Hochschule RheinMain
Fachbereich Design Informatik Medien
Marcus Thoss, M.Sc.

# Hardwarenahe Programmierung II
# SS 2020
# LV 2512

# Übungsblatt 3

## Aufgabe 3.1 (Text-to-Model):

Modellieren Sie das nachfolgend beschriebene Szenario in einem UML- Klassendiagramm:

A domestic animal is a special kind of animal, and an animal is a special kind of being. Animals can be fed food, which ends up in a stomach; thus the animal gains weight. An animal can be male or female, and every being has a distinct year of birth. A domestic animal has a name, which is initially undetermined and can be changed, and it has an owner (that is also a being). Other kinds of beings besides animal are plant and fungus. Some domestic animals are equipped with saddles. All animals can produce a distinct sound.

Überlegen Sie, welche Klassen, Verwandschaftsbeziehungen und Assoziationen Sie benötigen. Welche Eingeschaften (Data Members) haben die Klassen und welche Methoden (Member Functions) zum Abfragen oder Verändern von Eigenschaften sollen Sie besitzen? Achten Sie auch darauf, welche Eigenschaften zu Beginn festgelegt werden müssen und welche später änderbar sind (oder nicht).

Nutzen Sie sowohl PlantUML als auch einen grafischen Editor, um das Diagramm anzulegen.

## Aufgabe 3.2 (Model-to-Code):

a) Legen Sie sich in Ihrem `git`-Repository ein neues Unterverzeichnis für diese Aufgabe an.

b) Kopieren Sie die Diagramme der Aufgabe 3.1 und Quellcode und Makefile des 2. Aufgabenblatts in das Verzeichnis.

c) Passen Sie den Code Ihrer `Animal`-Klasse dem Modell von Aufgabe 3.1 an und ergänzen Sie die restlichen im Modell beschriebenen Klassen durch Hinzufügen entsprechender `.h`- und `.cc`-Dateien.

Achten Sie darauf, dass nun *nur* noch *domestic animal* einen Namen haben kann, dieses Attribut muss also aus `Animal` verschwinden.

Ordnen Sie Data Members und Member Functions, soweit nicht schon im Diagramm geschehen, die jeweils angemessene Sichtbarkeit (`public`/`protected`/`private`) zu und ergänzen Sie sie ggf. im Diagramm.

*Hinweis:* Vorschlag für den Klassennamen von *domestic animal*: `DomesticAnimal`, Dateiname `domestic_animal.[h|cc]`

d) Testen Sie zwischendurch mit dem Testprogramm des zweiten Aufgabenblatts und passen Sie es entsprechend an.

e) Fügen Sie in `Being` eine Methode `print()` hinzu, die die Eigenschaften des Objekts auf `cout` ausgibt. Überschreiben Sie die Methode in den abgeleiteten Klassen (override), und rufen Sie dort auch die `print()`-Methoden der Elternklassen auf, soweit angebracht.

f) Versuchen Sie, für die Vorgaben aus Aufgabe 3.1 jeweils eine Funktion als Test zu programmieren und lassen Sie Ihr Testprogramm alle diese Funktionen nacheinander ausführen.

- Jede Funktion sollte zu Beginn eine Kurzbeschreibung ihres Tests ausgeben.
- Die Ausgaben der Tests sollen auf `cerr` (nicht `cout`) erfolgen.
- Die Testfunktionen sollen als Prototyp dem Muster `bool Test....()` entsprechen
- Sie sollen `true` zurückgeben, wenn der Test erfolgreich war, sonst `false`.
- Schlägt ein Test fehlt, bricht das Hauptprogramm mit einer Fehlermeldung ab.

Beispiel:

- *Aussage*: „Animals can be fed food, which ends up in a stomach; thus the animal gains weight."
- *Ausgabe*: „Feeding test"
- *Funktionsprototyp*: `bool TestFeeding();`
- *Umsetzung*: Ein Tier wird mit bekanntem Gewicht angelegt und ausgegeben und bekommt eine bekannte Menge Nahrung gefüttert. Anschließend wird das Gewicht ausgegeben und es wird geprüft, ob das neue Gewicht dem ursprünglichen plus der Zufütterung entspricht (analog Aufgabe 2.1.h)

*Hinweis:* Verwandschaftsbeziehungen sind in Tests natürlich schwer zu erfassen, versuchen Sie hierfür, einem Basisklassenpointer ein Kindklassen-Objekt zuzuweisen.

g) Kommentieren Sie Ihren Code und erzeugen Sie mit `doxygen` die Dokumentation. Binden Sie darin auch das UML-Diagramm ein.