Polymorphismus I

AUFGABE 5.1 (Flughafen)

Das Ziel dieser Übung besteht darin, den Begriff des Polymorphismus anhand einer heterogenen Sammlung von Fahrzeugen zu veranschaulichen.

(a) Beginnen Sie in der Datei „flughafen.cpp“ mit der Definition der Klassen „Fahrzeug“, „Flugzeug“ und „Auto“. Flugzeuge und Autos sind Fahrzeuge. Statten Sie jede dieser Klassen mit einer Methode aus, die die Klasse anzeigt aus (Ausgabetext Ihrer Wahl).

(b) Anschließend erstellen Sie eine Klasse Flughafen, die eine Reihe von Fahrzeugen verwalten muss, die sowohl aus Autos als auch aus Flugzeugen bestehen. Ohne Polymorphismus ist man gezwungen, zwei verschiedene Arrays zu erstellen, da zwei verschiedene Arten von Objekten verarbeitet werden. Dies ist an sich vollkommen gültig, aber für die Zwecke der Übung gehen Sie hier davon aus, dass Sie sie lieber gemeinsam als eine Sammlung von Fahrzeugen verwalten möchten. Definieren Sie die Flughafen Klasse vor diesem Hintergrund. Statten Sie sie mit drei Methoden aus:

- `fahrzeuge_anzeigen(ostream&)` ermöglicht die Anzeige aller Fahrzeuge des Flughafens;
- `fahrzeuge_hinzufuegen(...)`, um ein Fahrzeug zum Flughafen hinzuzufügen;
- `fahrzeuge_leeren()` löscht alle Fahrzeuge vom Flughafen.

Probieren Sie eine eigene Umsetzung bevor Sie weiterlesen!

Eine mögliche Klassendefinition:

```
class Flughafen {
public:
	void fahrzeuge_anzeigen(ostream&);
	void fahrzeuge_hinzufuegen(Fahrzeug const&);
	void fahrzeuge_leeren();
protected:
	vector<Fahrzeug> fahrzeuge;
};
```

Aber das würde es nicht erlauben, Autos und Flugzeuge als solche anzuzeigen, kurz gesagt, Polymorphismus zu verwenden.

Dies erfordert die Verwendung von Referenzen oder Zeigern auf Objekte und nicht auf die Objekte selbst. Da wir keine Referenz in einen Vektor einfügen können, müssen wir hier ein dynamisches Array von Zeigern verwenden.

```
class Flughafen {
public:
	void fahrzeuge_anzeigen(ostream&) const;
	void fahrzeuge_hinzufuegen(Fahrzeug*);
	void fahrzeuge_leeren();
protected:
	vector<Fahrzeug*> fahrzeuge;
};
void Flughafen::fahrzeuge_anzeigen(ostream& ausgabe) const {
	for (auto const& fahrzeug : fahrzeuge) {
		fahrzeug->anzeige(ausgabe);
	}
}
void Flughafen::fahrzeuge_hinzufuegen(Fahrzeug* f) {
	fahrzeuge.push_back(f);
}
void Flughafen::fahrzeuge_leeren() {
	fahrzeuge.clear();
}
```

Damit die dynamische Auflösung der Links umgesetzt werden kann, müssen auch die auf die Objekte vom Typ Fahrzeug aufgerufenen Methoden virtuell sein.

Wenn es sich bei „fahrzeuge[i]“ um einen Zeiger auf ein Flugzeug handelt, ruft „fahrzeuge[i]->anzeige()“ auf diese Weise die Anzeigemethode der Klasse „Flugzeug“ und nicht die Methode von „Fahrzeug“ auf.

Unsere Fahrzeugklasse muss daher sein:

```
class Fahrzeug {
public:
	Fahrzeug(// zum Beispiel...
		string marke, unsigned int daum, double preis
	);
	virtual void anzeige(ostream&) const;
	virtual ~Fahrzeug() {}
protected:
	// zum Beispiel...
	string marke;
	unsigned int kaufdatum;
	double kaufpreis;
	double aktueller_preis;
};
```

Die Flughafen-Klasse ist jetzt nutzbar. Damit der Code vollständig zufriedenstellend ist, müssten jedoch die Mittel bereitgestellt werden, um möglicherweise den Speicher der im Vektor platzierten Objekte freizugeben (falls die Funktion, die sie erstellt und zum Vektor hinzugefügt hat sie löschen möchte). (Genau genommen wäre es auch notwendig, ein bestimmtes Element löschen zu können und für alle Fälle eine Deep-Copy-Methode bereitzustellen.)

(c) Fügen Sie eine Methode `fahrzeuge_loeschen()` hinzu, die diese Speicherbereinigung durchführt. Anschließend können Sie mit folgender `main()` testen:

```
int main() {
	Flughafen ffm;
	ffm.fahrzeuge_hinzufuegen(new
		Auto("Peugeot", 1998, 147325.79, 2.5, 5, 180.0, 12000));
	ffm.fahrzeuge_hinzufuegen(new
		Auto("Porsche", 1985, 250000.00, 6.5, 2, 280.0, 81320));
	ffm.fahrzeuge_hinzufuegen(new
		Flugzeug("Cessna", 1972, 1230673.90, PROPELLER, 250));
	ffm.fahrzeuge_hinzufuegen(new
		Flugzeug("Unbekannt", 1992, 4321098.00, DUESENFLUGZEUG, 1300));
	ffm.fahrzeuge_hinzufuegen(new
		Auto("Fiat", 2001, 7327.30, 1.6, 3, 65.0, 3000));
	ffm.fahrzeuge_anzeigen(cout);
	ffm.fahrzeuge_leeren();
	return 0;
}
```

AUFGABE 5.2 (Formen)

Das Ziel dieser Übung besteht darin, die Probleme zu veranschaulichen, die auftreten können, wenn Sie Objekte auf polymorphe Weise manipulieren möchten (ein Objekt kann ein anderes Objekt ersetzen). Dabei werden einige einfache geometrische Formen manipuliert, indem sie mit einer Methode verknüpft werden, die ihre Beschreibung anzeigt.

(a) Definieren Sie in einer Datei „formen.cpp“ eine Form-Klasse, indem Sie ihr eine Methode `void beschreibung()` zuweisen, die auf dem Bildschirm Folgendes anzeigt: „Dies ist eine Form!“.

(b) Fügen Sie dem Programm eine Kreis-Klasse hinzu, die von der Form-Klasse erbt und über die Methode `void beschreibung()` verfügt, die auf dem Bildschirm Folgendes anzeigt: „Dies ist ein Kreis. ".

(c) Kopieren Sie dann die folgende `main()`-Funktion und testen Sie Ihr Programm:

```
int main() {
	Form f;
	Kreis k;
	f.beschreibung();
	k.beschreibung();
	return 0;
}
```

(d) Fügen Sie nun folgende Zeilen am Ende Ihrer main hinzu:

```
	Form f2(k);
	f2.beschreibung();
```

(e) Testen Sie Ihr Programm erneut. Was geschieht? Warum passiert das?

(f) Fügen Sie Ihrem Programm eine weitere Methode `void anzeigeBeschreibung(Form& f)` hinzu, welche die Beschreibung der ihr als Parameter übergebenen Form ausgibt indem sie die Methode `beschreibung()` nutzt.
(g) Ändern Sie die `main` und testen Sie das Programm. Sind Sie mit dem Ergebnis zufrieden? Warum?
(h) Ändern Sie Ihr Programm, indem Sie ein einziges Wort hinzufügen, um das Erwartete Verhalten zu ermöglichen.