Mehrfachvererbung

AUFGABE 7.1 (Zoo)

Der Zweck dieser Übung besteht darin, am Beispiel aus der Vorlesung das Konzept der Mehrfachvererbung und der virtuellen Klasse zu veranschaulichen, indem das Beispiel der Ovoviviparität vertieft wird.

(a) Definieren Sie in der Datei `zoo.cpp` zunächst eine `Vivipare`-Klasse, die eine vorzeichenlose Ganzzahl enthält, die die Tragzeit (in Tagen) darstellt, und einen Konstruktor, der die Initialisierung dieses Werts ermöglicht und der als Standardkonstruktor (mit einem Wert Ihrer Wahl) verwendet werden kann.

(b) Definieren Sie nun eine `Ovipare`-Klasse mit einem Attribut, das die Anzahl der Eier pro Gelege darstellt, und einen geeigneten Konstruktor.

(c) Fügen Sie zu jeder Klasse eine Methode `geburt` hinzu, die Folgendes anzeigt: `„Nach X Schwangerschaftstagen habe ich gerade ein neues Baby zur Welt gebracht.“` im Fall eines Lebendgebärenden (Viviparen) und `„Ich habe gerade etwa X Ei(er) gelegt.“` im Falle eines Oviparen, wobei X dem Wert des Attributs entspricht.

(d) Definieren Sie nun die Ovovivipare-Klasse so, dass sie sowohl von Vivipare (zuerst) als auch von Ovipare erbt. Fügen Sie dieser Klasse ein boolesches Attribut hinzu, das angibt, ob die Art selten ist oder nicht. Fügen Sie außerdem einen Konstruktor hinzu, der eine Tragzeit, eine Anzahl von Eiern und einen booleschen Wert (standardmäßig falsch) angibt, der die Seltenheit der Art angibt.

(e) Lassen Sie die Konstruktoren der Klassen `Vivipare` und `Ovipare` eine Meldung anzeigen, um die Reihenfolge der Aufrufe zu beobachten, und erstellen Sie eine `main()`, die eine Instanz von `Ovovivipare` enthält. In welcher Reihenfolge werden die Konstruktoren aufgerufen? Ändern Sie die Reihenfolge der Konstruktoren im `Ovovivipare`-Konstruktor. Kompilieren Sie Ihr Programm neu und starten Sie es neu. Ändert sich dadurch die Aufrufreihenfolge?

(f) Rufen Sie nun die Methode `geburt` Ihrer Instanz auf. Kompilieren und testen Sie Ihr Programm. Was ist passiert? Korrigieren Sie das Programm so, dass die `Vivipare`-Geburtsmethode aufgerufen wird. Kompilieren Sie Ihr Programm neu und starten Sie es neu. Ändern Sie abschließend das Programm so, dass bei der ovoviviparen Geburtsmethode `„Nach X Schwangerschaftstagen habe ich gerade Y Baby(s) zur Welt gebracht“` angezeigt wird. Dabei ist X die Tragzeit und Y die Anzahl der Eier.

(g) Fügen Sie dem Programm eine Tier-Klasse hinzu, die nur aus einem Konstruktor (Standardkonstruktor) und einem Destruktor besteht, die jeweils eine Meldung anzeigen. Lassen Sie Vivipare und Ovipare von Tier erben. Kompilieren Sie Ihr Programm neu und führen Sie es aus. Was passiert? Um dieses Verhalten zu vermeiden, ändern Sie die Tier-Klasse zu einer virtuellen Klasse. Kompilieren Sie Ihr Programm

erneut und führen Sie es aus. Beachten Sie, wann der Konstruktor von Tier aufgerufen wird.

AUFGABE 7.2 (Schlachtschiffe) - **Abgabe**

In dieser Aufgabe geht es um ein Schlachtschiffspiel, für welches Sie Schiffe auf sehr einfache Weise modellieren. Es kann sich um Piraten- oder Handelsschiffe oder um Verräterschiffe handeln, die sowohl Piraten als auch Handelsschiffe sein können.

Laden Sie die Datei ueb_7_aufg_2_schiffe.cpp herunter und vervollständigen Sie Ihr Programm im genkennzeichneten Bereich. Beachten Sie, dass Ihr Code gut gekapselt sein muss und jegliche Codeduplizierung vermieden werden soll.

(a) Ergänzen Sie folgendes nach der Klasse Schiff:

- den Operator += für die Klasse `Koordinaten`. Dieser fügt Koordinaten hinzu (nützlich zum Bewegen von Schiffen); (x,y) ist das Koordinatenpaar, das als Parameter an diesen Operator übergeben wird; += fügt x und y jeweils zur x- und y-Koordinate dieser Koordinate hinzu;
- die Definition einer Funktion namens `entfernung`, die (als double) den Abstand zwischen zwei Koordinaten zurückgibt; der Abstand zwischen zwei Koordinaten (x1; y1) und (x2; y2) wird nach der folgenden Formel berechnet:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

- eine Überladung des Anzeigeoperators << für die Klasse `Koordinaten`; Das Anzeigeformat ist: „(<x>, <y>)“, wobei <x> der Wert der ersten Koordinate und <y> der Wert der zweiten ist; zum Beispiel: (1, 2)
- eine Überladung des Anzeigeoperators << für den Typ `Flagge`; der angezeigte Text soll sein:
  - „Piraten“ für `JollyRogers`;
  - „Frankreich“ für `CompagnieDuSenegal`;
  - „Österreich“ für `CompagnieDOstende`;
  - und „Unbekannt“ für alle anderen;
- eine Überladung des Anzeigeoperators << für den Typ `Zustand` ; der angezeigte Text soll sein:
  - „intakt“ für den Wert `Intakt`;
  - „beschädigt“ für den Wert `Beschaedigt`;
  - „versunken“ für den Wert `Versunken`;
  - und „unbekannter Zustand“ für alle anderen Werte.

(b) Ein Schiff ist durch folgendes gekennzeichnet:

- ein Koordinatenpaar (verwenden Sie die bereitgestellte Klasse), die es ermöglicht, die Position im Koordinaten-Raster abzurufen; benennen Sie dieses Attribut `position_` ;
- eine Flagge (vom Typ Flagge) mit dem Namen flagge_, die angibt, zu welcher Reederei das Schiff gehört;

- eine Information namens „zustand_“, die den Zustand des Schiffs angibt (verwenden Sie den bereitgestellten Typ „Zustand“);

Ein Schiff enthält:

- einen Konstruktor, der mit der bereitgestellten `main` kompatibel ist und die `Koordinaten` und die `Flagge` des Schiffes mithilfe von als Parameter übergebenen Werten initialisiert; das „konstruierte“ Schiff ist intakt;
- eine Methode `position`, die die Koordinaten des `Schiffes` ausgibt;
- eine Methode `void bewegen(int einheiten_x, int einheiten_y)`, die das Schiff auf der Horizontalen und Vertikalten bewegen, es sei denn, es ist versunken; `einheiten_x` und `einheiten_y` können negative Werte haben; benutzen Sie den operator+=, um `bewegen` zu implementieren;
- eine Methode `void aufsteigen()`, welche das Schiff wieder in den Zustand intakt setzt;
- eine Methode `anzeigen`, welche das Schiff folgenderweise anzeigt: <Typ des Schiffes (z.B. Piratenschiff, Handelsschiff, Verräterschiff)> an (<x>, <y>), mit Flagge <Flagge>, <Zustand>

Stellen Sie außerdem eine Funktion `entfernung` bereit, die den Abstand zwischen zwei Schiffen zurückgibt, und eine Überladung des <<-Operators für die Schiffe (was die gleiche Ausgabe wie `anzeigen` erzeugt).

Fügen Sie der Klasse Schiff außerdem ein konstantes Klassendatenelement namens `radius_zusammentreffen` mit dem Wert 10 hinzu.

(c) Ein Schiff kann ein `Piratenschiff`, ein `Handelsschiff` oder ein `Verräterschiff` (das sowohl Handelsschiff als auch Piratenschiff sein kann) sein. Diese Klassen verfügen über Konstruktoren, die mit der bereitgestellten `main` kompatibel sind.
Das Verräterschiff muss einen einzigen Zustand, eine einzige Position und eine einzige Flagge haben, wobei so viel Codeduplizierung wie möglich vermieden werden soll.
Die Klassen `Piratenschiff` und `Handelsschiff` können mit dem Teil der `main` getestet werden, der direkt nach `//Test Teil 1` liegt (siehe Ausführungsbeispiel).

Um das Schlachtschiffspiel zu simulieren, gelten folgende Spielregeln für Begegnungen: wenn zwei Schiffe (die nicht versunken sind) unterschiedliche Flaggen haben und der Abstand, der sie trennt, kleiner als `Schiff::radius_zusammentreffen` ist, stehen sie einander gegenüber, wenn nicht, passiert nichts.
Stellen Sie Ihren Klassen des Weiteren Folgendes zur Verfügung:

- eine Methode `angreifen()`, die ein anderes Schiff als Parameter verwendet und beschreibt, wie ein Schiff ein anderes angreift;
- eine Methode `wehren()`, die ein anderes Schiff als Parameter verwendet und beschreibt, wie ein Schiff reagiert, wenn es von einem anderen Schiff angegriffen wird;
- eine Methode `getroffen()`, ohne Parameter, die beschreibt, was passiert, wenn ein Schiff getroffen wird;
- und eine Methode begegnen(), die ein anderes Schiff als Parameter hat und die Begegnung mit diesem Schiff abwickelt. Diese Methode testet, ob die

Bedingungen für das Auftreten einer Konfrontation erfüllt sind. Wenn ja, wird die angreifen()-Methode aufgerufen, dann die `wehren()`-Methode.

Die Methoden `angreifen()`, wehren() und `getroffen()` können nicht konkret in der Klasse Schiff definiert werden. Die Definition dieser Methoden muss jedoch in allen speziellen Schiffsversionen vorgeschrieben werden.
Die Konfrontationsregeln für Spezialschiffe sind wie folgt:

1. Wenn ein Piratenschiff s1 ein anderes Schiff s2 angreift, „schreit es“ (= Anzeige) „Einsteigen!“; s2 wird dann getroffen (Methode getroffen());
2. Wenn ein Piratenschiff s1 auf den Angriff eines anderen Schiffs (s2) reagiert, unternimmt es nichts, wenn es als versenkt markiert ist; andernfalls wird angezeigt: „Wir werden angegriffen, lasst uns zurückschlagen!“ und greift dann s2 an;
3. Wenn ein Piratenschiff getroffen wird, wird sein Zustand um eine Stufe verringert, d.h. es wechselt von intakt zu beschädigt oder entsprechend von beschädigt zu versenkt.
4. Wenn ein Handelsschiff s1 ein anderes Schiff s2 angreift, ruft es „Wir kriegen euch! (Beleidigungen)“ und sonst passiert nichts;
5. Wenn ein Handelsschiff s1 auf den Angriff eines anderen Schiffs s2 reagiert, sendet es „SOS ich sinke!“, im Falle das es gerade sinkt (gerade als versinkend markiert wurde). Falls es bereits versunken ist, ruft es „Nicht einmal Angst!“;
6. Wenn ein Handelsschiff getroffen wird, sinkt es;
7. Ein Verräterschiff:
    - greift an und wird wie ein Piratenschiff getroffen;
    - Reagiert wie ein Händlerschiff;

Mit dem Abschnitt der main() direkt nach `//Test Teil 2` können verschiedene Begegnungsszenarien getestet werden (siehe Ausführungsbeispiel).

Ausführungsbeispiel:

```
===== Test Teil 1 =====
Piratanschiff an (0, 0) mit Flagge Piraten, intakt
Handelsschiff an (25, 0) mit Flagge Frankreich, intakt
Entfernung:  25
Die Schiffe bewegen sich...
hoch und nach rechts:
Piratenschiff an (75, 10) mit Flagge Piraten, intakt
nach unten:
Piratenschiff an (75, 5) mit Flagge Piraten, intakt
===== Test Teil 2 =====
Verfeindetes Piratenschiff und Handelsschiff (zu weit auseinander):
Vor dem Zusammentreffen:
Piratenschiff an (75, 5) mit Flagge Piraten, intakt
Handelsschiff an (25, 0) mit Flagge Frankreich, intakt
Entfernung:  50.2494
```

```
Nach dem Zusammentreffen:
Piratenschiff an (75, 5) mit Flagge Piraten, intakt
Handelsschiff an (25, 0) mit Flagge Frankreich, intakt
Verfeindetes Piratenschiff und Handelsschiff (nah) :
Vor dem Zusammentreffen:
Piratenschiff an (35, 3) mit Flagge Piraten, intakt
Handelsschiff an (35, 2) mit Flagge Frankreich, intakt
Entfernung:  1
Einsteigen!
SOS ich sinke!
Nach dem Zusammentreffen:
Piratenschiff an (35, 3) mit Flagge Piraten, intakt
Handelsschiff an (35, 2) mit Flagge Frankreich, versunken
Zwei verfeindete Piratenschiffen sind intakt (nah):
Vor dem Zusammentreffen:
Piratenschiff an (35, 3) mit Flagge Piraten, intakt
Piratenschiff an (33, 8) mit Flagge Österreich, intakt
Entfernung:  5.38516
Einsteigen!
Nein, sie greifen uns an! Wir wehren uns!!
Einsteigen!
Nach dem Zusammentreffen:
Piratenschiff an (35, 3) mit Flagge Piraten, beschädigt
Piratenschiff an (33, 8) mit Flagge Österreich, beschädigt
Beschaedigte Piratenschiffe, verfeindet :
Vor dem Zusammentreffen:
Piratenschiff an (35, 3) mit Flagge Piraten, beschädigt
Piratenschiff an (33, 8) mit Flagge Österreich, beschädigt
Entfernung:  5.38516
Einsteigen!
Nach dem Zusammentreffen:
Piratenschiff an (35, 3) mit Flagge Piraten, beschädigt
Piratenschiff an (33, 8) mit Flagge Österreich, versunken
Verfeindete Handelsschiffe:
Vor dem Zusammentreffen:
Handelsschiff an (21, 7) mit Flagge Frankreich, intakt
Handelsschiff an (27, 2) mit Flagge Österreich, intakt
Entfernung:  7.81025
Wir kriegen euch! (Beleidigungen)
Nicht einmal Angst!
Nach dem Zusammentreffen:
Handelsschiff an (21, 7) mit Flagge Frankreich, intakt
Handelsschiff an (27, 2) mit Flagge Österreich, intakt
Piratenschiff vs. Verräterschiff:
Vor dem Zusammentreffen:
Piratenschiff an (33, 8) mit Flagge Österreich, intakt
Verräterschiff an (32, 10) mit Flagge Frankreich, intakt
Entfernung:  2.23607
Einsteigen!
Nicht einmal Angst!
Nach dem Zusammentreffen:
Piratenschiff an (33, 8) mit Flagge Österreich, intakt
```

```
Verräterschiff an (32, 10) mit Flagge Frankreich, beschädigt
Verraeterschiff vs. Piratenschiff:
Vor dem Zusammentreffen:
Verräterschiff an (32, 10) mit Flagge Frankreich, beschädigt
Piratenschiff an (33, 8) mit Flagge Österreich, intakt
Entfernung:  2.23607
Einsteigen!
Nein, sie greifen uns an! Wir wehren uns!!
Einsteigen!
Nach dem Zusammentreffen:
Verräterschiff an (32, 10) mit Flagge Frankreich, versunken
Piratenschiff an (33, 8) mit Flagge Österreich, beschädigt
```

LÖSUNG AUFGABE 7.1 (Zoo)

(a)-(c)

```
class Vivipare {
public:
	Vivipare(unsigned int tage = 128) : tragzeit(tage) {
	}
	void geburt() const {
		cout << "Nach " << tragzeit << " schwangerschaftstagen " << endl
			<< "habe ich gerade ein neues Baby zur Welt gebracht" << endl;
	}
protected:
	unsigned int tragzeit;
};


class Ovipare{
public:
	Ovipare(unsigned int anzahl = 12) : eier(anzahl) {
	}
	void geburt() const {
		cout << "Ich habe gerade etwa " << eier
			<< " Ei(er) gelegt." << endl;
	}
protected:
	unsigned int eier;
};
```

(d)-(f) Mehrfachvererung

```
class Ovovivipare: public Vivipare, public Ovipare {
public:
	Ovovivipare(unsigned int tage, unsigned int anz, bool selten = false)
		: Vivipare(tage), Ovipare(anz), seltene_art(selten)
	{}
	// using Vivipare::geburt;
	void geburt() const {
		cout << "Nach " << tragzeit << " Schwangerschaftstagen"
			<< "habe ich gerade " << eier << " Baby(s) zur Welt gebracht"
```

```
                << endl;
        }
protected:
        bool seltene_art;
};
```

(g) Virtuelle Klassen

```
#include <iostream>
using namespace std;
// ------------------------------------------------------------------------
class Tier {
public:
        Tier() { cout << "Hallihallo, ein Tier mehr" << endl; }
        virtual ~Tier() { cout << "Tschuess..." << endl; }
};
// ------------------------------------------------------------------------
class Vivipare : public virtual Tier {
public:
        Vivipare(unsigned int tage = 128) : tragzeit(tage) {
                cout << "Ich bin ein Vivipare" << endl;
        }
        void geburt() const {
                cout << "Nach " << tragzeit << " schwangerschaftstagen " << endl
                        << "habe ich gerade ein neues Baby zur Welt gebracht" << endl;
        }
protected:
        unsigned int tragzeit;
};

// ------------------------------------------------------------------------
class Ovipare : public virtual Tier {
public:
        Ovipare(unsigned int anzahl = 12) : eier(anzahl) {
                cout << "Ich bin ein Ovipare" << endl;
        }
        void geburt() const {
                cout << "Ich habe gerade etwa " << eier
                        << " Ei(er) gelegt." << endl;
        }
protected:
        unsigned int eier;
};
// ------------------------------------------------------------------------
class Ovovivipare : public Vivipare, public Ovipare {
public:
        Ovovivipare(unsigned int tage, unsigned int anz, bool selten = false)
                : Vivipare(tage), Ovipare(anz), seltene_art(selten)
        {}
        // using Vivipare::geburt;
        void geburt() const {
                cout << "Nach " << tragzeit << " Schwangerschaftstagen"
                        << "habe ich gerade " << eier << " Baby(s) zur Welt gebracht"
                        << endl;
        }
protected:
        bool seltene_art;
```

```
};
// ========================================================================
int main()
{
	Ovovivipare ein_hai(220, 11); //  6 bis 12 Monate und 2 bis 20+ Eier
	ein_hai.geburt();
	return 0;
}
```