**Vererbung und Polymorphie**

* Polymorphie erklären (anhand Bsp), Einsatzmöglichkeiten dieser
  + Game: Entity-Class aller objekte im Game
  + Biz: Collections, in C# can implement own functions for default Collection-Actions
* Welche Member werden vererbt von Super-, zu Subklasse?
  + nicht vererbt: statische Member, Konstruktoren, Finalizer
* Konsequenzen der Vererbung, bei Instanzierung von Objekten, Anwendung der Konstruktoren?
  + Wenn Deklaration und Instanzierung die gleiche Klasse aufweisen, dann wird in eine Methode X, der Superklasse, mit virtual, direkt gesprungen und ausgeführt, bei *new* und *override*.
  + Wenn Deklaration [Superklasse] = Instanzierung [Subklasse], dann, wird nur bei *override* die Methode einer Subklasse ausgeführt, aber nicht so bei *new*.
* Zugriffsmodifizierer [public, private, protected] innerhalb der Vererbung erklären?
  + public: Subklasse hat Zugriff
  + private: Subklasse hat keinen Zugriff
  + protected: Nur in Subklasse
* Syntax zur Deklaration der *Generalisierung*?

public class A {}

public class B : A {}

* </>: Vererbungshierarchie entwickeln, 3-6 Klassen, implementieren
* </>: Methoden und Eigenschaften überschreiben können
  + superclass' method: virtual ...
  + subclass' method: override ...
* </>: Member der Superklasse in der Subklasse verbergen, durch neue Implementation
  + new ...
* </>: Anwenden von *new* und *override*
* unterscheiden zwischen override und new
  + **new**: forced den Compiler, dass gleichnahmige Methode wie in Parentclass existieren soll.
  + **override**: reimplementiert explizit eine Methode der Parentklasse.

**UML (Unified Modeling Language)**

* Klassendiagramme
  + *Attribut*: Teil einer Klasse
  + *Operation*: Aktion welche ein Teil der Klasse auführen soll
* Arten von Assoziationen: Siehe[Zusammenfassung BBZW](https://gh.3n3a.ch/my-notes/berufsschule/m226a-objektorientiertes-programmieren.html)
  + [Zusammenfassung Klassendiagramme](https://gh.3n3a.ch/my-notes/berufsschule/m226a-klassendiagramme.html)
* </>: Klassendiagramme erstellen anhand vorgaben
* </>: UML Diagramm in C# abbilden

| **Zahl** | **Bezeichnung** | **Notiz** |
| --- | --- | --- |
| 1 | Name Klasse |  |
| 2 | Felder mit Namen und deren Typ | <-- definiert die Werte (Interface/Constructor) |
| 3 | Methoden mit Namen, Parametern und deren Return-Typen | --> gibt/setzt Werte (getters/setters) |

**Klassendiagramme**

**Begriffe**

**Glossary**

* **Generalisierung:** Benutzung eines Lösungsweges für andere, ähnliche Probleme durch anpassung des originalen Weges.
* **Vererbung:** Eine Klasse kann ein, oder mehrere Eigenschaften einer anderen Klasse übernehmen. Dient der Abstraktion / Vereinfachung.
* **Superklasse:** Eine Klasse von der andere Klassen Code erben.
* **Subklasse:** Eine Klasse welche von einer Superklasse Code erbt.
* **UML:** eine etablierte, objektorientierte, standardisierte und werkzeugunterstützte Modellierungssprache für die Visualisierung, Beschreibung, Spezifikation und Dokumentation von Systemen.

| **Symbol** | **Beschreibung** |
| --- | --- |
| - - -> | Abhängigkeit |
| -----> | unidirektionale Assoziation |
| ------- | bidirektionale Assoziation |
| <>----- | Aggregation |
| ◀▶----- | Zusammensetzung |
| \* | unendlich (kann assc. sein) |
| 1..2 | Bereich (hängt davon ab, ob assc.) |
| 1 | konstante (muss assc. sein) |

**Klassen**

*Rechtecke mit Feldern*

**Relationships**

**Abhängigkeit**

* Klasse kann abhängig von anderer Klasse sein
  + Bsp: Buch kann im Besitz von Person sein

Wenn Variable, wie in Klasse *Person*, nicht definiert ist im constructor, von anderer Klasse kommt, dann modeliert mit - - -> Pfeil

**Direkte Assoziation**

* Klasse speichert andere Klasse in Feld
  + Bsp: Person kann im Besitz von *n* Büchern sein, ein Buch kann besessen werden von *n* Personen

**Bidirektional Assoziation**

* Beide Klassen speichern sich gegenseitig in ihren Feldern ab
  + Bsp: Person speichert alle Bücher in Besitz, \ Buch speichert alle Personen welche es besitzen.

**Aggregation (Sammlung/Gruppe)**

* Klasse eins *"besitzt"* Klasse zwei.
  + Klasse zwei ist ein Teil von Klasse eins

***Linie****: eine P. viele Bücher; ein Buch, eine P.*

Fast schon als ob eine Person im echten Leben ein Buch besitzt, und egal ob das Buch einer anderen Person den gleichen Titel und Inhalt hat - es ist ein anderes Buch.

**Komposition (Zusammensetzung)**

* Klasse eins *"besitzt"* Klasse zwei.
  + wenn eins *"stirbt"*, dann stirbt zwei auch \ (gleiche Lifetime)

*In idealer Welt gibt es Ebooks mit DRM, wo das DRM nur von einer einzigen Person benutzt werden kann und dann sozusagen das Buch als solches nur lesbar ist solange die Person am leben ist.*

**Vererbung**

* Eine Klasse erweiter eine andere Klasse
  + Mutter, Tochterklassen
  + *Email ist eine Erweiterung der Dokument-Klasse. Benutzt Dokument als Grundbaustein*

**Testing**

**Unit Tests**

* Don't have access to filesystem, database, networking ...
* These tests should use mocking (C# -> Moq)

**Default Pattern**

* Arrange (instantiate objects, prepare data)
* Act (execute method)
* Assert: (Check Result)

*Arrange could also be missing, and be defined in constructor*

**Integration Tests**

* Have access to system, network resources
* Test how a function integrates into these services