Hochschule Aalen
Fakultät Elektronik und Informatik Donnerstag, 10. Februar 2011

Klausur

Objektorientierte Modellierung

WS 2010/2011

Prüfer: Prof. Dr. R. Dietrich

Aufgabe 1. *Modellierung für eine Hochzeitsagentur* (*60 Punkte*)

Eine ***Hochzeitsagentur*** unterstützt Hochzeitspaare beim Erstellen von Geschenklisten. Personen, die ein Hochzeitspaar beschenken möchten, können ein Geschenk von der Geschenkliste des Paares bestellen. Das Geschenk wird dann von der Agentur besorgt. Diese Geschäftsprozesse sollen durch ein Softwaresystem unterstützt werden. Das System soll von Agentur-Mitarbeitern, von Hochzeitspaaren und von Personen, die ein Geschenk bestellen möchten („Käufer"), bedient werden können.

Abbildung 1 zeigt die vorgesehenen Anwendungsfälle des Systems als Anwendungsfalldiagramm, auf der nächsten Seite sind sie beschrieben.

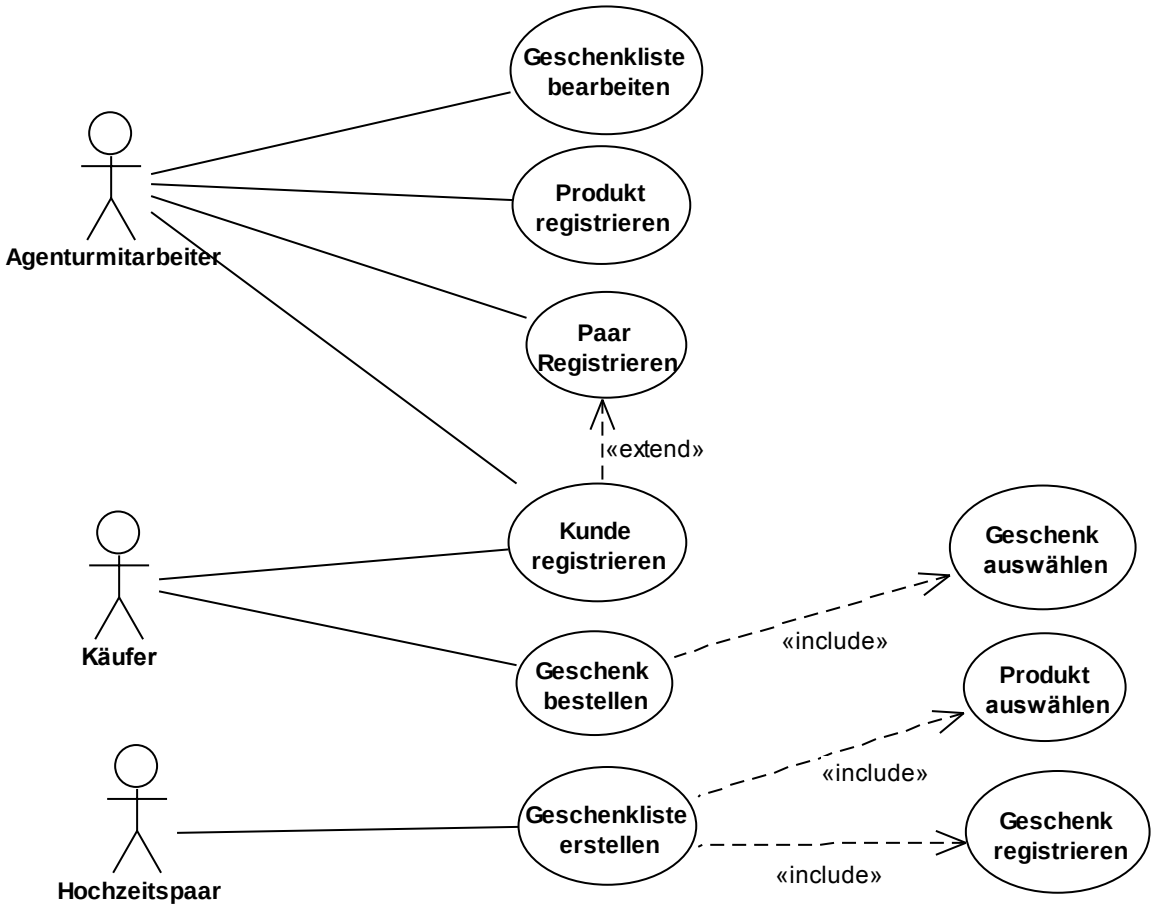

Abbildung 1: Anwendungsfälle des Hochzeitsagentur-Systems.

Die **Anwendungsfälle** des Hochzeitsagentur-Systems – informelle Beschreibung.

Produkt registrieren: Die Agentur-Mitarbeiter können Produkte registrieren, die dann von Hochzeitspaaren für ihre Geschenklisten ausgewählt werden können. Zu einem Produkt muss ein Produktname, eine eindeutige Produktnummer und der Preis eingegeben werden sowie der Name des jeweilige Lieferanten. Zusätzlich kann zu einem Produkt der Name eines Herstellers registriert werden.

Paar registrieren: Paare, die eine Geschenkliste erstellen möchten, müssen zuerst von einem Agentur-Mitarbeiter registriert werden. Die beiden Partner eines Paares werden als Kunden registriert, sofern sie es nicht schon sind (weil sie z.B. früher schon einmal ein Geschenk für ein anderes Paar bestellt hatten). Darüber hinaus werden das Hochzeitsdatum und die Adresse (Straße, PLZ, Ort), wo die Hochzeit stattfindet und zu dem evtl. Geschenke geliefert werden müssen, registriert.

Kunde registrieren: Als Kunden werden außer den Hochzeitspaaren auch die Käufer von Geschenken registriert. Diese müssen sich selbst registrieren, bevor Sie Geschenke bestellen können. Dabei müssen Namen und Anschrift des Kunden angegeben werden.

Geschenkliste erstellen: Sobald ein Hochzeitspaar registriert ist, kann es eine Geschenkliste erstellen. Eine Geschenkliste enthält beliebig viele Geschenke, jedes bezieht sich auf ein bestimmtes, in der Agentur registriertes Produkt. Die Geschenke der Geschenkliste haben eine Positionsnummer und einen Status. Dieser ist zunächst *erfasst*. Solange das Geschenk diesen Status hat, kann es auch wieder von der Geschenkliste gelöscht werden.

Geschenk bestellen: Alle registrierten Kunden können Geschenke von der Geschenkliste eines Hochzeitspaares bestellen. Hierzu muss ein Geschenk von der Geschenkliste eines Paares ausgewählt und die Bestellung des Geschenks veranlasst werden. Für eine Bestellung werden der Kunde, der bestellt, die Geschenkliste, auf die sie sich bezieht, das bestellte Geschenk und das Bestelldatum registriert sowie eine automatisch generierte, eindeutige Bestellnummer. Der Status des Geschenks ist danach *bestellt*.

Geschenkliste bearbeiten: Zwei Wochen vor dem Hochzeitstermin beginnen Agenturmitarbeiter die Geschenkliste eines Hochzeitspaares zu bearbeiten. Jedes Geschenk wird beim Lieferanten oder direkt beim Hersteller angefordert (der Status ist dann *angefordert*). Nach Lieferung hat das Geschenk den Status *geliefert*. Am Hochzeitstag werden alle Geschenke zur Hochzeitsadresse gebracht und der Status wird auf *abgegeben* gesetzt.

(1.1) *Statisches Analysemodell* (*Klassendiagramm*) (*45 Punkte*)

Erstellen Sie ein statisches Analysemodell (Klassendiagramm mit Klassen, Attributen, und Beziehungen – keine Operationen), welches die Objekte und ihre Beziehungen beschreibt, die zur Realisierung der beschriebenen Anwendungsfälle notwendig sind.

Attribute sind möglichst genau zu spezifizieren (Attributtypen, gegebenenfalls Multiplizitäten und Eigenschaftswerte).

Verwenden Sie die vordefinierten UML-Datentypen. Für die Darstellung von Adressen steht ein Datentyp `Adresse` (siehe Abbildung 2) zur Verfügung. Falls Sie weitere Datentypen benötigen, sind diese zu definieren.

Für Assoziationen sind die Multiplizitäten auf beiden Seiten möglichst genau anzugeben.

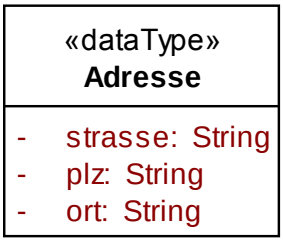

Abbildung 2: ein Datentyp für Adressen

(1.2) *Sequenzdiagramm* (*15 Punkte*)

Der Ablauf des Anwendungsfalls *Paar registrieren* kann wie folgt beschrieben werden:

Beschreibung:

1. Die Kundendaten des ersten Partners werden erfasst (Name und Adresse).
2. Die Kundendaten des zweiten Partners werden erfasst (Name und Adresse).
3. Ein neues Paar wird registriert (mit den beiden Kunden als Partnern, einem Hochzeitstag und einer Hochzeitsadresse).
4. Eine leere Geschenkliste für das Paar wird erstellt

Alternativen:

1a. Der erste Partner ist bereits erfasst: die Kundendaten werden nur abgerufen

2a. Der zweite Partner ist bereits erfasst: die Kundendaten werden nur abgerufen.

Erstellen Sie ein Sequenzdiagramm für folgendes Szenario:

Ein Mann und eine Frau registrieren sich als Paar. Die Daten des Mannes sind bereits registriert, die Daten der Frau müssen neu erfasst werden.

Beschreiben Sie das Szenario möglichst detailliert. Verwenden Sie als Nachrichten Verwaltungsoperationen wie *erfassen* und *abrufen* sowie Basisoperationen wie *set*- und *get*-Operationen für Attribute, *link*-Operationen und *new*. Das Sequenzdiagramm soll auch darstellen, wann im Ablauf des Szenarios neue Objekte erstellt werden.

Aufgabe 2 *Implementierung von Modellen in C++* (*40 Punkte*)

Abbildung 3 zeigt ein vereinfachtes statisches Entwurfsmodell eines Systems einer Versicherungsgesellschaft, mit dem diese ihre Kunden und Versicherungsverträge verwaltet.

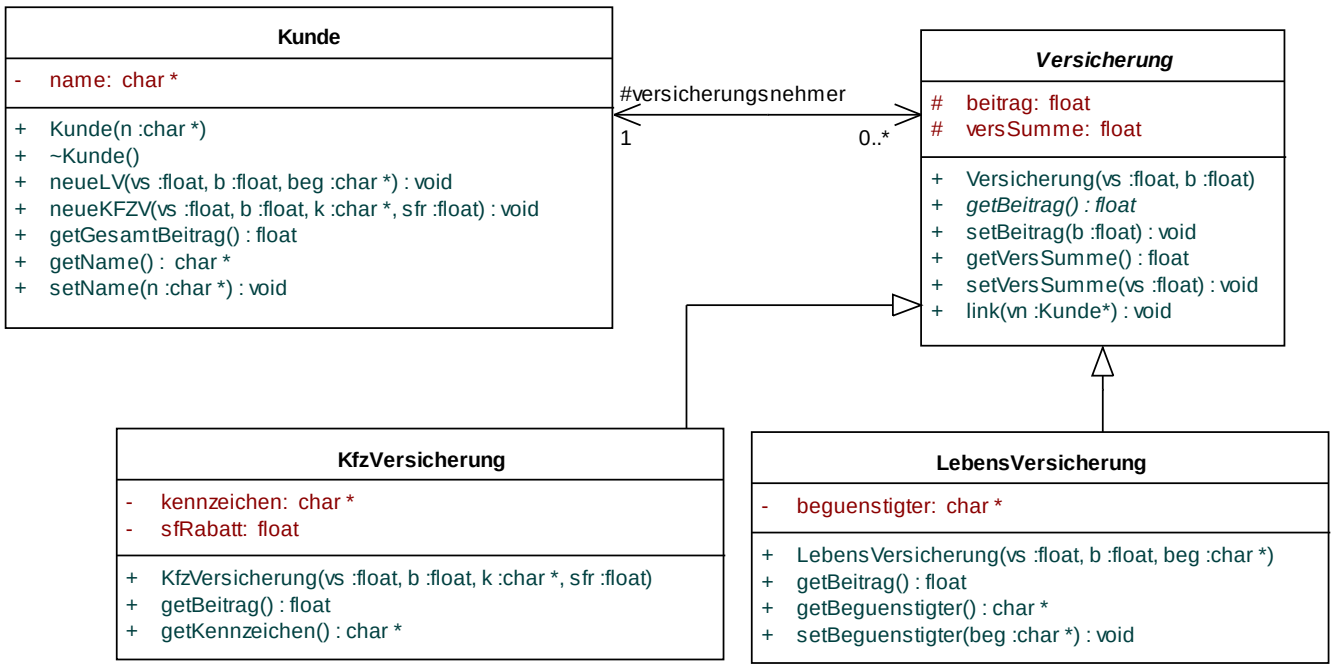

Abbildung 3: Ein einfaches System zur Verwaltung von Kunden und Versicherungen: ein Kunde kann viele Versicherungen abschließen. Bei Versicherungen werden Lebens- und Kfz-Versicherungen unterschieden. Zu allen Versicherungen gibt es eine Versicherungssumme und einen Versicherungsbeitrag (Jahresprämie). Bei Lebensversicherungen gibt es zusätzlich einen Begünstigten, der die Versicherungssumme im Todesfall des Versicherungsnehmers erhält. Bei Kfz-Versicherungen werden das Kennzeichen des versicherten Fahrzeugs und ein Schadens-Freiheits-Rabatt registriert. – Die *set*- und *get*-Methoden für die Attribute haben die übliche Bedeutung. Ausnahme: Bei der Klasse *KfzVersicherung* liefert *getBeitrag()* den Wert des Beitrags multipliziert mit dem Wert des Attributs *sfRabatt.* – Die Konstruktoren für Versicherungen initialisieren alle Attribute mit einem jeweils als Parameter übergebenen Wert. – Die Operation *link* der Klasse Versicherung stellt eine Verknüpfung her von einer Versicherung zu ihrem Versicherungsnehmer.

Folgende Code-Teile (1) bis (3) der Implementierung des Modells in C++ sind vorgegeben:

(1) Die Definition der Klasse *Kunde*

Zur Implementierung der Assoziation von Kunden zu ihren Versicherungen wird die MFC-Klasse `CObList` verwendet. Zeichenketten werden durch nullterminierte `char`-Felder realisiert.

```
class Kunde {
public:

    Kunde(char * n);
    void neueLV(float vs, float b, char * beg);
    void neueKFZV(float vs, float b, char * k, float sfr);
    float getGesamtBeitrag();
    char * getName();
    void setName(char * n);
    ~Kunde();

private:

    // Implementierung der Assoziation: die Versicherungen eines Kunden
    CObList * versicherungen;

    char * name;
};
```

(2) Implementierung der Methode *Kunde::getGesamtBeitrag()*

Die Methode berechnet den gesamten Jahresbeitrag, den ein Kunde für alle seine Versicherungen zu entrichten hat:

```
float Kunde::getGesamtBeitrag(){
    POSITION pos = versicherungen -> GetHeadPosition();
    float summe = 0.0 ;
    while (pos != NULL)
    {
        Versicherung * pV = (Versicherung *)versicherungen -> GetNext(pos);
        summe = summe + pV->getBeitrag();
    }
    return summe;
}
```

(3) Implementierung der Methoden *KfzVersicherung::getBeitrag()* und *LebensVersicherung::getBeitrag()*

```
float LebensVersicherung::getBeitrag()
{ return beitrag; }

float KfzVersicherung::getBeitrag()
{ return sfRabatt * beitrag; }
```

Die folgenden Aufgaben (2.1) bis (2.4) sind so zu lösen, dass sie mit den bisher gemachten Code-Vorgaben und dem Modell in Abbildung 3 konsistent sind.

(2.1) *Klassendefinitionen für Versicherungen* (*25 Punkte*)

Erstellen Sie Klassendefinitionen in C++ für Versicherungen (Klassen *Versicherung*, *LebensVersicherung* und *KfzVersicherung*). Beachten Sie, dass die Assoziation zwischen *Kunde* und *Versicherung* bidirektional zu implementieren ist.

(2.2) *Konstruktoren* (*7 Punkte*)

Definieren Sie die Konstruktoren für die Klassen *Versicherung* und *LebensVersicherung*. Sämtliche Attribute einer Versicherung müssen initialisiert werden. Ein Versicherungsnehmer wird noch nicht festgelegt.

(2.3) *Link-Methode* (*2 Punkte*)

Definieren Sie die Methode *link* der Klasse *Versicherung*. Sie stellt die Verknüpfung von einer Versicherung zu ihrem Versicherungsnehmer (ein Kunde) her.

(2.4) Methode *Kunde::neueLV()* (*6 Punkte*)

Definieren Sie die Methode

```
void Kunde::neueLV(float vs, float b, char * beg);
```

Aufgabe der Methode ist es, den Versicherungen eines Kunden eine neue Lebensversicherung mit der Versicherungssumme `vs`, dem Beitrag `b` und dem Begünstigten `beg` hinzuzufügen.

Alternative für Aufgabe 2: Verwendung von C++-Standard-Klassen

Aufgabe 2 *Implementierung von Modellen in C++* (*40 Punkte*)

Abbildung 3 zeigt ein vereinfachtes statisches Entwurfsmodell eines Systems einer Versicherungsgesellschaft, mit dem diese ihre Kunden und Versicherungsverträge verwaltet.

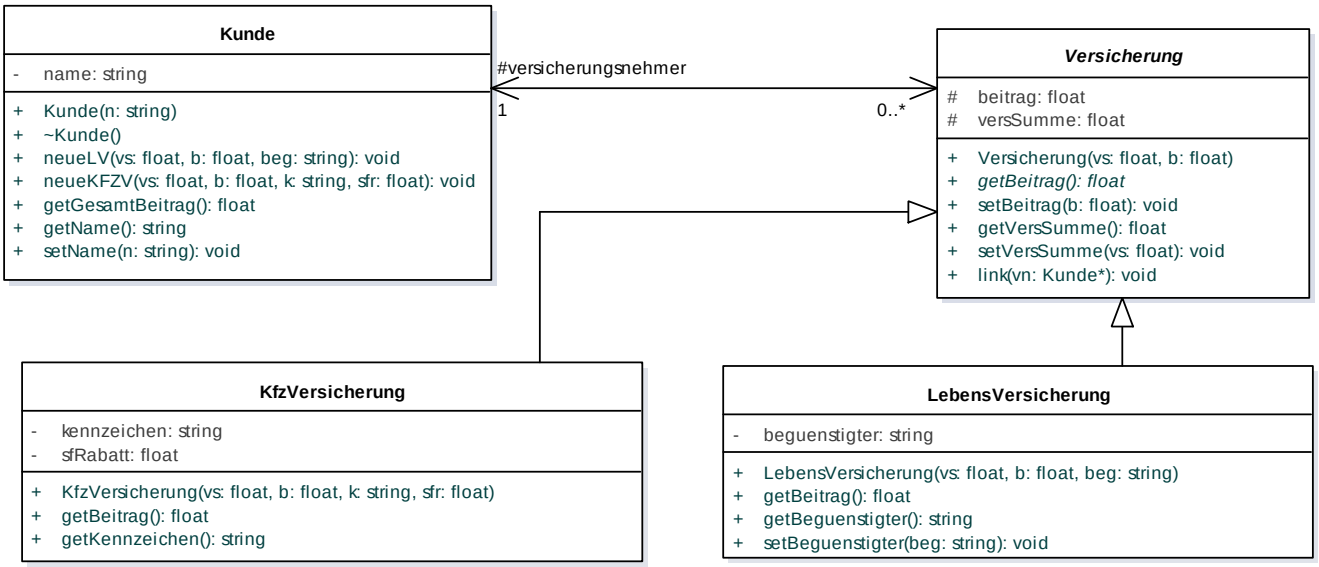

Abbildung 3: Ein einfaches System zur Verwaltung von Kunden und Versicherungen: ein Kunde kann viele Versicherungen abschließen. Bei Versicherungen werden Lebens- und Kfz-Versicherungen unterschieden. Zu allen Versicherungen gibt es eine Versicherungssumme und einen Versicherungsbeitrag (Jahresprämie). Bei Lebensversicherungen gibt es zusätzlich einen Begünstigten, der die Versicherungssumme im Todesfall des Versicherungsnehmers erhält. Bei Kfz-Versicherungen werden das Kennzeichen des versicherten Fahrzeugs und ein Schadens-Freiheits-Rabatt registriert. – Die *set*- und *get*-Methoden für die Attribute haben die übliche Bedeutung. Ausnahme: Bei der Klasse *KfzVersicherung* liefert *getBeitrag()* den Wert des Beitrags multipliziert mit dem Wert des Attributs *sfRabatt.* – Die Konstruktoren für Versicherungen initialisieren alle Attribute mit einem jeweils als Parameter übergebenen Wert. – Die Operation *link* der Klasse Versicherung stellt eine Verknüpfung her von einer Versicherung zu ihrem Versicherungsnehmer.

Folgende Code-Teile (1) bis (3) der Implementierung des Modells in C++ sind vorgegeben:

(1) Die Definition der Klasse *Kunde*

Zur Implementierung der Assoziation von Kunden zu ihren Versicherungen wird die MFC-Klasse `CObList` verwendet. Zeichenketten werden durch nullterminierte `char`-Felder realisiert.

```
class Kunde {
public:

    Kunde(string n);
    void neueLV(float vs, float b, string beg);
    void neueKFZV(float vs, float b, string k, float sfr);
    float getGesamtBeitrag();
    string getName();
    void setName(string n);
    ~Kunde();

private:

    // Implementierung der Assoziation: die Versicherungen eines Kunden
    list<Versicherung*> versicherungen;

    string name;
};
```

(3) Implementierung der Methode *Kunde::getGesamtBeitrag()*

Die Methode berechnet den gesamten Jahresbeitrag, den ein Kunde für alle seine Versicherungen zu entrichten hat:

```
float Kunde::getGesamtBeitrag(){
    list<Versicherung *>::iterator pos = versicherungen.begin();
    float summe = 0.0 ;
    while (pos != versicherungen.end())
    {
       summe = summe + (*pos)->getBeitrag();
    }
    return summe;
}
```

(3) Implementierung der Methoden *KfzVersicherung::getBeitrag()* und *LebensVersicherung::getBeitrag()*

```
float LebensVersicherung::getBeitrag()
{ return beitrag; }

float KfzVersicherung::getBeitrag()
{ return sfRabatt * beitrag; }
```

Die folgenden Aufgaben (2.1) bis (2.4) sind so zu lösen, dass sie mit den bisher gemachten Code-Vorgaben und dem Modell in Abbildung 3 konsistent sind.

(2.1) *Klassendefinitionen für Versicherungen* (*25 Punkte*)

Erstellen Sie Klassendefinitionen in C++ für Versicherungen (Klassen *Versicherung*, *LebensVersicherung* und *KfzVersicherung*). Beachten Sie, dass die Assoziation zwischen *Kunde* und *Versicherung* bidirektional zu implementieren ist.

(2.2) *Konstruktoren* (*7 Punkte*)

Definieren Sie die Konstruktoren für die Klassen *Versicherung* und *LebensVersicherung*. Sämtliche Attribute einer Versicherung müssen initialisiert werden. Ein Versicherungsnehmer wird noch nicht festgelegt.

(2.3) *Link-Methode* (*2 Punkte*)

Definieren Sie die Methode *link* der Klasse *Versicherung*. Sie stellt die Verknüpfung von einer Versicherung zu ihrem Versicherungsnehmer (ein Kunde) her.

(2.4) Methode *Kunde::neueLV()* (*6 Punkte*)

Definieren Sie die Methode

```
void Kunde::neueLV(float vs, float b, string beg);
```

Aufgabe der Methode ist es, den Versicherungen eines Kunden eine neue Lebensversicherung mit der Versicherungssumme `vs`, dem Beitrag `b` und dem Begünstigten `beg` hinzuzufügen.