sowie weiterhin den Mittepunkt anzeigen.

Testen Sie das Programm mit der folgenden `main`:

```
int main() {
      buntesRechteck recht(4.3, 12.5, 4);
      cout << recht.get_laenge() << endl;
      recht.anzeigen(cout);
      cout << endl;

      Kreis kr(12.3, 2.3, 4.5);
      kr.anzeigen(cout);
      cout << endl;

      Rechteck recht2(1.2, 3.4, 12.3, 43.2);
      recht2.anzeigen(cout);
      cout << endl;

      return 0;
}
```

AUFGABE 4.2 (Fahrzeug) - **Abgabe**

Definieren Sie in der Datei `Fahrzeug.cpp` eine Klasse `Fahrzeug`, deren Attribute Informationen enthalten, die für jeden Fahrzeugtyp gelten: Marke, Kaufdatum, Kaufpreis, aktueller Preis.

(a) Definieren Sie einen Konstruktor, der als Parameter die drei Attribute Marke, Kaufdatum und Kaufpreis (der aktuelle Preis wird später berechnet) enthält.

(b) Definieren Sie eine öffentliche Methode `void anzeigen(ostream&) const;`, die den Status der Instanz, d. h. die Werte ihrer Attribute, anzeigt.

(c) Definieren Sie dann zwei Klassen Auto und Flugzeug, die von der Klasse Fahrzeug erben und die folgenden zusätzlichen Attribute haben:

Für die Klasse Auto:
- seinen Hubraum;
- seine Anzahl an Türen;
- seine Leistung;
- seinen Kilometerstand;

Für die Klasse Flugzeug:
- seinen Typ (Propeller oder Düsenflugzeug);
  - enum Flugzeug_Typ { PROPELLER, DUESENFLUGZEUG };
- seine Flugstundenzahl.

(d) Definieren Sie für jede dieser Klassen einen Konstruktor, der alle Attribute explizit initialisiert, sowie eine Methode, die die Werte der Attribute anzeigt. Sowohl der Konstruktor als auch die Anzeigemethode müssen die entsprechenden Methoden der Elternklasse verwenden!

(e) Fügen Sie der Klasse `Fahrzeug` eine Methode `void berechnePreis()` hinzu, die den aktuellen Preis ermittelt. Der aktuelle Preis wird berechnet, indem vom Kaufpreis 1 % pro Jahr, das seit dem Kauf vergangen ist, abgezogen wird.

(f) Definieren Sie diese Methode in den beiden Unterklassen `Auto` und `Flugzeug` neu, um den aktuellen Preis nach bestimmten Kriterien zu berechnen, und aktualisieren Sie das Attribut für den aktuellen Preis:

Bei einem Auto entspricht der aktuelle Preis dem Kaufpreis abzüglich:

- 2% für jedes Jahr seit dem Kaufdatum bis zum aktuellen Datum;
- 5% für jeweils 10.000 gefahrene Kilometer (es wird aufgerundet);
- 10%, wenn es sich um ein Fahrzeug der Marke "Renault" oder "Fiat" handelt (eine völlig willkürliche Wahl, die man natürlich frei ändern kann);
- und plus 20%, wenn es sich um ein Fahrzeug der Marke "Ferrari" oder "Porsche" handelt (gleiche Bemerkung wie oben).

Bei einem Flugzeug entspricht der Listenpreis dem Kaufpreis abzüglich:

- 10% für je 1000 Flugstunden, wenn es sich um ein Düsenflugzeug handelt;
- 10% für jede 100 Flugstunden, wenn es sich um ein Propellerflugzeug handelt.
- Der Preis muss positiv bleiben (d. h., wenn er negativ ist, setzen wir ihn auf 0).

Um die oben implementierten Methoden zu testen, ergänzen Sie die `main` wie folgt:

```
int main(){
	vector<Auto> garage;
	vector<Flugzeug> flugplatz;
	garage.push_back(Auto("Peugeot", 1998, 147325.79, 2.5, 5, 180.0,
		12000));
	garage.push_back(Auto("Porsche", 1985, 250000.00, 6.5, 2, 280.0,
		81320));
	garage.push_back(Auto("Fiat", 2001, 7327.30, 1.6, 3, 65.0,
		3000));
	flugplatz.push_back(Flugzeug("Cessna", 1972, 1230673.90, PROPELLER,
		250));
	flugplatz.push_back(Flugzeug("Unbekannt", 1992, 4321098.00,
DUESENFLUGZEUG,
		1300));
	for (auto pkw : garage) {
		pkw.berechnePreis();
		pkw.anzeigen(cout);
	}
	for (auto flugzeug : flugplatz) {
		flugzeug.berechnePreis();
		flugzeug.anzeigen(cout);
	}
	return 0;}
```

Ausgabebeispiel für das Jahr 2015:

```
---- Auto ----
Marke: Peugeot, Kaufdatum: 1998, Kaufpreis: 147326, aktueller Preis: 89868.7
2.5 Liter, 5 Tueren, 180 PS, 12000 km.
 ---- Auto ----
Marke: Porsche, Kaufdatum: 1985, Kaufpreis: 250000, aktueller Preis: 50000
6.5 Liter, 2 Tueren, 280 PS, 81320 km.
 ---- Auto ----
Marke: Fiat, Kaufdatum: 2001, Kaufpreis: 7327.3, aktueller Preis: 4542.93
1.6 Liter, 3 Tueren, 65 PS, 3000 km.
 ---- Flugzeug mit Propellern ----
Marke: Cessna, Kaufdatum: 1972, Kaufpreis: 1.23067e+06, aktueller Preis: 923005
250 Flugstunden.
 ---- Flugzeug mit Duesen ----
Marke: Unbekannt, Kaufdatum: 1992, Kaufpreis: 4.3211e+06, aktueller Preis:
3.75936e+06
1300 Flugstunden.
```

AUFGABE 3.4 (3D Vektoren) - **Abgabe**

Hier geht es darum, die Klasse zu erstellen, die Vektoren in 3D darstellt. Vom Design her erbt die Klasse Vektor von der Klasse Punkt3D, die in Aufgabe 1.4 vorgestellt wurde.

(a) Beginnen Sie mit der Wiederholung dieser Aufgabe und passen Sie sie an Ihre neuen Kenntnisse an: Konstruktoren und Operatorüberladungen.

(b) Wechseln Sie dann zur Klasse `Vektor`. Zusätzlich zu allen geerbten Methoden muss diese neue Klasse die folgenden Operationen durchführen können:

- Addition und Subtraktion von zwei Vektoren:

```
Vektor& Vektor::operator+=(const Vektor&);
Vektor& Vektor::operator-=(const Vektor&);
const Vektor operator+(Vektor, const Vektor&);
const Vektor operator-(Vektor, const Vektor&);
```

- Der negative Vektor:

```
const Vektor Vektor::operator-() const;
```

- Multiplikation mit einem Skalar:

```
Vektor& Vektor::operator*=(double);
const Vektor operator*(Vektor, double);
const Vektor operator*(double, const Vektor&);
```

- Skalarprodukt von zwei Vektoren:

```
double operator*(const Vektor&, const Vektor&);
```

- Berechnung der Norm des Vektors (Quadratwurzel des Skalarprodukts mit sich selbst);

- Winkel zwischen zwei Vektoren als Arkuskosinus ihres Skalarprodukts geteilt durch das Produkt der Normen:

```
double winkel(const Vektor& v1, const Vektor& v2){
      return acos((v1 * v2) / (v1.norme() * v2.norme()));
}
```

(c) Testen Sie diese Operationen an beliebigen Vektoren, z. B.:

```
Vektor V1: (1, 2, -0.1)
Vektor V2: (2.6, 3.5, 4.1)
V1 + V2: (3.6, 5.5, 4)
V1 - V2: (-1.6, -1.5, -4.2)
 -V1: (-1, -2, 0.1)
3.2 * V1: (3.2, 6.4, -0.32)
V1 * V2: 9.19
Normale von V1: 2.2383
Normale von V2: 5.98498
Normale von V1+V2: 7.6948
Winkel zwischen V1 et V2: 0.814798
Winkel zwischen V2 et V1: 0.814798
```

LÖSUNG AUFGABE 4.1 (Vererbung Figur)

Teilaufgaben (a)-(d):

```
#define _USE_MATH_DEFINES
#include <iostream>
#include <string>

using namespace std;

class Rechteck {
protected:
	double laenge, breite;
public:
	Rechteck(double l, double b): laenge(l), breite(b) {}
	double oberflaeche(double l, double b) const{
		return l * b;
	}
	double get_laenge() const {
		return laenge;
	}
	double get_breite() const {
		return breite;
	}
	void set_laenge(double wert) {
		laenge = wert;
	}
	void set_breite(double wert) {
		breite = wert;
	}
};

class Kreis {
	double radius;
public:
	double oberflaeche(double r) {
		return M_PI * r * r;
	}
	void set_radius(double wert) {
		if (wert < 0.0) wert = 0.0;
		radius = wert;
	}
	double get_radius() const{
		return radius;
	}
};

class buntesRechteck: public Rechteck {
protected:
	unsigned int farbe;
public:
	buntesRechteck(unsigned int f, double l, double b) : farbe(f),
Rechteck(l, b) {};

};
```

```
int main() {
	buntesRechteck recht(4.3, 12.5, 4);
	cout << recht.get_laenge() << endl;
	return 0;
}
```

Teilaufgabe (e)

```
#define _USE_MATH_DEFINES
#include <iostream>
#include <string>

using namespace std;
class Figur {
protected:
	double mitte_x, mitte_y;
public:
	Figur(double x = 0.0, double y = 0.0): mitte_x(x), mitte_y(y) { }
	void set_mitte(double x, double y) {
		mitte_x = x;
		mitte_y = y;
	}
	void get_mitte(double& x, double& y) const {
		x = mitte_x;
		y = mitte_y;
	}
	void anzeigen(ostream& ausgabe) {
		ausgabe << "Mittelpunkt = (" << mitte_x << " ," << mitte_y << ")";
	}
};
class Rechteck: public Figur {
protected:
	double laenge, breite;
public:
	Rechteck(double l, double b): laenge(l), breite(b) {}
	double oberflaeche(double l, double b) const{
		return l * b;
	}
	double get_laenge() const {
		return laenge;
	}
	double get_breite() const {
		return breite;
	}
	void set_laenge(double wert) {
		laenge = wert;
	}
	void set_breite(double wert) {
		breite = wert;
	}
};

class Kreis: public Figur{
	double radius;
public:
```

```
	double oberflaeche(double r) {
		return M_PI * r * r;
	}
	void set_radius(double wert) {
		if (wert < 0.0) wert = 0.0;
		radius = wert;
	}
	double get_radius() const{
		return radius;
	}
};

class buntesRechteck: public Rechteck {
protected:
	unsigned int farbe;
public:
	buntesRechteck(unsigned int f, double l, double b) : farbe(f), Rechteck(l, b)
{};

};

int main() {
	buntesRechteck recht(4.3, 12.5, 4);
	cout << recht.get_laenge() << endl;
	recht.anzeigen(cout);
	cout << endl;

	Kreis kr;
	kr.set_mitte(2.3, 4.5);
	kr.set_radius(12.2);
	kr.anzeigen(cout);
	cout << endl;
	return 0;
}
```

Teilaufgabe (f)

Konstruktoren anpassen:

```
Rechteck(double l, double b, double x, double y): Figur(x, y), laenge(l), breite(b)
{}
Kreis(double r, double x = 0.0, double y = 0.0): Figur(x,y), radius(r){}
```

Anpassung der Methode `anzeigen` (Prototypen in den Klassen nicht vergessen!):

```
void Kreis::anzeigen(ostream& ausgabe) {
	Figur::anzeigen(ausgabe);
	ausgabe << " , r = " << radius;
}

void Rechteck::anzeigen(ostream& ausgabe) {
	Figur::anzeigen(ausgabe);
	ausgabe << ", Laenge = " << laenge << " , Breite = " << breite;
}
```