

```
        Beispiel einBeispiel(2);
        cout << "    Block wird verlassen\n    ";
    }
    cout << "main wird verlassen\n";
}
```

Die Ausgabe des Programms ist:

```
Objekt 0 wird erzeugt.
main wird begonnen
Objekt 1 wird erzeugt.
    neuer Block
    Objekt 2 wird erzeugt.
    Block wird verlassen
    Objekt 2 wird zerstört.
main wird verlassen
Objekt 1 wird zerstört.
Objekt 0 wird zerstört.
```

Der Destruktor statischer Objekte (static oder globale Objekte) wird nicht nur beim Verlassen eines Programms mit `return`, sondern auch bei Verlassen mit `exit()` aufgerufen. Im Gegensatz zum normalen Verlassen eines Blocks wird der Speicherplatz bei `exit()` jedoch nicht freigegeben.

4.7 Wie kommt man zu Klassen und Objekten? Ein Beispiel

Es kann hier keine allgemeine Methode gezeigt werden, wie man von einer Aufgabe zu Klassen und Objekten kommen kann. Es wird jedoch anhand eines Beispiels ein erster Eindruck vermittelt, wie man von einer Problemstellung zum objektorientierten Programm kommen kann.

Simulation der Benutzung eines einfachen Getränkeautomaten

In der Kantine stehen Getränkeautomaten, unter anderem einer mit *ObjektCola*. Der Automat sei so eingestellt, dass eine Dose dieses Getränks 2 € kostet. Der Automat nimmt einen beliebigen Geldbetrag an. Wenn zu wenig eingeworfen wird, wird das eingeworfene Geld wieder herausgegeben. Wenn zu viel eingeworfen wird, wird eine Dose *ObjektCola* sowie der Restbetrag herausgegeben. Nach Geldeinwurf löst ein Knopfdruck die Prüfung des Geldbetrags und gegebenenfalls die Ausgabe einer Dose aus. Wenn keine Dose mehr vorrätig ist, ist der Automat gesperrt, das heißt, dass jeder eingeworfene Geldbetrag vollständig zurückgegeben wird. Das folgende, bewusst einfach gehaltene Szenario soll mit einem Programm simuliert werden:



Szenario

Der Automat wird anfangs mit 50 Dosen befüllt. Zum Automaten gehen Personen, die eine Dose aus dem Automaten ziehen wollen. Sie tragen unterschiedliche Geldbeträge bei sich. Wenn der Betrag ausreicht, werfen sie Münzen in den Automaten, wobei manche nicht genau zählen und zufällig zu viele oder zu wenige Münzen einwerfen. Anschließend drücken sie auf den Ausgabeknopf mit der Wirkung, dass gegebenenfalls eine Dose und Rückgeld ausgegeben werden. Das Szenario endet, wenn der Automat gesperrt wird, weil er leer ist.

Das Programm soll zur Kontrolle die einzelnen Schritte auf dem Bildschirm dokumentieren. Der Einfachheit halber genügt es, einheitliche Münzen anzunehmen, zum Beispiel 1-€-Stücke.

4.7.1 Einige Analyse-Überlegungen

Um die Problemstellung zu verdeutlichen, wird sie aus verschiedenen Blickwinkeln betrachtet. Es handelt sich dabei nur um *Möglichkeiten*, nicht um den einzig wahren Lösungsansatz (den es nicht gibt).

1. In der Analyse geht es zunächst einmal darum, Prozesse *in der Sprache des (späteren Programm-) Anwenders* zu beschreiben. Dabei sind typische Abläufe, Szenarien genannt, ein gutes Hilfsmittel. Die Aufgabenstellung ist als Szenario formuliert.
2. Im zweiten Schritt wird versucht, beteiligte Objekte, ihr Verhalten und ihr Zusammenwirken zu identifizieren. Dies ist nicht unbedingt einfach, weil spontan identifizierte Beziehungen zwischen Objekten im Programm nicht immer die wesentliche Rolle spielen. Auf einem geeigneten Abstraktionsgrad muss entschieden werden, was zum »System« und was zur »Außenwelt« gehören soll.

Strukturierung des Ablaufs

Das Szenario beschreibt einen *Vorgang*, dessen einzelne Schritte als Pseudocode strukturiert dargestellt werden können.

Anfang des Szenarios

- 01 Automat mit 50 Dosen füllen.
- 02 Solange der Automat nicht gesperrt, das heißt nicht leer ist, wiederhole:
- 03 Eine durstige Person (z.B. eine Studentin) kommt vorbei, sie hat X € dabei.
- 04 X steht für eine zufällige Zahl.
- 05 Falls sie genug Geld hat (d.h. mindestens den Preis pro Dose).
- 06 steckt sie eine Anzahl Y Münzen in den Münzschlitz
- 07 und drückt auf den Knopf.
- 08 Falls Rückgeld ausgegeben wurde,
- 09 nimmt sie es.
- 10 Falls eine Dose ausgegeben wurde,
- 11 nimmt sie sie und
- 12 trinkt sie aus.

- 13 Andernfalls stellt sie fest: »Zu wenig Geld!«.
- 14 Die Person verlässt die Szene
(gegebenenfalls nicht mehr durstig und mit weniger Geld).

Ende des Szenarios

Objekte und Operationen identifizieren

Im nächsten Schritt wird versucht, die beteiligten Objekte und damit ihre Klassen zu identifizieren und eine Beschreibung ihres Verhaltens zu finden. Dazu wird auf die einzelnen nummerierten Zeilen des Szenarios Bezug genommen, die *kursiv* dargestellt sind. Bezeichnungen in dieser Schrift deuten auf *mögliche* Klassen- und Methodennamen. Es ist nicht nur das aktive Verhalten wichtig, sondern manchmal benötigt man außerhalb des Objekts Informationen über seinen inneren Zustand.

01 Automat mit 50 Dosen füllen.

Der Automat ist ein `GetraenkeAutomat`. Er enthält Dosen, deren Anzahl uns interessiert. Die Anzahl ist eine Eigenschaft des Automaten, nicht der Dose. Andere Automaten können gleichartige Dosen in anderer Stückzahl enthalten. Unklar ist hier, *wer* den Automaten füllt. Da wir uns in diesem Szenario nicht dafür interessieren, liegt die Antwort außerhalb unseres »Systems«. Wir müssen nur dafür sorgen, dass der Automat am Anfang des Szenarios ein paar Dosen enthält.

02 Solange der Automat nicht gesperrt, das heißt leer ist, wiederhole:

Der Automat hat ein Attribut `gesperrt`, das anzeigt, dass keine Dosen mehr da sind.

03 Eine Person (z.B eine Studentin) kommt vorbei, sie hat X €.

Anstelle einer Studentin könnte natürlich eine andere Person mit etwas Geld kommen. Die Aktivität hier ist `kommen`.

05 Falls sie genug Geld hat (d.h. mindestens den Preis pro Dose),

Hat sie genug Geld, verglichen mit dem Preis pro Dose, der in dem Automaten eingestellt ist? Die Menge des Geldes ist ein Attribut der Person, der Dosenpreis ein Attribut des Automaten.

06 steckt sie eine Anzahl Y Münzen in den Münzschlitz

Das `GeldEinwerfen` ist eine Aktivität der Studentin. Dabei ist wichtig, wie viele Muenzen in welchen `Getraenkeautomat` (hier gibt es nur einen) gesteckt werden. Der Automat muss die Muenzen akzeptieren und die Anzahl der eingeworfenen Muenzen kennen.

07 und drückt auf den Knopf.

`Knopf druecken` ist eine Aktivität der Studentin, die sich auf einen bestimmten Automaten bezieht. Gleichzeitig ist dies eine Aufforderung an den Automaten, das Geld zu prüfen und eine Dose herauszugeben.

08 Falls Rückgeld ausgegeben wurde,

09 nimmt sie es.

Nur wenn `Rueckgeld` vorhanden ist, kann sie Geld entnehmen. Rückgeld kann nur vorhanden sein, wenn es vorher eine `Geldrueckgabe` gab.

10 Falls eine Dose ausgegeben wurde,

11 nimmt sie sie und

12 trinkt sie aus.

Falls der Automat eine Dose herausgegeben hat, kann die Studentin die Dose entnehmen.

Dann trinkt sie.

13 Andernfalls stellt sie fest: »Zu wenig Geld!«.

Die Studentin sagt....

14 Die Person verlässt die Szene (hoffentlich nicht mehr durstig und mit weniger Geld).

Die Aktivität ist hier verlassen.

Ende des Szenarios

Analyse einiger Aktivitäten

In der objektorientierten Programmierung kommunizieren Objekte durch Botschaften (englisch *message*). Das Wort »Botschaft« ist eine häufig benutzte, aber ungenaue Übersetzung, weil der Aufforderungscharakter unter den Tisch fällt. Besser ist es, nur von Aufforderungen zu sprechen, die an ein Objekt gerichtet sind. Die Notation ist *Objekt.Aufforderung(Daten)*. In den oben beschriebenen Aktivitäten stecken Aufforderungen: zu 06:

Die Daten, die zu der Aktivität `Geld einwerfen` gehören, sind der `Automat`, in den die Münzen gesteckt werden, sowie die Anzahl der Münzen. In der obigen Notation geschrieben: *dieStudentin.GeldEinwerfen(derAutomat, Münzenanzahl)*

Die Aufforderung an den Automaten, die *innerhalb* der Aktivität `Geld einwerfen` steckt, ist es, die Münzen zu akzeptieren. Das Akzeptieren der Münzen beinhaltet die Kenntnis, wie viele Münzen eingeworfen wurden. In der obigen Notation geschrieben: *derAutomat.akzeptieren(Münzenanzahl)*

zu 07:

Ähnliche Überlegungen sind für die Aktivität `Knopf druecken` anzustellen, weil diese Aktivität den Automaten dazu veranlasst, das eingeworfene Geld zu prüfen und eine Dose herauszugeben.

Erster Klassenentwurf

In der Analyse können Objekte identifiziert und damit schon erste Klassen gebildet werden. Die Aktivitäten oder das Verhalten der Objekte sind nach außen sichtbar und erscheinen deshalb im Programm als öffentliche Schnittstelle. Im Design werden die Klassen näher untersucht und genauer formuliert. Insbesondere ergeben sich aus den Aktivitäten, welche Informationen über ein Objekt benötigt werden bzw. welche *Attribute* es hat.

Die Klassen ergeben sich aus den handelnden Objekten mit ihren Attributen und Aktivitäten. Der Ansatz, zunächst die Substantive mit Objekten gleichzusetzen und die Verben mit Methoden, kann irreführend sein, weil beides austauschbar ist. Beispiel: »Die Entnahme der Dose wird von der Studentin durchgeführt.« Entnahme als Objekt? Durchführen als Aktivität? – etwas zweifelhaft. »Die Studentin entnimmt die Dose« ist erheblich klarer. Passivkonstruktionen sollten also vor der Analyse der Substantive und Verben stets in Aktivkonstruktionen verwandelt werden. Ebenso ist das Subjekt genau zu identifizieren.

»Die Messung erfolgt um 13 Uhr.« ist in diesem Sinne ein unbrauchbarer Satz, weil die Frage nach dem »wer?« nicht beantwortet werden kann.

Objekte sind aus dem vorherigen Abschnitt identifizierbar. Es gibt Objekte, die jedoch passiv sind: Dosen und Münzen. Zu diesen Objekten gibt es keine Attribute, es interessiert hier nur die Anzahl. Wenn zu einem Objekt keine Attribute und keine Aktivitäten (passiv!) angebbbar sind, ist es im Allgemeinen nicht notwendig, es als Klasse zu formulieren. Die *aktiven*, handelnden Objekte sind die Studentin und der Automat. Da anstelle der Studentin (Spezialfall) auch andere Personen kommen können, ist es sinnvoll, dafür eine Klasse *Person* zu erfinden. Die Klasse für den Automaten nennen wir *GetraenkeAutomat*. Die *vorläufig* gefundenen Attribute und Aktivitäten sind in Tabelle 4.1 zusammengefasst.

Tabelle 4.1: Vorläufige Kandidaten für Klassen, Attribute und Aktivitäten

Klasse	Attribute	Aktivitäten/Zustandsabfragen
Person	Geldmenge	kommen hat genug Geld? Geld einwerfen Knopf drücken Geld entnehmen Dose entnehmen trinken gehen / Szene verlassen
GetränkeAutomat	Anzahl Dosen gesperrt Preis pro Dose Rückgeld eingeworfene Münzen	befüllt werden (von wem?) Geld prüfen Dose herausgegeben? Rückgeld vorhanden? Münzen akzeptieren Dose herausgeben ist gesperrt? Geldrückgabe

4.7.2 Formulierung des Szenarios in C++

Bei den identifizierten Objekten mit ihren Methoden handelt es sich *zunächst um eine erste Näherung*, die weiter verfeinert werden muss. Weil nur ein erster Eindruck vermittelt werden soll, wird auf eine vollständige objektorientierte Analyse (OOA) und ein entsprechendes Design (OOD) verzichtet und auf die Literatur verwiesen, die die OOA/D-Thematik ausführlich behandelt, zum Beispiel [Oe] oder [Bal05]. Hier wird *als Starthilfe* das oben strukturiert beschriebene Szenario in einer möglichen C++-Darstellung formuliert. Es gibt nur einige Besonderheiten:

- `main()` beschreibt im Ablauf Aktivitäten einer Person, die ihrerseits Aktivitäten beim Getränkeautomaten auslösen.
- Der Automat wird mit Anfangswerten initialisiert (Konstruktor). Damit ist er am Anfang mit Dosen gefüllt, ohne dass wir uns Gedanken machen müssen, wer ihn gefüllt hat.
- Die Methoden `kommen` und `gehen` werden in C++ geeignet durch Konstruktor und Destruktor beschrieben.

Listing 4.10: Simulation des Getränkeautomaten

```

// /cppbuch/loesungen/k4/5_4/main.cpp
#include "person.h"
#include "automat.h"
using namespace std;

int zufall(int x) { // gibt eine Pseudo-Zufallszahl zwischen 0 und x zurück
    static long r = 1;
    r = (125 * r) % 8192;
    return (x+1)*r/8192;
}

int main() {
    const int DOSEANZAHL = 50,
            DOSENPREIS = 2, // in €
            MAX_GELD = 20; // in €

    // Szenario:
    // Der Konstruktor initialisiert den Automaten mit der
    // gewünschten Anzahl von Dosen und dem Dosenpreis.
    GetraenkeAutomat objektColaAutomat(DOSEANZAHL, DOSENPREIS);
    while(!objektColaAutomat.istGesperrt()) {
        // eine Person betritt die Szene:
        // Der Konstruktor erzeugt ein Objekt der Klasse Person (siehe Text) mit Namen
        // einePerson, wobei das Objekt (das heißt sein Geldvorrat) mit einer zufälligen
        // Zahl zwischen 0 und MAX_GELD initialisiert wird.
        Person einePerson(zufall(MAX_GELD));
        if(einePerson.genugGeld( objektColaAutomat.getraenkePreis())) {
            // eine zufällig gewählte Anzahl Münzen wird eingeworfen,
            // aber nicht mehr als vorhanden:
            int wieviel = zufall(einePerson.wievielGeld());
            einePerson.geldEinwerfen(objektColaAutomat, wieviel);
            einePerson.knopfDruecken(objektColaAutomat);
            if(objektColaAutomat.rueckgeldVorhanden())
                einePerson.geldEntnehmen( objektColaAutomat.geldRueckGabe());
            if(objektColaAutomat.doseHerausgegeben()) {
                einePerson.doseEntnehmen(objektColaAutomat);
                einePerson.trinkt();
            }
            if(objektColaAutomat.istGesperrt())
                cout << "Automat gesperrt! (leer)" << endl;
        }
        else einePerson.sagt(" Leider zu wenig Geld.");
    } // Blockende: die Studentin verlässt die Szene:
    // hier automatisch realisiert durch den Destruktor
}

```



Übungen

4.4 Versuchen Sie, die einzelnen Schritte nachzuvollziehen. Stellen Sie fest, wo es Unzulänglichkeiten und Unvollständigkeiten gibt. Entwerfen Sie die nötigen Klassen in C++, vervollständigen Sie das Programm und bringen Sie es zum Laufen. Das Programm sollte verschiedene Fälle abdecken (verschiedene, zufällig gewählte anfängliche Geldmengen),

die in der zu erzeugenden Testdokumentation aufgeführt werden. Die Testdokumentation erhält man am einfachsten durch Umleiten der Bildschirmausgaben in eine Datei.

Falls der Automat einer Methode als Parameter übergeben wird, die ihrerseits eine Methode des Automaten aufruft, welche den Zustand des Automaten ändert, ist nur die Übergabe per Referenz sinnvoll. Änderungen des Zustands des Automaten müssen im Original wirksam werden, nicht in einer Kopie, die am Ende der Methode weggeworfen wird. Dies kann erzwungen werden, wenn man den Kopierkonstruktor privat deklariert. Eine Definition ist nicht erforderlich, weil kein Aufruf erfolgt.

4.5 Wie können Sie mit C++ erreichen, dass ein Attribut *direkt*, also ohne Einsatz einer Methode, zwar gelesen, aber nicht verändert werden kann? Beispiel:

```
int main() {
    MeineKlasse objekt;
    objekt.readonlyAttribut = 999; // Fehler! nicht erlaubte Aktion
    // erlaubter direkter lesender Zugriff:
    cout << "objekt.readonlyAttribut="
         << objekt.readonlyAttribut << endl; // ok
}
```

Wie sieht die Realisierung in der Klasse `MeineKlasse` aus?

4.8 Gegenseitige Abhängigkeit von Klassen

In der Lösung der Aufgabe des vorhergehenden Abschnitts werden in den inline-Funktionen der Klasse `Person` teilweise Methoden der Klasse `GetraenkeAutomat` benutzt, weswegen `automat.h` von `person.h` eingeschlossen wird. Hingegen benutzt die Klasse `GetraenkeAutomat` keinerlei Eigenschaften oder Methoden der Klasse `Person`. Was ist aber, wenn jede der beiden Klassen Methoden der jeweils anderen Klasse benutzt? Es nutzt nichts, gegenseitig die Header-Datei der jeweils anderen Klasse einzuschließen, weil der Compiler die nötigen Informationen nicht bekommt. Betrachten wir zwei Header-Dateien, die sich aufeinander beziehen:

```
// Datei A.h
#ifndef A_h
#define A_h
#include "B.h"
... usw.
#endif
```

```
// Datei B.h
#ifndef B_h
#define B_h
#include "A.h"
... usw.
#endif
```

Wenn `A.h` zuerst gelesen wird, wird bei Ausführung der dritten Zeile `B.h` eingelesen. Die Ausführung der dritten Zeile von `B.h` scheitert jedoch, weil `A_h` nun definiert ist und der Rest von `A.h` nicht zur Kenntnis genommen wird. Die Lösung des Problems besteht in der *Vorwärtsdeklaration*: