Agenda

- TypeScript Overview
- Features
- Using TypeScript in VS Code
- Type Acquisition
- The TypeScript Compiler
- Project Configuration
- Build Tasks
- Create an Object Model

TypeScript Overview

Features

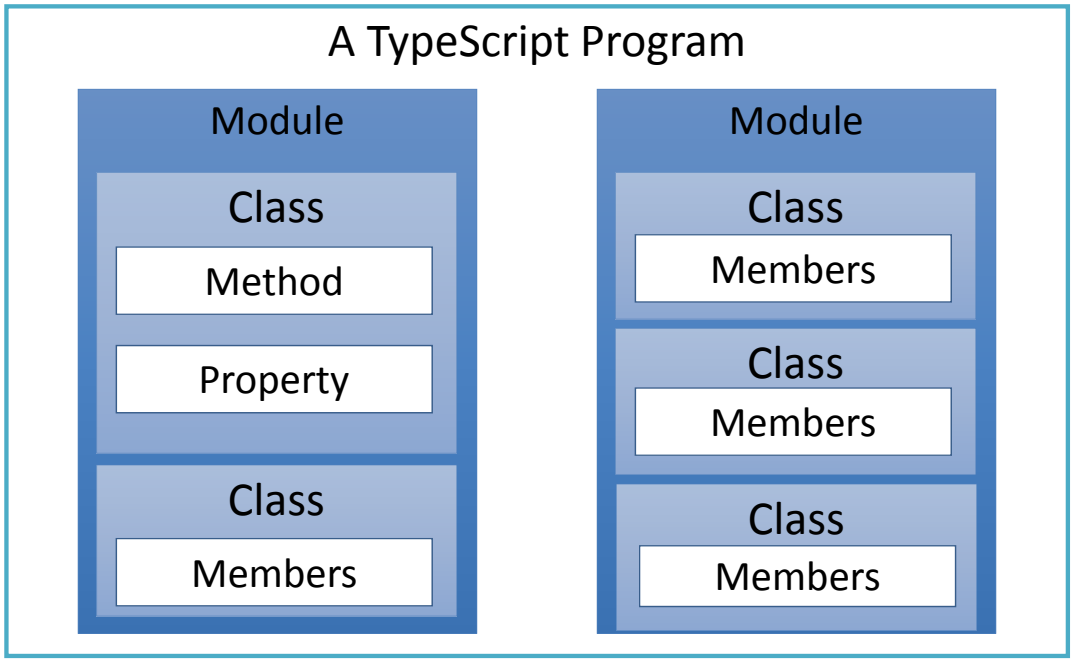
- Non-nullable types
- Control flow analysis
- Discriminated Union types
- Never types
- Read-only properties
- This types for functions
- Glob support in tsconfig
- New module resolution
- Quick ambient modules
- @types .d.ts acquisition
- UMD module definitions
- Optional class properties
- Private constructors
- More…

The TypeScript Transpiler

tsc.exe

tsc -p ./path-to-project-directory

https://www.typescriptlang.org/docs/handbook/compiler-options.html



Type Acquisition

- Definitely Typed (types repository)
- TSD (Package mgr for type definition files, deprecated)
- Typings (.d.ts registry)
- @types (for npm)

Type Acquisition

https://github.com/typings/typings
https://github.com/DefinitelyTyped/tsd
https://www.npmjs.com/~types

Compilation

- .tsconfig to set compiler configurations
- https://basarat.gitbooks.io/typescript/docs/project/tsconfig.html

The TypeScript Language

Types

TypeScript Types

Primitive and Object
Any
Number
Boolean
String
Null *
Undefined *
Object
Void *
HTMLElement
Functions
Enum
Never

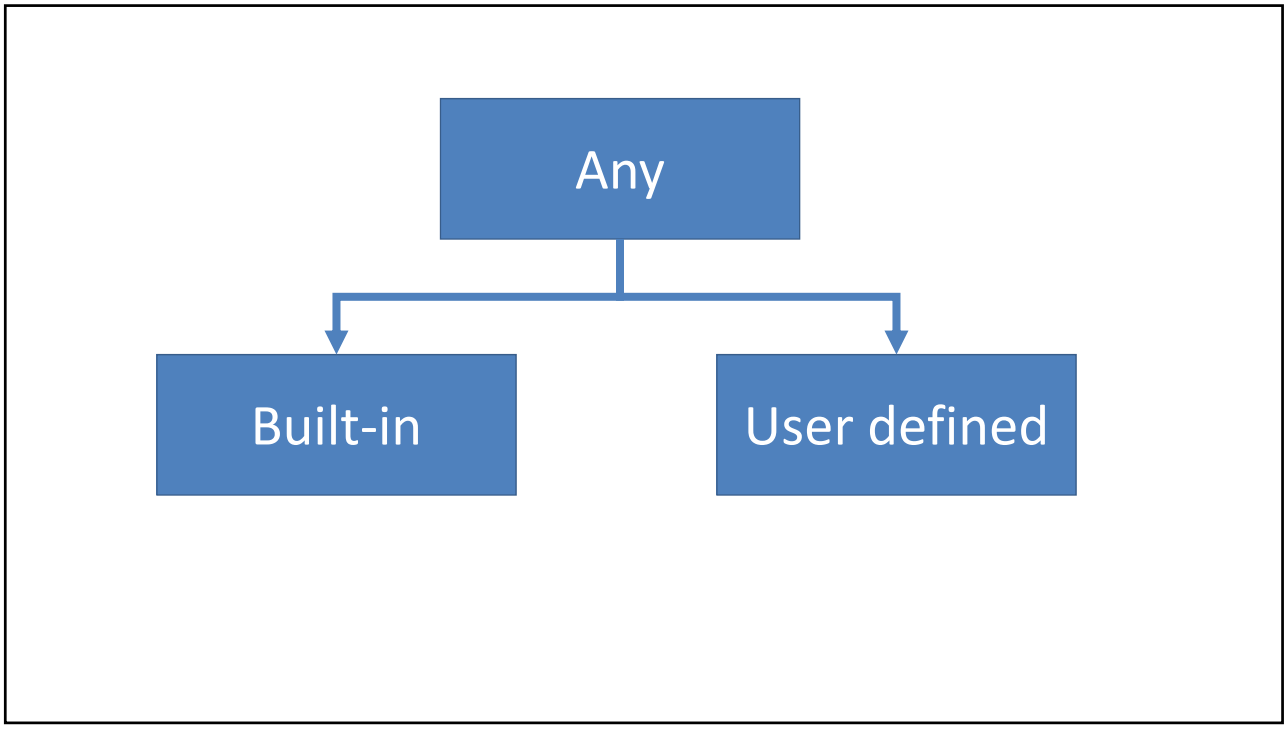
Declaring types

var

let

Any type

```
let notSure: any = 4;
notSure = "maybe a string instead";
notSure = false; // okay, definitely a boolean
```



Type annotations

Argument types

Return types

Type inference

Boolean

```
var boolArray: boolean[];

boolArray = [true, false];
console.log(boolArray[0]); // true
console.log(boolArray.length); // 2
boolArray[1] = true;
boolArray = [false, false];

boolArray[0] = 'false'; // Error!
boolArray = [true, 'false']; // Error!
```

Enums

```
enum Color {Red, Green, Blue}
let c: Color = Color.Green;
```

Interfaces

```
interface Name {
   first: string;
   second: string;
}

var name: Name;
name = {
   first: 'John',
   second: 'Doe'
};
```

Tuples

```
// Declare a tuple type
let x: [string, number];

// Initialize
x = ["hello", 10];

console.log(x[0].substr(1));
```

Null and undefined

- Null: value is unknown.
- Undefined: value has not been set yet.

Void

```
function warnUser(): void { alert("This is my warning message"); }

let unusable: void = undefined;
```

Never

The never type is a subtype of, and assignable to, every type; however, no type is a subtype of, or assignable to, never (except never itself). Even *any* isn't assignable to never.

Type Assertions

<> or "as"

```
let someValue: any = "this is a string";
let strLength: number = (<string>someValue).length;

let someValue: any = "this is a string";
let strLength: number = (someValue as string).length;
```

Type Alias

```
type StringOrNumber= string|number;

// Usage: just like any other notation
var sample: StringOrNumber;
sample = 123;
sample = '123';

sample = true; // Error! Because that's a Boolean!
```

Read only

```
function foo(config: {
   readonly bar: number,
   readonly bas: number
}

let config = { bar: 123, bas: 123 };
foo(config);
```

OOP TypeScript

Overloads

```
declare function fn(x: HTMLDivElement): string;
declare function fn(x: HTMLElement): number;
declare function fn(x: any): any;

var myElem: HTMLDivElement;
var x = fn(myElem); // x: string,
```

Create an Object Model

- Object Orientation
- Classes
  - Getters/setters
  - Methods
  - Types
- Interfaces
- Abstraction
- Encapsulation
- Inheritance

Classes

```
class BankAccount {}
```

TS

```
var BankAccount = (function () {
    function BankAccount() { }
    return BankAccount;
})();
```

JS

Classes: Members

```
deposit(amount: number) {
    this.Balance += amount;
}

calculateInterest() : number {
    this.Balance = (this.Balance * this.InterestRate);
    return this.Balance;
}
```

TS

```
BankAccount.prototype.deposit = function (amount) {
    this.Balance += amount;
};
BankAccount.prototype.calculateInterest = function () {
    this.Balance = (this.Balance * this.InterestRate);
    return this.Balance;
};
```

JS

Inheritance

```
class CheckingAccount extends BankAccount {}
```

TS

```
var CheckingAccount = (function (_super) {
    __extends(CheckingAccount, _super);
    function CheckingAccount() {
        _super.apply(this, arguments);

    }
    return CheckingAccount;
})(BankAccount);
```

JS

Classes: Access modifiers

```
class BankAccount {
    public AccountHolderName: string;
    public Balance: number;
    private InterestRate: number;

    public deposit(amount: number) {
        this.Balance = this.Balance + amount;
    }

    public calculateInterest() : number {
        this.Balance = (this.Balance * this.InterestRate);
        return this.Balance;
    }
}
```

Accessing a Class and its members

```
window.onload = () => {
    var elem = document.getElementById('content');
    var account = new BankAccount();
    account.deposit(500);
    elem.innerText = account.Balance.toString();
};
```

Thank you