TypeScript

- Linguaggio di programmazione, superset di JavaScript
- Nato nel 2012 (Anders Hejlsberg @ Microsoft) “JavaScript that scales”

  https://www.typescriptlang.org/
- Ben supportato da VS Code via Node JS
- Installazione via npm
  - npm install -g typescript
- tsc compila codice TypeScript in JavaScript (source to source compiler, transcompiler, o transpiler)
- Progetto di riferimento
  - https://github.com/egalli64/ntsi

Hello TypeScript

- Creazione di un file TypeScript
- Transpiler: genera il corrispondente JS
- Esecuzione dello script JS via Node
- Il file di configurazione tsconfig.json automatizza la generazione dei file .js via tsc
  - tsc --init
  - tsc --watch
- Via VS Code
  - Control-shift-P, Configure Default Build Task, tsc:watch
  - Control-shift-B per far partire la build (.vscode/tasks.json) in terminal
  - Debug, Start Debugging (F5) – richiede sourceMap in configurazione

tsc hello.ts

node hello.js

```
function greetings(name: string) {
    return "Hello, " + name;
}

console.log(greetings("TypeScript"));
```

type annotation opzionale

```
function greetings(name) {
    return "Hello, " + name;
}
console.log(greetings("TypeScript"));
```

```
{
    "compilerOptions": {
        "target": "es5",
        "module": "commonjs",
        "sourceMap": true,
        "outDir": "./out",
        // ...
    }
}
```

Tipi

- Type-checking (opzionale) per scrivere e leggere più facilmente il codice
- Tipizzazione statica, specificata al momento della dichiarazione
  - let i: number = 42;
  - function hello(name: string): string { /* … */ }

- Tipi primitivi JS
  - boolean
  - number
  - string
- Array
  - type[] o Array<type>
  - let values: number[] = [42, 12];
- any /* ogni valore è ammissibile */
- Tupla
  - [type1, type2]
  - let couple: [string, number] = ['hi', 42];
- Enumeration
  - enum Role { Model, View, Controller };
  - let role: Role = Role.View;
- void
  - Funzione che non ha un return type

class

```
class Person {
    private first: string;
    private last: string;

    constructor(first: string, last: string) {
        this.first = first;
        this.last = last;
    }

    fullInfo(): string {
        return this.first + ' ' + this.last;
    }
}
```

visibilità membri:
public (default)
o private

```
let john = new Person('John', 'Doe');
console.log(john.fullInfo());
```

Ereditarietà

```
class Employee extends Person {
    private salary: number;

    constructor(first: string, last: string, salary: number) {
        super(first, last);
        this.salary = salary;
    }

    fullInfo(): string {
        return super.fullInfo() + ': ' + this.salary;
    }
}
```

```
let jon = new Employee('Jon', 'Voight', 2000);
console.log(jon.fullInfo());
```

interface

Possono essere usate per indicare quali data member mi aspetto

```
interface User {
    first: string;
    last: string;
}

let tom: User = {
    first: 'Tom',
    last: 'Jones'
};
```

```
interface Message {
    sender: string,
    recipient: string,
    subject: string,
    message?: string
}
```

optional

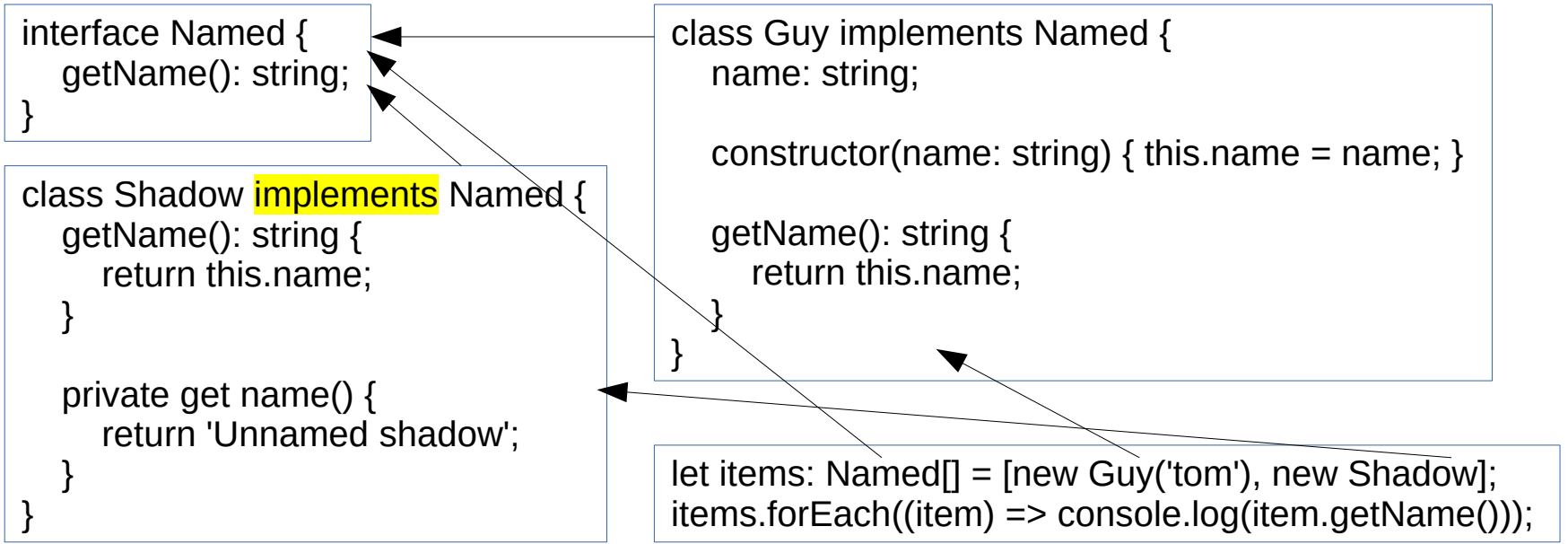
inline type annotation

```
let bob: {
    first: string;
    last: string;
} = {
    first: 'Bob',
    last: 'Coe'
};
```

```
function sayHello(user: User, message: Message) {
    // ...
}
```

Interfacce e classi

In ambito Object-Oriented l'interfaccia ha più propriamente lo scopo di dichiarare le funzionalità richiamabili sulle classi che la implementano



generic

Cfr: Array<T>.reverse()

```
function reverseCopy<T>(data: T[ ]): T[ ] {
    let result = [ ];
    for (let i = data.length - 1; i >= 0; i--) {
        result.push(data[i]);
    }
    return result;
}
```

Modulo

- ES 6, ma correntemente non implementato
- Uno script è un modulo se ha almeno un import o un export
- By default il contenuto di un modulo è privato
- export → permette l'accesso da altri file
- import → dichiara l'accesso ad altri file

```
export function hello(): void {
    console.log('hello export');
}
```


```
import { hello } from './s09exp';

hello();
```

Moduli

- Definizione ed esportazione anche separate
- Alias in esportazione o importazione con ‘as’
- Default export
- Full import

```
function hello(): void {
    console.log('hello export');
}

function local(): void {
    console.log('hello local');
}

function bye(): void {
    console.log('bye');
}

export { hello, bye as goodbye }
```

```
export default function hi(): void {
    console.log('hi');
}
```

```
import hi from './s10exp2';

hi();
```

```
import * as cheers from './s10exp';

cheers.hello();
cheers.goodbye();
```

```
import { hello as hi, goodbye } from './s10exp';

hi();
goodbye();
```

Decorator

- ES Stage 2 TC 39, TS sperimentale
- Usato in Angular, i decorator hanno @ come prefisso
- Python decorator, Java annotation
- Applicabile a una classe o ai suoi membri
- Permettono agli elementi associati di essere
  - identificati come appartenenti ad una certa categoria
  - configurati