Die Simulation einer Tankstellenkasse

Anhand der vereinfachten Simulation einer Tankstellenkasse soll die objektorientierte Programmierung eingeübt und vertieft werden. Dynamisches Erzeugen von Objekten, der Umgang mit einer einfachen sequentiellen Datenbank oder XML Daten für die Waren der Tankstelle, die Verwendung von Namensräumen, objektorientiertes Arbeiten mit Streams, die Ausnahmebehandlung (exception – handling) in C++ sowie das Verteilen von Klassen in UML Packages stellen dabei die Schwerpunkte des Projekts dar.

1. Die Ausgangssituation

Ihre Firma beschäftigt sich mit der hardwarenahen Programmierung von embedded systems vornehmlich in der Programmiersprache C++. Als neuer Auftrag steht die Software für ein Tankstellenkassensystem ins Haus. Da das Programm auch für andere Kassensysteme verwendet werden soll, drängt sich im Hinblick auf die Wiederverwendbarkeit ein objektorientierter Lösungsansatz auf.

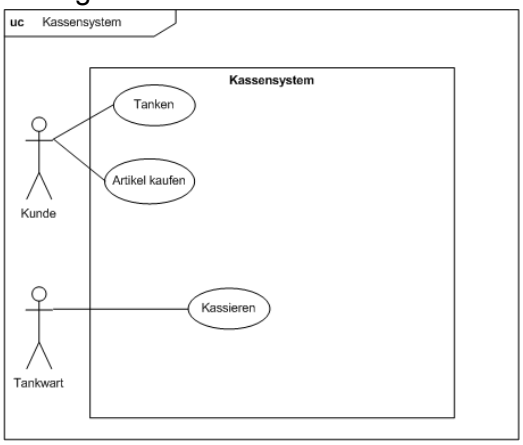
- ☞ **Beschreiben Sie die Vorgänge bei einer Kasse nach dem Tankvorgang!**
- ☞ **Nennen Sie die beiden beteiligten Akteure des Kassensystems!**
- ☞ **Mit welchen grundlegenden Use Cases lässt sich das Kassensystem beschreiben?**

2. Die Problembeschreibung zur Tankstellenkasse

An der Kasse können unterschiedliche Artikel eingegeben werden. Die Artikelnummer von Waren wird über einen Scanner eingelesen, anschließend gibt der Tankwart die gekaufte Menge ein. Hat ein Kunde getankt, so liefert die Zapfsäule mit Tankanzeige die Artikelnummer und Menge des getankten Kraftstoffs an die Kasse. Über die Artikeldatenbank werden die Daten wie Bezeichnung, Nettopreis und Handelsspanne anhand der Artikelnummer an die Kasse gesendet. Jeder eingegebene Artikel wird in eine Liste für die entsprechende Buchung aufgenommen. Am Ende der Buchung wird ein Beleg erstellt. Zur Auswahl der Aktionen verfügt die Kasse über ein Menü.

Die Erzeugung der Objekte und die Steuerung des Programmablaufs übernimmt die Applikation in der Methode Run().

- ☞ **Erstellen Sie ein neues Visio Projekt und benennen Sie es in Kassensystem um!**
- ☞ **Modellieren Sie ein Anwendungsfallmodell der Tankstellenkasse. Verfeinern Sie dabei die grundlegenden Anwendungsfälle des Tankkassensystems!**



3. Analyse und Design

Komplizierte Projekte werden im seltensten Fall mit dem ersten Entwurf gelöst. Trotzdem benötigt das Projekt einen Startpunkt, der mit einer eingehenden Analyse der Aufgabenstellung beginnt.

3.1 Klassen und Objekte:

Im ersten Schritt sind die Aufgabenstellung zu analysieren und eine Liste der Klassen zu erstellen.

- ☞ **Analysieren Sie die Aufgabenstellung des Kassensystems und erstellen Sie eine Liste der möglichen Klassen des Programms unter Visio!**

3.2 Ein erstes statisches Modell:

Im statischem Modell sind die Beziehungen der Klassen zueinander zu modellieren. Auch hier ist es im ersten Entwurfsstadium noch nicht notwendig, alle Beziehungen exakt durchzuplanen. Die Grundzüge des Programms sollten jedoch bereits ersichtlich sein.

Vorüberlegungen:

- Einige Beziehungen sind augenscheinlich (z.B. Menue „is part of“ Kasse)! Beginnen Sie zunächst mit diesen Beziehungen.
- Um die Datenbank möglichst unabhängig von anderen Klassen zu halten kommuniziert sie nur mit der Kasse. Die Datenbank dient als reiner „Datenlieferant“.
- Überdenken Sie die Beziehung zwischen Artikel, Ware und Kraftstoff!
- Die Kasse erstellt die Artikel dynamisch entsprechend den Daten der Datenbank und legt sie in der Buchung ab.
- Je weniger Beziehungen zwischen Klassen bestehen, um so unabhängiger voneinander können sie entwickelt und gewartet werden!

☞ **Setzen Sie die Klassen zueinander in Beziehung! Nutzen Sie dazu die Problembeschreibung!**

3.3 Die Gliederung des Projektes

Um einzelne Teile des Projekts unabhängig voneinander entwickeln und warten zu können, empfiehlt es sich, die Klassen in Pakete (Packages) aufzuteilen. Dies kann unter C++ mit Hilfe von Namensräumen erfolgen. Besonderes Augenmerk verdient in diesem Zusammenhang die Datenbank. Sie sollte unabhängig vom Kassensystem funktionieren, damit später verschiedene Datenbanksysteme genutzt werden können. Ebenso gehören wieder verwendbare Teile des Projekts wie Menü und Applikation in ein eigenes Paket.

☞ **Gliedern Sie die Klassen des Kassensystems in die Pakete Anwendung, Kassensystem und DB_Treiber!**
☞ **Nutzen Sie dazu das Arbeitsblatt „Die Symbolik der UML (4)“!**
☞ **Stellen Sie die Beziehungen zwischen den Paketen dar!**

Zur Verstärkung der Unabhängigkeit der Datenbank vom eigentlichen Kassensystem kann in einem weiteren Schritt der Zugriff auf die Datenbank über ein Interface umgesetzt werden. Verwendet der Kunde später unterschiedliche Datenbankmanagementsysteme, so muss nur ein neuer Datenbanktreiber programmiert werden. Das Kassenprogramm ist davon nicht betroffen.

☞ **Fügen Sie dem Paket DB_Treiber das Interface IArtDatenBank hinzu!**

4. Die Artikeldatenbank und das Interface

4.1 Ein Puffer für Artikeldaten

Die Datenbank dient als reiner Datenlieferant. Dazu muss sie die Datensätze für die Artikel aus einer Datei auslesen.

☞ **Bilden sie Teams zu den Themengebieten „Sequentielle Datenspeicherung“ und „XML Datenspeicherung“!**
☞ **Bearbeiten Sie die Informationsblätter!**
☞ **Präsentieren Sie Ihre Ergebnisse vor den anderen Teams!**

Da vom Kassensystem aus in der Regel auf einen ganzen Datensatz zugegriffen wird, benötigt die sequenzielle Datenbank einen Zwischenspeicher im Programm, in den der gerade angeforderte Datensatz eingelesen wird. Dazu bietet sich eine C++ Structure an:

```
struct DatenSatz{
        long artNummer;
        char bez[15];

        _____________________________

        _____________________________
}ihrDatensatz;
```

☞ **Ergänzen Sie die C++ Structure!**

☞ **Ergänzen Sie das Klassendiagramm um die Structure (Hilfsklasse) DatenSatz! Die Structure ist im Paket DB_Treiber abzulegen!**

4.2 Die Klasse und das Interface

Mit der Methode bool SucheDatenSatz(long no) wird der entsprechende Datensatz in die Structure geladen. Anschießend können die Einzeldaten mit long GetArtNummer( ), double GetHSP( ), double GetNetto( ) und char* GetBezeichnung( ) abgefragt werden. SucheDatenSatz(…) liefert true, wenn ein Datensatz mit der entsprechenden Artikelnummer (no) in der Datei vorhanden ist.

Um in C++ eine Datei zu öffnen benötigt man ein Stream Objekt vom Typ ifstream (Input File Stream). Die Datenbankklasse sollte als Eigenschaft daher über ihre Datei (ifstream ihreDaten) verfügen.
In der Methode void OeffneDatenQuelle( ) wird der entsprechende Stream geöffnet.

☞ **Überlegen Sie, welche fünf Methoden im Interface deklariert werden müssen!**
☞ **Ergänzen Sie die Klassendiagramme für IArtDatenBank und ArtDatenBank!**

4.3 Die Realisierung des Interface

Für die Implementierung eines Interface besteht in C++ anders als in neueren Programmiersprachen kein besonderes Schlüsselwort. Die Regeln für Interfaces sind jedoch die gleichen:

- alle Methoden sind public,
- alle Methoden sind rein virtuell (abstrakt) ohne Implementierung.

☞ **Erzeugen Sie im Visual Studio das neue leere C++ Konsolenprojekt Tankkasse!**
☞ **Implementieren Sie das Interface IArtDatenBank in der Headerdatei IArtDatenBank.h!**
☞ **Vergessen Sie auch nicht den Namensraum zu erstellen.**

Um einen Namensraum zu erstellen verwendet man das Schlüsselwort *namespace*.
Nach der Bezeichnung des Namespaces gehören alle Elemente im folgenden Klammernblock zum entsprechenden Namensraum.

```
// Interface zur Artikeldatenbank
namespace DB_Treiber
{
        class IArtDatenBank
        {
                …
        };
}
```

4.4 Die Realisierung der Datenbank

☞ **Programmieren Sie die Klassendefinition der ArtDatenBank!**

Binden Sie die **Headerdatei** <fstream> für das Objekt vom Typ ifstream ein!
Alle Zugriffsmethoden der Datenbank sind öffentlich.
Vergessen Sie nicht, den Konstruktor, Destruktor und den Schutz vor Mehrfacheinbindung und den Namensraum zu implementieren!

```
public:
...
        // Methoden
        long  GetArtNummer( );
        ...
private:
        ifstream ihreDaten;
        struct DatenSatz{
        {
                long artNummer;
                ...
        }ihrDatensatz;
```

Im **Quelltext** (ArtDatenBank.cpp) der Datenbank sind zuerst die vier Zugriffsfunktionen zu erstellen, aber das können Sie bereits selbständig! Vergessen Sie nicht den Namensraum!

```
char* ArtDatenBank::GetBezeichnung( )
{
        return ihrDatensatz.bez;
}
```

In **OeffneDatenQuelle( )** muss zunächst die Datei der Datenbank geöffnet werden. Die Testdatenbank kann im csv oder xml-Format vorliegen (z.B. K:\Daten\Daten.dat oder Daten.xml). Öffnen Sie die Datei im Lesemodus.

Ist die Datei nicht vorhanden, so erfolgt der Programmabbruch mit der Funktion exit(-1). Diese ist in <cstdlib> definiert und beendet das Programm!

```
void ArtDatenBank::OeffneDatenQuelle( )
{
        // Datendatei zum lesen öffnen
        ihreDaten.open("K:\\...", ios_base::in);
        // bei Fehler Abbruch!
        if(!ihreDaten)
        {
                cout << "\n\aFehler ...
                exit(-1);
        }
}
```

Im **Konstruktor** der Datenbank ruft man die Methode OeffneDatenQuelle( ) auf. Somit ist die ArtDatenBank ab ihrer Erzeugung mit der Datei verbunden.
Im **Destruktor** wird die Datei mit der Methode close( ) des ifstream Objekts geschlossen.

```
ArtDatenBank::ArtDatenBank( )
{
    OeffneDatenQuelle( );
}
```

In der Methode **SucheDatenSatz( )** ist der richtige Datensatz zu suchen. Wird er gefunden, so müssen die Einzeldaten in die Structure geladen werden. Die Funktion liefert true zurück.
Wird kein Datensatz mit der übergebenen Artikelnummer gefunden, so wird false zurückgegeben.

☞ **Für die komplexe Methode werden wieder zwei Teams eingeteilt!**
- **Team 1 liest die Daten über die CSV Datei,**
- **Team 2 arbeitet mit der XML Datei.**

☞ **Erstellen Sie die Methode SucheDatenSatz(…) anhand der Hilfestellung!**
- **Ein Gruppensprecher stellt im Anschluss daran den Quellcode vor!**

4.5 Der Testlauf der Version 1.0 in der Applikation

Zum Testlauf der Datenbank soll die Klasse Applikation implementiert werden. Da es nicht erforderlich sein muss, erst ein Applikationsobjekt zu erzeugen um das Programm zu starten soll die Run( ) Methode statisch deklariert werden.

4.5.1 Die Klasse Applikation

Statische Methoden oder Attribute einer Klasse beziehen sich nicht auf bestimmte Objekte. Sie gelten für alle Objekte dieser Klasse. Da sie sich nicht auf ein Objekt beziehen, werden sie in C++ über den Klassennamen und einen zweifachen Doppelpunkt aufgerufen. Ein Objekt ist zum Ausführen nicht erforderlich. In der UML stellt man statische Elemente einer Klasse durch das Unterstreichen dar. Für statische Methoden gelten folgende Regeln:

- Sie dürfen nicht auf Objektdaten zugreifen,
- in der Methode kann kein this Pointer verwendet werden.

Damit auch sichergestellt ist, dass kein Objekt vom Typ Applikation erstellt werden kann, verwendet man einen einfachen Trick: Der Konstruktor der Applikation wird private deklariert. Kein anderes Objekt im Programm kann nun eine Applikation instanziieren.

☞ **Fügen Sie dem Visio Klassendiagramm die Klasse Applikation hinzu!**
☞ **Implementieren Sie die Klasse Applikation sowie die Methode Run( ) mit leerem Methodenrumpf!**

4.5.2 Die Implementierung der Methode Run( )

Der Test der Datenbank erfolgt über die Methode Run( ) der Applikation. Hier soll nach einem bestimmten Artikel gesucht und dessen Daten ausgegeben werden.

Zum Benutzen der Datenbank sind natürlich die Headerdateien sowie der Namensraum der Datenbankklassen einzubinden.

Nach einem Begrüßungsbildschirm erzeugt man polymorph über das Interface einen Zeiger auf die Datenbank im Heap.

Anschließend sucht man einen bestimmten Artikel und lässt alle Daten zum Artikel ausgeben.
Vergessen Sie nicht, die Datenbank im Heap wieder zu löschen!

```
...
using namespace DB_Treiber;
...
void Applikation::Run( )
{
 // Begrüßungsbildschirm:
       cout << ...
 // ArtDatenBank erzeugen
 IArtDatenBank* dbank = new ArtDatenBank;
 // wenn Artikel gefunden...
 if(dbank->SucheDatenSatz(200102))
 {
   cout << dbank->GetArtNummer( ) << endl
          ...;
 }
 ...
}
```

☞ **Erstellen Sie eine neue Quelltextdatei namens Hauptprogramm.cpp mit einer main( ) Funktion!**
☞ **Testen Sie die Funktion Ihrer Datenbank mit der Methode Run( ) der Applikation!**