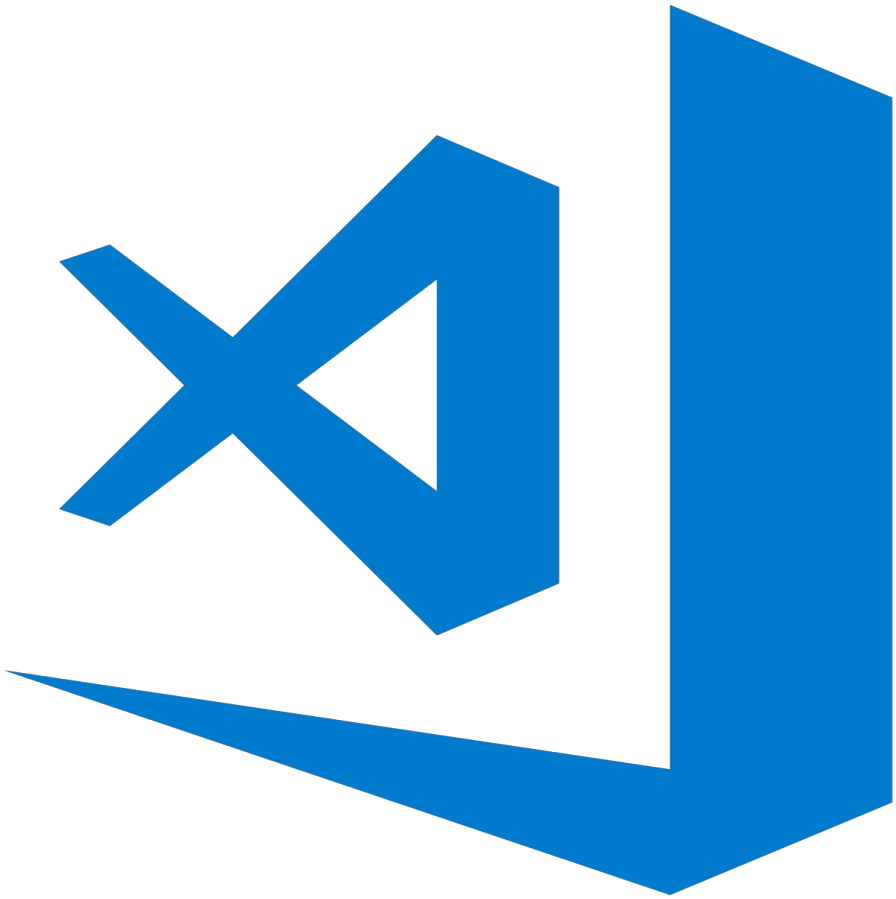
HOGESCHOOL ROTTERDAM

Creative Media and Game Technology

Object Oriented Game Development

CMTTHE01-4 en PRG01-8
2018 - 2019

QUICK START



Lesvoorbereiding

Werkomgeving

Visual Studio Code en Typescript

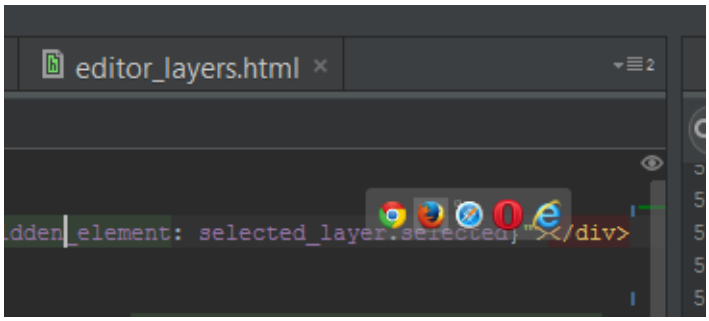
Installeer **VS code** via https://code.visualstudio.com en NodeJS via https://nodejs.org/en/
Open daarna een terminal om **Typescript** te installeren:

```
sudo npm install -g typescript
```

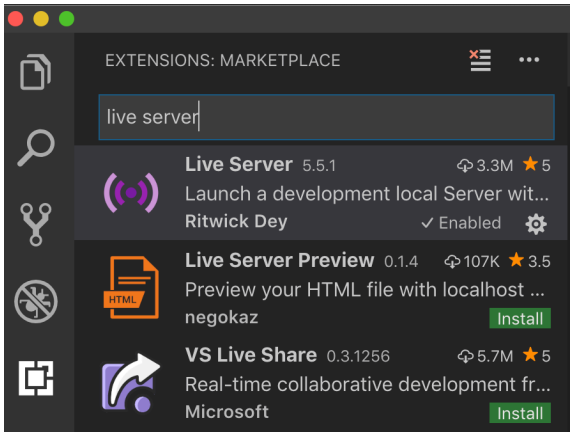
Localhost

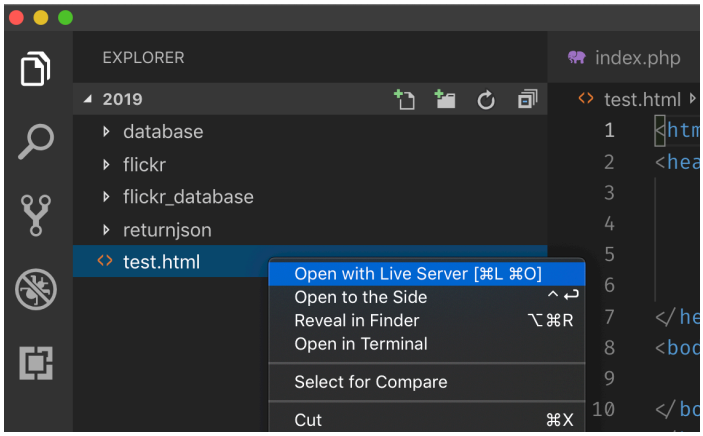
Je test een project altijd via **`http://localhost/..`**, en niet via **`file://...`**

Als je **MAMP** of **XAMPP** al hebt geïnstalleerd dan heb je al een localhost server. Plaats je projecten in wwwroot / htdocs



In PHPstorm kan je via de browser knoppen een localhost openen





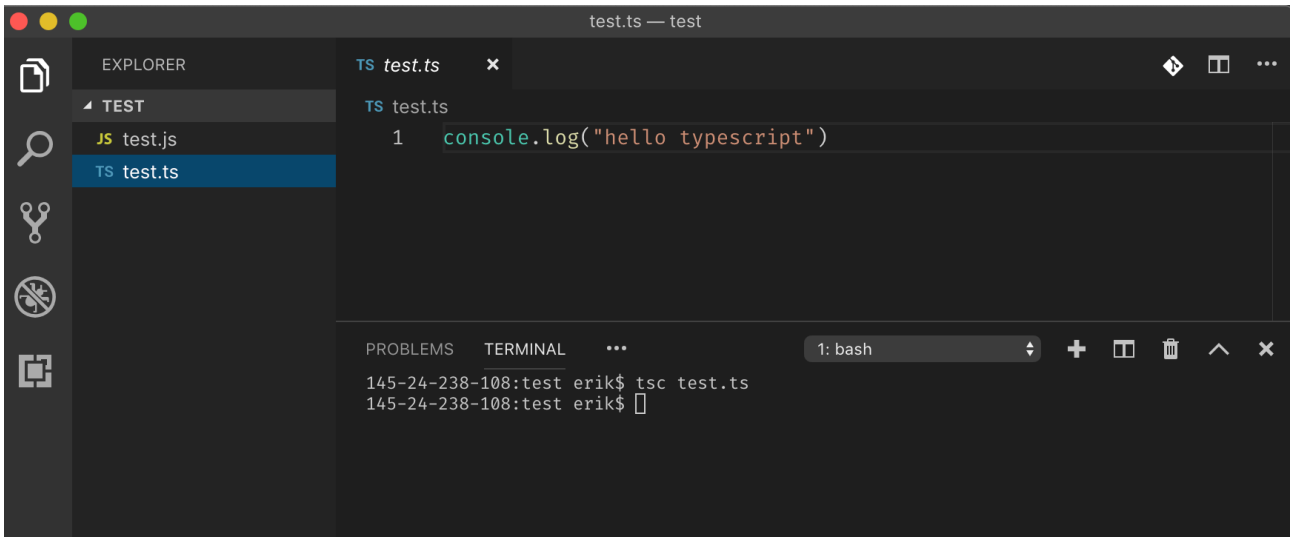
In VS Code kan je de Live Server plugin installeren. Daarna kan je een html file openen in een server met de knop "Go Live".

Typescript testen in de browser

Het **tsc** commando compileert **typescript** files naar **javascript.** Maak een test.ts bestand aan en open een terminal. Typ "**`tsc test.ts`**" om te zien of typescript goed omgezet wordt.

De **-w** flag zorgt dat je niet telkens tsc hoeft te typen als je code is aangepast.

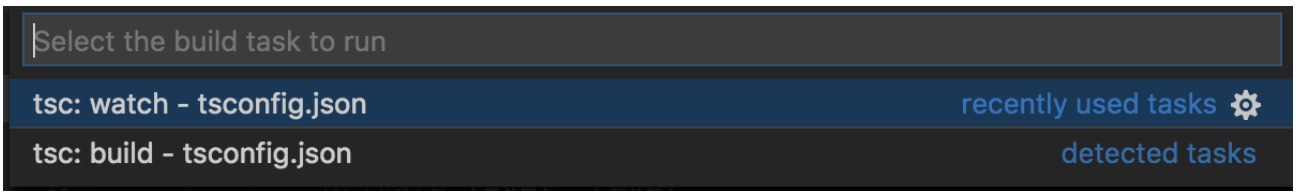
```
tsc test.ts -w
```



Met een "tsconfig.json" (een configuratiebestand) kan je aangeven dat al je losse files gebundeld moeten worden naar één .js file. Dit bestand vind je ook in het voorbeeldproject.

```
{
    "compilerOptions": {
        "strict": true,
        "target": "es5",
        "removeComments": true,
        "noUnusedLocals": true,
        "noUnusedParameters": true,
        "sourceMap": true,
        "outFile": "docs/js/main.js"
    },
    "include": [
        "dev/**/*"
    ]
}
```

Druk op **CMD + SHIFT + B** *(windows: CTRL + SHIFT + B)* in VS Code om het compile proces op te starten zonder terminal.



Als je de Live Server plugin gebruikt, zie je code wijzigingen meteen in je browser!

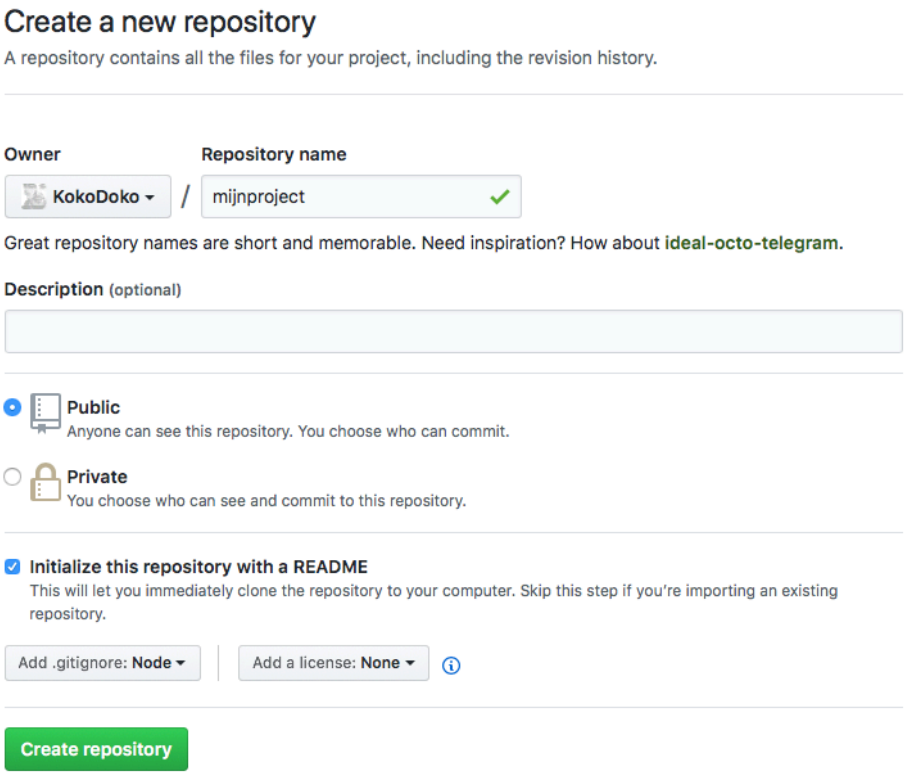
GIT

Git installeren

Installeer git via https://git-scm.com/downloads
Maak een GitHub account aan op https://www.github.com
Optioneel: vraag een gratis student account aan: https://help.github.com/articles/applying-for-a-student-developer-pack/

Een repository starten via GitHub

Je kan een nieuw git project zowel lokaal starten als via GitHub. Als je dit via GitHub doet staan al je instellingen meteen goed. Ga naar je GitHub account en klik 'new repository'

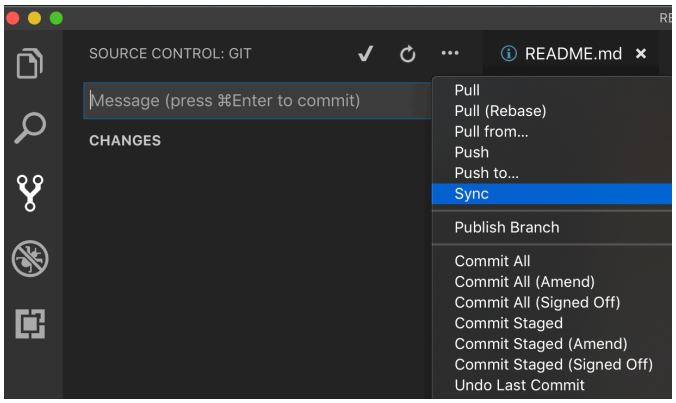


Navigeer in een terminal venster naar je projectmap.

```
cd /Users/Naam/schoolprojects/
git clone https://github.com/naam/my-project.git
```

Stap 1 - Sync

Het is raadzaam om eerst je project te synchroniseren voordat je iets gaat wijzigen. Klik in VS Code op de sync knop. Je project wordt nu opgehaald van GitHub.

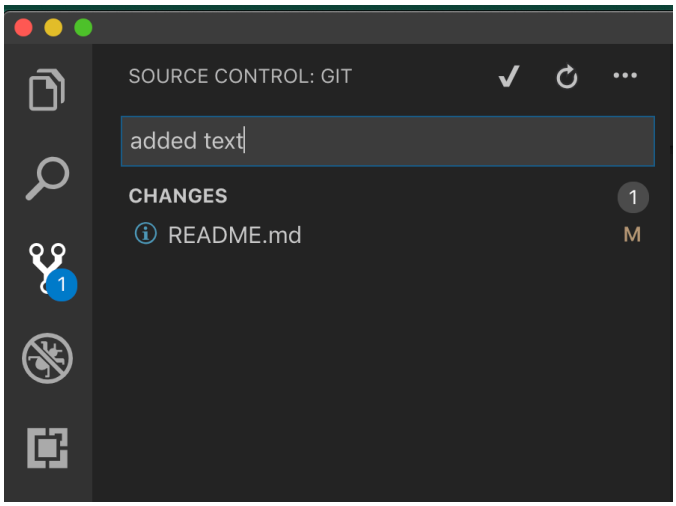


Stap 2 - Code wijzigen

Maak nu een wijziging in je project, bijvoorbeeld in de ReadMe file. Sla de wijziging op. Je ziet de wijziging verschijnen onder "changes"

Stap 3 - Code commit

Als je tevreden bent over je wijziging klik je op het commit vinkje. Je moet een '*commit message*' meegeven. Na de commit is je wijziging onderdeel geworden van het versiebeheer.



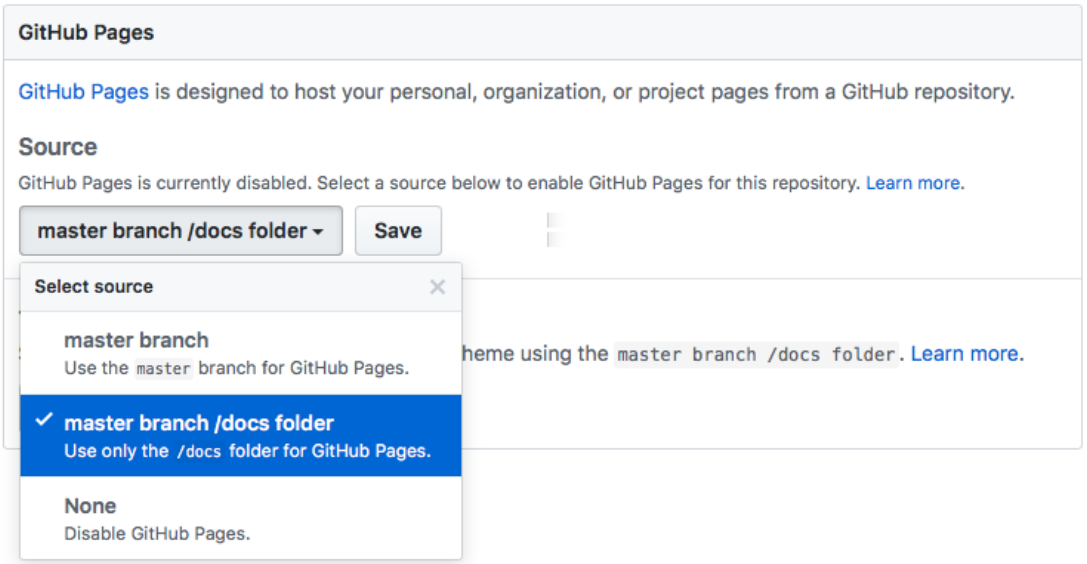
Stap 4 - Sync

Druk nogmaals op sync om je project te synchroniseren met de online versie op GitHub. Ga naar je GitHub pagina om te zien of je ReadMe file daar ook is aangepast!

Je project live zetten op GitHub Pages

Ga naar '**settings**' in je repository op github.com. Onder 'GitHub Pages' kies je master branch/ docs folder. Klik op 'save'.

Je **docs** folder staat nu live op: https://projectnaam.github.io/



Object Oriented Game Development

Herhaling / samenvatting van module CMTTHE01-4

Typescript

Static typing

Typescript is Javascript met *static Types*. Dit betekent dat je het **type** van je variabelen niet meer kan veranderen.

```
let a = 1                    // NUMBER
let b = true                 // BOOLEAN
let c = "Charmander"         // STRING

a = "hoi"                    // ERROR!!!
```

Als je een variabele declareert zonder er een waarde aan te geven, dan moet je handmatig zeggen wat het type is:

```
let a : number
let b : boolean
let c : string
```

Arrays

```
let list: number[]              // een array van numbers
let list = [1,4,3]              // een array van numbers
```

Functions

Bij een function argument moet je altijd het type aangeven

```
function getValue(a:number) {
     return true
}
```

Return type

Het is meestal niet nodig om het type van de return value aan te geven

```
function getValue(a:number) : boolean {
     return true
}
```

Object Oriented Programming

Classes

```
class Person {
    name:string
    constructor(){
        this.name = "Finn"
    }
    sayHi(){
        console.log("hi!")
    }
}
```

```
let p = new Person()

console.log(p.name)
p.sayHi()
```

Inheritance

```
class Person {
    constructor(){
       console.log("I am a person!")
    }

}
```

```
class Driver extends Person {

    private vehicle:string = "bus"

    constructor(){
       super()
       console.log("and I drive a " + this.vehicle)
    }
}
```

Driver is een Person

```
let d = new Driver()

// I am a person!
// and I drive a bus!
```

Composition

Een class bevat een instance van een andere class.

```
class Driver {

    public name : string = "Jake"

    constructor(){
    }
}
```

```
class Car {

    private driver : Driver

    constructor(){
       this.driver = new Driver()
       console.log("My driver is " + this.driver.name)
    }
}
```

Car heeft een Driver

```
let d = new Car()

// My driver is Jake!
```

Encapsulation

De **private** en **public** keywords gebruik je om methods en variabelen af te schermen van code buiten de class

```
class Person {
    private age:number

    public setAge(i:number){
        this.age = i
    }

    public getAge(){
        return this.age
    }
}
```

De public functie zorgt er voor dat we de private variabele nog wel kunnen opvragen buiten de class:

```
let p = new Person()

p.setAge(12)

let a = p.getAge()
```

get en set

Een andere mogelijkheid om dit te doen is het werken met **get** en **set.** Let op dat je variabele een andere naam heeft dan de get en set functies, bijvoorbeeld door er een underscore voor te zetten.

```
class Person {
    private _age:number

    get age() {
        return this._age
    }

    set age(i:number) {
        this._age = i
    }
}
```

Bij het werken met **get** en **set** lijkt het net alsof je een variabele aanroept:

```
let p = new Person()

console.log(p.age)

p.age = 12
```

Game techniques

Game loop

RequestAnimationFrame wordt in 1/60 seconde aangeroepen. Doordat de functie zichzelf aanroept krijg je een framerate van 60fps. De functie pauzeert automatisch als de tab niet actief is.

```
class Game {
    constructor() {
       this.gameLoop()
    }
    private gameLoop(){
        // game code hier

        requestAnimationFrame(()=>this.gameLoop())
    }
}
```

DOM elementen

Geef semantische elementen via CSS **position: absolute**, en **display:block**.
Via **transform:translate** verplaats je het object - dit gebruikt de GPU van je device.

HTML

```
<car></car>
```

CSS

```
car {
   position:absolute
   display:block
}
```

Car.ts

```
class Car {
    this.div:HTMLElement
    this.x = 100

    constructor(){
        this.div = document.createElement("car")
        document.body.appendChild(this.div)
    }
    public update(){
        this.x++
        this.div.style.transform = `translate(${this.x}px 20px)`
    }
}
```

Bounding box

Een DOM rectangle is een object met acht properties: **left, top, right, bottom, x, y, width, height**
Deze kan je gebruiken om precies te zien waar het object staat en hoe groot het is.
De rectangle houdt rekening met CSS transforms.

```
let div : HTMLElement = document.getElementById("car")
let rectangle : ClientRect = div.getBoundingClientRect()
```

Collision

Je kan deze collision functie gebruiken om te zien of twee rectangles overlappen

```
function intersects(a: ClientRect, b: ClientRect) {
  return (a.left <= b.right &&
          b.left <= a.right &&
          a.top <= b.bottom &&
          b.top <= a.bottom)
}
```

Aanroepen doe je door de rectangles van je game objecten mee te geven:

```
let carOne = car1.div.getBoundingClientRect()
let carTwo = car2.div.getBoundingClientRect()

console.log(intersects(carOne, carTwo))      // true or false
```

Vector

Een vector is een variabele die een **x** en **y** positie bevat. Een vector is handig als je veel berekeningen moet doen die met de positie te maken hebben.

```
let position = new Vector2(200,300)
```

Het berekenen van de afstand tussen twee punten:

```
let distance = position1.difference(position2)
```

Je kan vector ook gebruiken als snelheid. Dit object beweegt diagonaal, want de x snelheid is 1 en de y snelheid is 1

```
let speed = new Vector2(1,1)
```

Je kan in elk animatieframe de speed bij de positie van het object optellen met de **add()** functie.

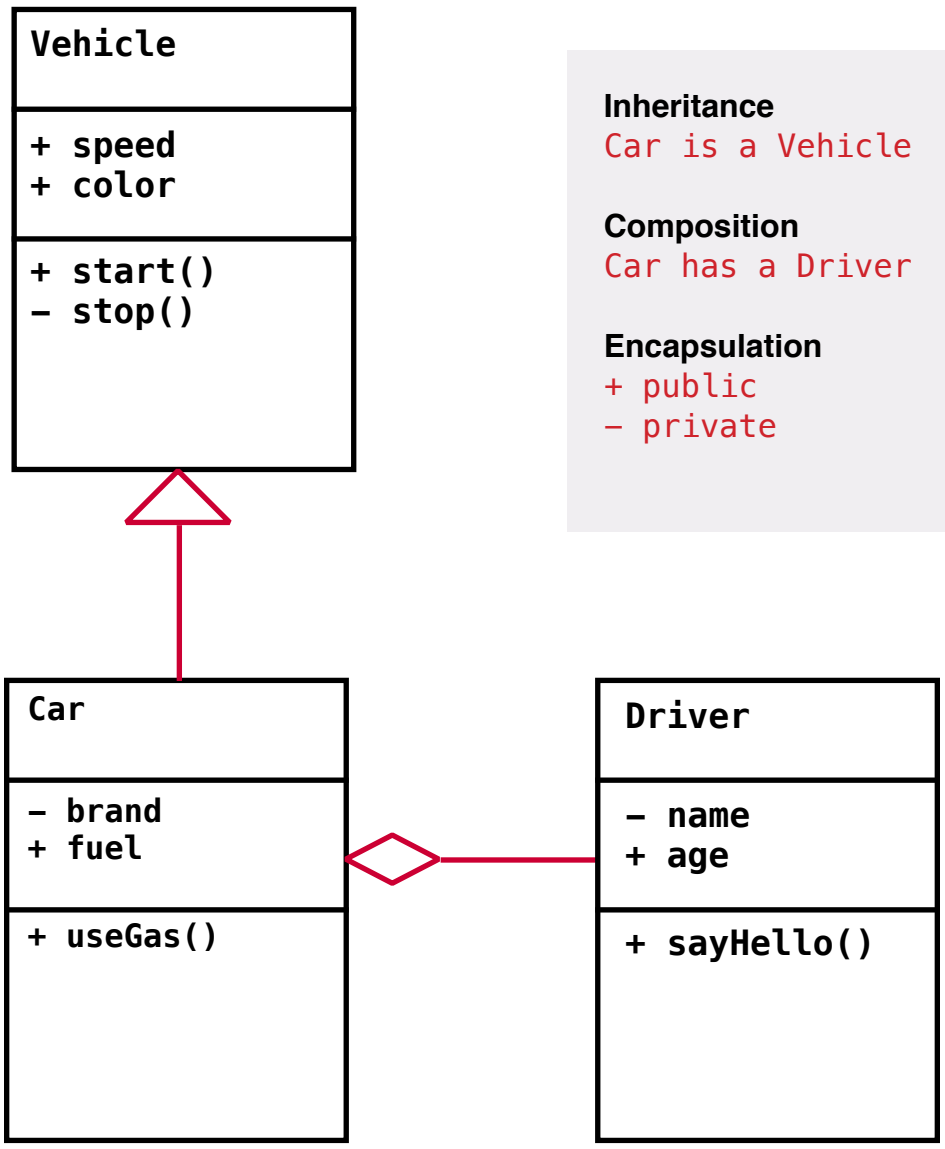
```
position = position.add(speed)
```

Door eerst **normalize()** te gebruiken is een diagonaal bewegend object niet sneller dan een horizontaal of verticaal bewegend object.

```
let normalizedSpeed = speed.normalize()
```

Lees alles over de Vector class op GitHub.

Klassendiagram



Inheritance
Car is a Vehicle

Composition
Car has a Driver

Encapsulation
+ public
– private

Links

Typescript startproject
https://github.com/HR-CMGT/Typescript-startproject

VS Code
https://code.visualstudio.com

XAMPP
https://www.apachefriends.org/index.html

Localhost op Mac High Sierra
https://websitebeaver.com/set-up-localhost-on-macos-high-sierra-apache-mysql-and-php-7-with-sslhttps

Localhost op Windows 10
https://www.youtube.com/watch?v=6LjpyHoXVjo

Javascript
https://webapplog.com/es6/
http://es6-features.org/

Typescript
https://www.typescriptlang.org
https://basarat.gitbooks.io/typescript/

Git documentation
https://www.atlassian.com/git

Practice git
https://try.github.io/levels/1/challenges/1

Student plan GitHub
https://help.github.com/articles/applying-for-a-student-developer-pack/

Live Server Extension
https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer