

# Catch up Code I - Code II

## GitHub HTUGLIB24

- new Repository
- command line | Ordner erstellen > RUB in Konsole öffnen  
> Befehl in Konsole eingeben
- main: lokal  
origin main: remote

Commit message immer  
eingeben, wenn vergessen  
in neue Datei oben  
eingeben und speichern

## VS Code / TS / JS

### First Steps

- Projektordner erstellen
- Files erstellen ts, index.html, style.css
- tsconfig (außerhalb, wenn projektspezifische Einstellungen  
getroffen werden, dann innerhalb vom Projekt > + keine Fehler  
von anderen Projekten)
- Terminal öffnen tsc --watch -> js Datei
- **ctrl + shift + b** run build task
  - build (einmal)
  - watch (dauerhaft)  
rechts im Terminal zu sehen
- html -> ! + enter (Aufbau, Struktur html Seite)
  - ↳ Seiten verlinken
    - <link rel = "stylesheet" href = "style.css"> link muss nicht geschlossen werden
    - <script src = "blub.js" defer></script> defer: erst wenn html tag geschlossen ist wird das script ausgeführt
  - ↳ es können mehrere files verlinkt werden
  - ↳ Verlinkungen im Live Server testen

## Funktionen

- let  $c = a + b$  = ist immer eine Zuweisung  
↳ Leserichtung Code
- Code beim Speichern formatieren
  - ↳ Settings > Format > Format on paste / save
  - ↳ muss bei Zusammenarbeit gleich sein
- F2 Variablenbezeichnung an jeder Stelle im Code ändern
- Blockkonzept ↳ (hierarchisch)
  - ↳ alles im Block ist gültig in der Ebene und darunter, nicht darüber
- Aufbau

function Name (Übergabeparameter): Ausgabetyp {  
... nur hier gültig

?

↳ Übergabeparameter „Platzhalter“, der Typ wird hier definiert, sind immer allgemein

- Funktionen sind logische sich wiederholende Funktionen

## Interfaces

- bei Benennung in der Einzahl
- Beispiel Buch

bei Arrays macht Mehrzahl Sinn

```
▽ interface Book {  
    title: string;  
    pages: number;  
    authors: string[];  
    chapters: string[]  
}  
  
interface Bookcase {  
    room: string  
    books: Book[]  
}
```

## Schleifen und Bedingungen

- console.log nutzen zum Überprüfen
- for loop
  - ↳ immer gleicher Aufbau
  - ↳  $i \geq \text{index}$
- for in : gibt einen key zurück
  - ↳ der key in einem Array ist der index i
- for of | for each : gibt Element aus

```
function howManyPagesDoesBookHave(_title: string): number {  
  
    // for (let i = 0; i < bookcase.books.length; i++) {  
    //     const book = bookcase.books[i];  
    //     return book.pages  
    // }  
  
    // for (const i in bookcase.books) {  
    //     const book = bookcase.books[i];  
    //     return book.pages  
    // }  
  
    for (const book of bookcase.books) {  
        if (book.title == _title) {  
            return book.pages  
        }  
    }  
  
    return -1  
}
```

- if Bedingung
  - = Zuweisung
  - => Vergleich
- Debugger
  - ↳ tsconfig: sourceMap: true
  - debugger console ts
  - Breakpoints (auf Zahl klicken) > refresh page

! line beim Breakpoint  
o ist noch nicht  
ausgeführt  
↳ darüber

# Objektorientiertes Programmieren

EIA 2 Inverted Classroom Chapter 9

## Objektorientierung

- die Abstraktion der Umwelt
- wir bauen ein vereinfachtes Modell der Umwelt, um diese besser erfassen / erklären zu können
  - ↳ klassifizieren, generalisieren
- zu bestimmten Dingen gehören bestimmte Eigenschaften
  - ↳ ermöglicht es Voraussagen über unbekannte Dinge zu treffen
  - ↳ z.B. hat ein Auto immer eine Marke, Fenster, vier Reifen, ein Lenkrad, ...
- Abstraktion durch das Aufschlüsseln in einzelne Bestandteile
  - ↳ diese sind generalisiert, können das eigentliche Teil aber näher beschreiben
  - ↳ Klasse Auto → Eigenschaften: Reifen, Marke, Baujahr, Farbe, ...

## Modellierung

Vector
x : number
y : number
scale (factor: number) : void
add (addend: Vector) : void

Name  
Eigenschaften

Aktivitäten  
↳ Funktionen / Methoden

# Implementierung

- statt interface → **class** benutzen
- die Aktivitäten werden als Funktion implementiert (ohne function als Schlüsselwort)
- Funktionen innerhalb von Klassen nennt man Methoden
  - ↳ Objekte einer Klasse können mit diesen Methoden sich und ihre Umwelt beeinflussen
- die interne Arbeitsweise / Funktionen müssen dem Programm nicht bekannt sein, um das Objekt zu nutzen
  - ↳ das Objekt ist „gekapselt“ (eine Black-Box)
  - ↳ braucht dann nach die notwendigen Parameter

Methoden

Kapselung

# Instanzierung

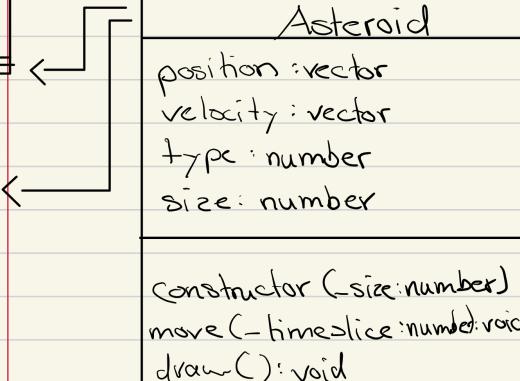
- Erzeugung eines neuen Objektes einer Klasse
- mit der Anweisung **new**
- **namespace** to organize code

# Class Diagram

am Beispiel Asteroids

Was weiß es?

Canvas  
Rendering  
Context



Was hat es?

Was kann es?

## 5 Fragen für jeden Objekttyp

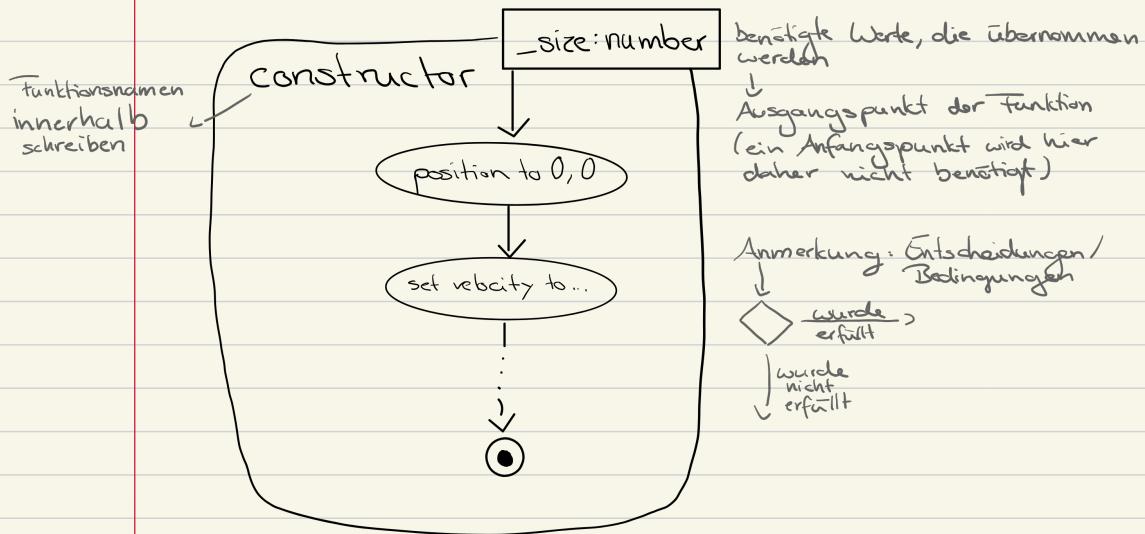
1. was hat es?
2. was kann es?
3. was weiß es?
4. wer hält es?
5. was ist es?

## Klassen Beispiel

```
namespace FarmSimulation {  
  
    export class Animal {  
  
        public species: string;  
        public name: string;  
        public noise: string;  
        public food: FoodSupply;  
        public stomach: number;  
  
        public constructor(_species: string, _name: string, _noise: string, _food: FoodSupply, _stomach: number) {  
            this.species = _species;  
            this.name = _name;  
            this.noise = _noise;  
            this.food = _food;  
            this.stomach = _stomach;  
        }  
  
        public eat(): void {  
            this.food.stock -= this.stomach; // kennt die Variablen durch FoodSupply, kann darauf zugreifen / x -= y  
            this.food.report();  
        }  
  
        public sing(): void {  
  
            const song: string = this.noise + " " + this.noise;  
  
            // \n new line for line breaks  
            const verse1: string = "Old MacDonald had a farm. E - I - E - I - O.\nAnd on that farm he had a " + this.species;  
  
            console.log("This is " + this.name + " the " + this.species);  
            console.log(verse1);  
        }  
    }  
}
```

# Activity Diagram

am Beispiel des Konstruktors



# Zeitsignale

EventListeners bleiben erhalten und aktiv, auch wenn das Hauptprogramm bereits beendet ist

- ~~X~~ Symbol für Zeitsignale
- Zeitsignale: Bild anzeigen, dann einige Zeit warten, dann das nächste Bild zeichnen und anzeigen
  - ↳ bei Schleifen ist der Browser nur mit dem Loop beschäftigt und hat keine Zeit mehr das Bild anzeigen
- 20 times per sec

 .  

- `window.setTimeout(handler, time, args ...)`
  - ↳ die handler-Funktion wird nach Ablauf der angegebenen Zeit (time, in Millisekunden) aufgerufen
  - ↳ Zeit läuft ab dem Anweisungsaufzug
  - ↳ weitere Parameter können dabei an die Funktion übergeben werden, wie z.B. args
- `window.setInterval(handler, time, args ...)`
  - ↳ die handler-Funktion wird periodisch aufgerufen mit einem zeitlichen Abstand (time)
  - ↳ weitere Parameter können dabei an die Funktion übergeben werden, wie z.B. args
- `window.requestAnimationFrame(handler)`
  - ↳ die handler-Funktion wird periodisch aufgerufen mit einem zeitlichen Intervall, das für die grafische Aufbereitung sinnvoll erscheint
  - ↳ dabei entscheidet der Browser über das Intervall
  - ↳ z.B. versucht Chrome eine Bildwiederholrate von 60 Bildern pro Sekunde zu erreichen (60 fps (frames per second))

# Votes

## Null

- null pointer, "i do not point to any data"
- a variable
- value nothing

## ! Ausdruck

- vor einem Ausdruck bedeutet das Ausufezeichen „nicht“  
↳ nicht Name

## -beispiel?: typ

- das Fragezeichen gibt die Option den Ausdruck abzufragen
- es wird kein Fehler ausgegeben, wenn der Ausdruck nicht abgefragt/angegaben wird

## for in/of loop

- for | in | of  
  | index   | value  
  | type   |

## Werte und Referenzen

### call by value

- bei der Übergabe an eine Funktion wird die Variable kopiert
- außerhalb der Funktion bleibt die ursprüngliche Variable unverändert erhalten

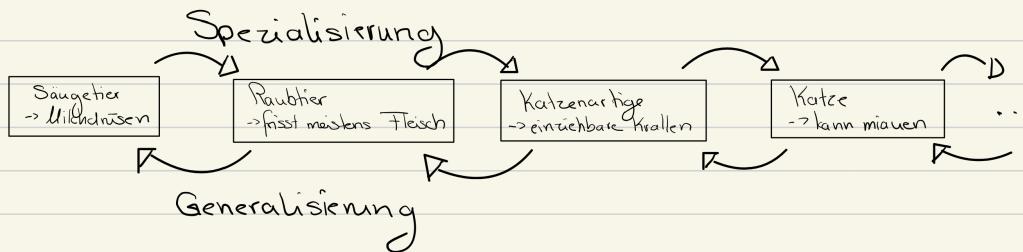
### call by reference

- hier wird die Variable nicht kopiert, sondern die ursprüngliche Variable verändert, d.h. sie ändert sich auch außerhalb der Funktion
- gilt für komplexe Datentypen, wie Objekte
- verschiedene Funktionen können so das gleiche Objekt referenzieren

# Inheritance

Vererbung - Inverted Classroom L10

## Klassendiagramm



Raubtier ist die generalisierte Superklasse von Katzenartige.  
Katzenartige ist die spezialisierte Subklasse von Raubtier.

## Vererbung

3. Prinzip der Objektorientierung!

- eine Superklasse vererbt all ihre Eigenschaften an die Subklasse
  - ↳ eine Art Stammbaum
- Superklasse  $\hat{=}$  Elternklassen / Oberklassen  
Subklassen  $\hat{=}$  Kindklassen / Unterklassen
- jedes Objekt soll so
  - ↳ schlau wie nötig sein
  - ↳ dumm wie möglich
- die Subklassen haben alle Eigenschaften und Methoden der Superklasse
- bei neuen Eigenschaften braucht die Subklasse einen Konstruktor
- → Klassen kennen sich

## Vererbung

- Super- und Subklassen sind trotz ihrer Verwandtschaft noch zwei Klassen, die beide erzeugt werden müssen  
↳ wird die Subklasse erzeugt muss die Superklasse auch instanziert werden
- **Super()** zum Aufrufen des Constructors der Superklasse in der Subklasse
  - ↳ Parameter, die übergeben werden sollen kommen in die Klammern (auch Parameter aus der Superklasse)

## Polymorphie

- Klassen können verschiedene Gestalten annehmen  
↳ verschiedene Subklassen

# Prinzipien der Objektorientierung

**Abstraktion** unterbrechen der Objekt in die grundlegenden Funktionen und Eigenschaften

**Kapselung** Klassen sind in sich geschlossen und können daher variabel in verschiedenen Projekten genutzt werden

**Inheritance** die Superklasse vererbt ihre Attribute und Funktionen an ihre Subklassen

**Polymorphie** eine Klasse kann (in Form von Subklassen) verschiedene Gestalten annehmen

## 5-Fragen

1. was hat es?
2. was kann es?
3. was weiß es?
4. wer hält es?
5. was ist es?

## 2 Grundprinzipien

1. Die Objekte sollen so dumm wie möglich sein.
2. Die Objekte sollen so schlau wie nötig sein.