Jannis Schenk Matrikelnummer: 70480743

Fragen im Rahmen der Studienleistung Objektorientierte Programmierung

Teilstudienleistung 1:

Frage 1: Nenne jeweils fünf Vor- und Nachteile von Objektorientierter Programmierung:

Vorteile:

- Wiederverwendbarkeit von Code: Durch die Erstellung von Klassen, welche als Art Baupläne für Objekte verstanden werden können, ergibt sich daraus das Prinzip der Wiederverwendbarkeit von Code, sodass für die Erzeugung von mehreren gleichartigen Objekten die alle einer spezifischen „Sorte/Art“ zugeordnet werden können, es nur einen Bauplan benötigt, welcher die repetitive Erstellung ermöglicht. Dieser Vorteil kann auch als „Generalisierung“ zusammengefasst werden. Die objektorientierte Softwareentwicklung nutzt generalisierte Klassen und Objekte um gemeinsame Eigenschaften (Attribute) und Funktionen (Methoden) in logischen Einheiten bündeln zu können. Die Vorteile der **Generalisierung** sind somit eine deutliche Reduktion des Programmieraufwands, übersichtliche und verständliche Objektklassen und schnelle Erweiterung von Objekten durch Veränderungen der Objektklassen.
- Datenkapselung: Die Datenkapselung ermöglicht es, Daten, Eigenschaften oder Methoden vor dem Zugriff von außen, beziehungsweise vor ungewollten Zugriff zu schützen. Es kann auch als eine Art **Geheimnisprinzip** verstanden werden und kann als Grundlage von Daten-/Informationssicherheit in den Programmen dienen.
- Polymorphie: Ein weiteres wichtiges vorteilhaftes Konzept innerhalb der objektorientierten Programmierung ist die **Polymorphie**. Die Polymorphie steht dabei stellvertretend dafür, dass Bezeichner abhängig von ihrer jeweiligen Verwendung Objekte unterschiedlicher Datentype annehmen können. Dabei kann Polymorphie beispielsweise durch die Implementierung von überladenen Konstruktoren in Programm-Code realisiert werden.
- Vererbung: Vererbung im Kontext der Programmierung bedeutet, dass sich Objekte von übergeordneten Objekten ableiten können, sprich sie von diesen übergeordneten Objekten/Klassen Funktionalität (Methoden) oder Eigenschaften (Attribute) erben. Als abstraktes Beispiel: Die Klasse Katze kann von einer übergeordneten Klasse Säugetier erben, da eine Katze vermutlich gleiche Funktionalitäten und Eigenschaften aufweist

wie ein Säugetier. **Vererbung** spart somit das Schreiben von Programm-Code und dient zur Erstellung von nachvollziehbaren Programm-Codes.

- Flexibilität von Software: Ein weiterer entscheidender Vorteil ist **die flexible Anpassungsfähigkeit von Software** im Kontext der objektorientierten Programmierung. Klassen können einfach an neu aufkommende Anforderungen angepasst und optimiert werden, ohne große strukturelle Veränderung in der Software vornehmen zu müssen. Die ist ein entscheidender Vorteil, vor allem in der modernen und agilen Softwareentwicklung.

Nachteile:

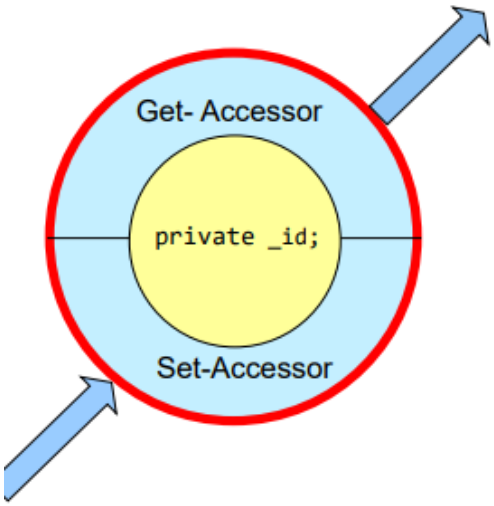
- Ein wesentlicher Nachteil der objektorientierten Programmierung ist die **Nachverfolgbarkeit des Programmflusses** im Gegensatz zur üblichen prozeduralen Programmierung. Dabei kann der Programmfluss zur Laufzeit meistens mithilfe des Stichwortes „Top-Down“ beschrieben werden. Sprich, eine klare Hierarchie ist erkennbar – vor allem bei einer schrittweisen Ausführung ist dies vorteilhaft um eventuell Fehler zu beheben. In der objektorientierten Programmierung ist dies meist schwieriger, da durch die Instanziierung von Objekten und damit verbundenen Sprüngen im Programmablauf eine Nachverfolgbarkeit erschwert wird. Vor Allem bei sehr komplexer Software, welche mehrere Klassen, Beziehungen und Vererbungen besitzt, kann dies problematisch werden.
- Ein zweiter Nachteil ist, dass der **Speicherplatzbedarf** in der Objektorientierten Programmierung im Vergleich recht hoch ist. Durch die Generierung von Objekten wird Speicher benötigt, oftmals werden Objekte aber nur indirekt benötigt, beispielsweise beim Konzept der Vererbung. Dabei werden diese durch die vererbten Konstruktoren aufgerufen, aber im eigentlichen Sinne werden sie nicht direkt verwaltet, sondern laufen im Hintergrund ab.
- Ein weiteres Problem ist die **Performance** von objektorientierter Software, welche häufig ineffizienter ist, im Vergleich zu anderen Programmierkonzepten, wie der prozeduralen Programmierung
- Zum Abschluss lässt sich als Nachteil von Objektorientierter Programmierung sagen, dass Software-Projekte einer sehr genauen und **detailreichen und lückenlosen Planung** zugrunde liegen müssen. Frühe Planungsfehler in der nachhaltigen Gestaltung

der Software können bewirken, dass das Programm ineffizient und im schlimmsten Falle nicht mehr brauchbar wird.

Frage 2: Warum sind Properties besser als public Variablen?

Wie bei den Vorteilen der Objektorientierten Programmierung schon erläutert ist die **Datenkapselung**, also das Schützen von Eigenschaften und Methoden vor Zugriff von außen, ein zentrales Konzept der objektorientieren Programmierung.

Properties dienen dabei sehr gut als Zugriffs-Operationen auf Daten/Methoden von außen, ohne das Prinzip der Datenkapselung zu vernachlässigen. Dabei gibt es Get- und Set-Accessoren (Lese- und Schreib-Accessoren).

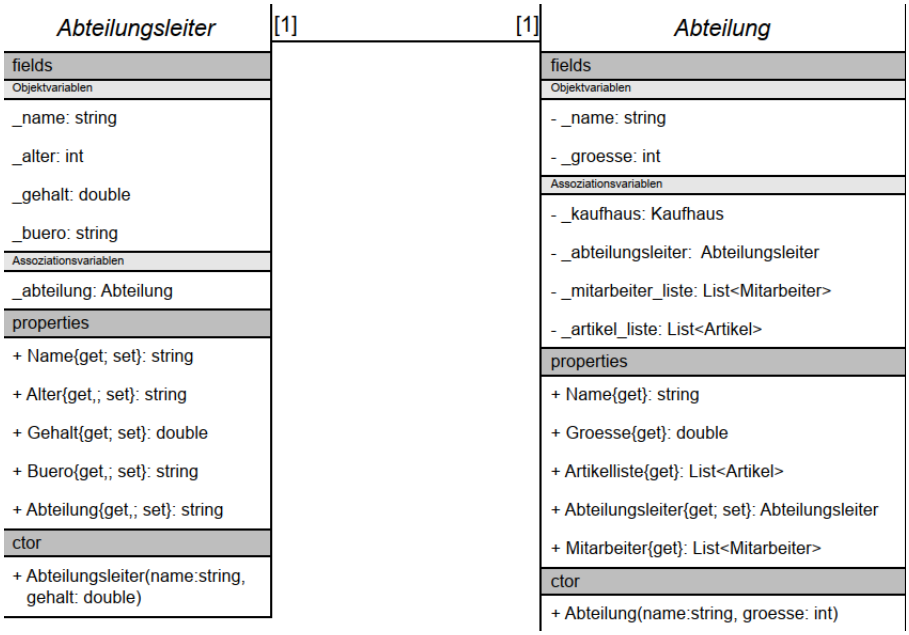


Wenn Variablen als **Public** deklariert werden, ist das Problem, dass ein ungeschützter Zugriff erfolgen kann. Sowohl Lese- als auch Schreibberechtigungen möglich sind. Vor allem im Umgang mit sensiblen Daten kann das zu einem Problem führen. Daher sollten Daten in der Regel erst einmal als **private** deklariert werden und dann können mithilfe von Properties die jeweiligen Zugriffsberechtigungen eingestellt werden.

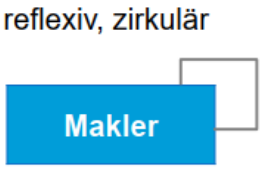
Frage 3: Was ist eine Assoziation und was eine Komposition – Nenne jeweils zwei Beispiele, eine aus dieser Studienleistung und eine beliebige.

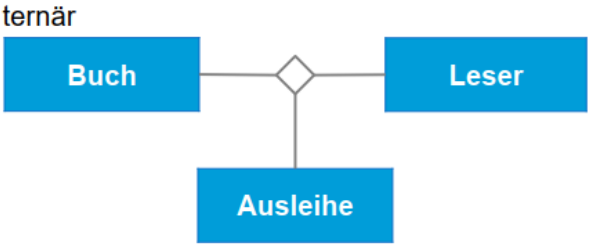
Assoziationen sind grundsätzlich (inhärent) **bidirektionale Beziehungen**. Als Beispiel hierfür kann die Beziehung zwischen einem Makler und Immobilien angeführt werden. Der Makler kennt seine Immobilien/Immobilien(besitzer) und diese kennen umgekehrt den Makler.

Als Beispiel aus der Kaufhaus-Software kann die Beziehung zwischen Abteilung und Abteilungsleiter angeführt werden. Die Abteilung kennt ihren Abteilungsleiter und der Abteilungsleiter kennt seine zugehörige Abteilung.

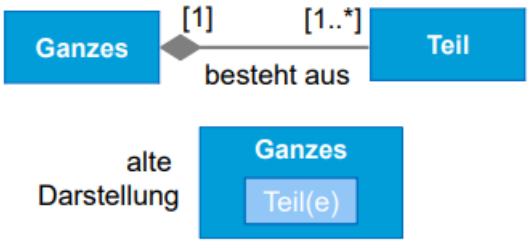


Allgemein kann bei einer Assoziation gesagt werden, dass sich Objekte gegenseitig „kennen“ aber unabhängig voneinander existieren. Neben den binären Assoziationen gibt es auch die reflexiven (zirkulären) und ternären oder andere höherwertige Assoziationen.





Die **Komposition** ist eine strenge Form der **Aggregation**. Eine Aggregation ist dabei eine spezielle Form der Assoziation – Eine Aggregation liegt dann vor, wenn zwischen Objekten eine Beziehung wie „ist ein Teil von“ oder „besteht aus“ vorliegt. Eine Komposition kann dabei folgendermaßen definiert werden: Wenn die eine Klasse „Ganzes“ haben eine eine Klasse Teile, dann besteht „Ganzes aus der Summe aller „Teile. Wenn „Ganzes“ gelöscht wird, dann werden auch alle „Teile“ gelöscht – die Teile können nicht ohne „Ganzes“ existieren.

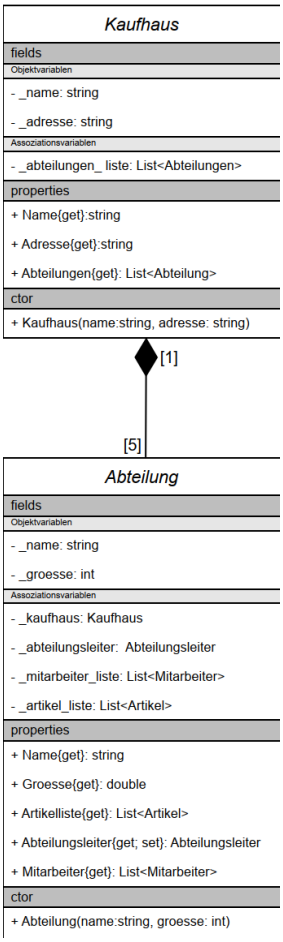


Ein Weiteres Beispiel für eine Komposition ist das folgende:

Das Gebäude besteht aus mehreren Räumen – ein Raum kann aber nicht ohne das Gebäude existieren.

Ein Beispiel aus der Kaufhaus-Software ist die Beziehung zwischen der Klasse Kaufhaus und der Klasse Abteilung – Das Kaufhaus besteht aus Abteilungen, die Abteilungen können aber nicht ohne das Kaufhaus existieren.

Jannis Schenk Matrikelnummer: 70480743



Jannis Schenk Matrikelnummer: 70480743

Teilstudienleistung 2:

Frage 1: Schaue dir die Klasse *DynamicCustomer* an. Die Daten werden von einer API in dem Format XML geladen. Was sind die Unterschiede zwischen XML und JSON? Verwende hierfür noch weitere Quellen.

JSON (JavaScript Object Notation) und XML (Extensible Markup Language) sind standardisierte Formate, welche für den Datenaustausch genutzt werden. Dabei können JSON und XML genutzt werden, um Daten auf dem Computer zu speichern.

XML ist ein offener Standard für die Speicherung und den Austausch von Daten und ist eine sogenannte Auszeichnungssprache („Beschreibungssprache“ → definiert die Bausteine eines Dokuments und legt die Beziehungen fest in denen sie zueinander stehen), welche zur Beschreibung der Struktur und des Inhaltes einer beliebigen XML-Datei dient. Nützlich wird XML vor allem im Kontext von Datenbanken oder Websites. XML ermöglicht es ebenfalls zusätzliche Informationen an Kontenpunkte in einem Dokument anzuhängen, ohne das sich das zugrundeliegende Format ändert.

XML stellt somit selbst keine Sprache zur Definition von Inhalten dar, sondern lediglich die Grundlage für die Definition einer solchen Sprache – Beispiel folgende Abbildung:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Cocktail>
<id>2</id>
<description>"Cuba Libre"</description>
<price>4.50</price>
<ingredient>"Rum"</ingredient>
<ingredient>"Cola"</ingredient>
<ingredient>"Lime"</ingredient>
<ingredient>"Sugar"</ingredient>
</Cocktail>
```

JSON ist ein „schlankes“ Datenaustauschformat, welches einfach zu lesen ist. Für Computer ist die Sprache ebenfalls einfach zu parsen und zu generieren. Es handelt sich um ein Textformat, welches unabhängig von Programmiersprachen ist.

JSON baut dabei auf zwei Strukturen auf: Name/Wert Paare (Bsp. Realisierung durch eine Hash Tabelle oder Schlüssel-Liste oder Wörterbuch…) und einer geordneten Liste von Werten. Bei JSON handelt es sich um eine universelle Datenstruktur, die von so gut wie allen modernen Programmiersprachen unterstützt wird – Als Beispiel folgende Abbildung:

```
{"Cocktail": {
        "id": "2",
        "description": "Cuba Libre",
        "price": "4.50",
        "ingredient": [
                "Rum",
                "Cola",
                "Lime",
                "Sugar"
        ]
}}
```

Die Unterschiede: JSON ist ein offenes Standardformat für den Datenaustausch. Es ist leichtgewichtig und einfach zu lesen, bietet aber keine Schema- oder Typinformationen. XML hingegen ist eine Markup-Sprache/ Auszeichnungssprache.

Im Gegensatz zu JSON ist XML weniger leichtgewichtig und eine langwierige Art der Darstellung strukturierter Daten. Dadurch ist XML jedoch aber auch funktionaler und auch komplexer, da es mehr Informationen über die Struktur des Dokuments erfordert, bevor es gelesen werden kann.

Frage 2: Was bedeutet das Schlüsselwort static?

Das Schlüsselwort **„static“** wird in vielen Programmiersprachen bei der Deklaration von Variablen und Funktionen verwendet. In C# wird es verwendet, um einen **statischen Member** zu deklarieren, sprich Klassen oder Methoden als statisch zu deklarieren. Statisch bedeutet, dass du diesen Typ nicht instanziieren kannst, also kein Objekt davon erstellen kannst. Daher kann kein Objekt einer statischen Klasse erstellt werden, oder über ein Objekt auf statische Member (Felder, Eigenschaften, Methoden,…) zugegriffen werden. Man greift stattdessen über den Klassennamen auf die Member zu.

Beispiel:

```
public class Planet
{
    public static int totalUniverseHumanCount = 0;

    public Planet(){
       totalUniverseHumanCount += 100000; // Statische variable erhöhen
    }
}
```

Jannis Schenk Matrikelnummer: 70480743

Teilstudienleistung 3:

Frage 1: Warum ist es besser viele kleine Funktionen/ Methoden zu haben als eine große? Nenne mindestens zwei Gründe.

Ein wesentlicher Vorteil darin, Funktionalitäten einzeln zu implementieren und nicht zu bündeln besteht darin, den **Programm-Code simpel dynamisch anpassen** zu können. Wenn Funktionen neu hinzukommen müssen, oder entfernt werden sollen, dann kann diese meist unabhängig von anderen Methoden geschehen. Es ermöglicht somit eine einfache und schnelle agile Bearbeitung des Codes.

Ebenfalls ein großer Vorteil ist, die **bessere Übersichtlichkeit**. Es ist deutlich einfacher zu erkennen, was die einzelnen Methoden im Detail machen, wenn ihre Funktionalität kompakt gehalten wird und es schnell ersichtlich ist, wofür die Methode im Kern verwendet wird.

Frage 2: In der Objektorientierten Programmierung gibt es das Konzept der Kapselung. Was ist damit gemeint?

- Datenkapselung: Die **Datenkapselung** ermöglicht es, Daten, Eigenschaften oder Methoden vor dem Zugriff von außen, beziehungsweise vor ungewollten Zugriff zu schützen. Es kann auch als eine Art **Geheimnisprinzip** verstanden werden und kann als Grundlage von Daten-/Informationssicherheit in den Programmen dienen.

Frage 3: Was ist der Unterschied zwischen einem relativen und einem absoluten Dateipfad und warum könnte es besser sein, einen relativen Pfad zu benutzen?

Der absolute Dateipfad kann beispielsweise wie folgt angegeben werden:

C:\Ordner1\Ordner2\datei1.txt

Absolute Pfadangabe enthalten hierbei die vollständige Position einer Datei und sind somit unmissverständlich zu finden. Dabei wird immer vom Rootverzeichnis (oberste Ebene der Datenstruktur eines Computers) ausgegangen. Es lässt sich komprimiert aussagen:

Absolute Pfade geben den absoluten, spezifischen Pfad an, während relative Pfade in Relation zu ihrem Ausgangspunkt stehen.

Der relative Dateipfad könnte dabei wie folgt aussehen:

..\Ordner2\datei.txt

Relative Pfade beschreiben die Position einer Datei relativ zu der Datei oder dem Ordner, in dem man sich gerade befindet. Damit sind sie komplett unabhängig von der Datenstruktur des Computersystems, auf dem sie sich befinden. Dabei ist zu beachten: Relative Pfade können nicht auf einen Speicherort auf einem anderen Laufwerk verweisen.

Relative Pfade bieten sich vor Allem in der Programmierung von Ordnerzugriffen an, da nicht immer der absolute vollständige Pfad bekannt ist und somit über den relativen Pfad, aus Ordnern Funktionen ablaufen können und somit trotzdem das Dateiverwaltungssystem verwendet werden kann. Beispielsweise wenn eine kommerzielle Software aufs Dateisystem von lokalen Benutzern zugreift und sich die absoluten Pfade untereinander unterscheiden.

Literatur XML/JSON

- https://appmaster.io/de/blog/json-vs-xml-de