

# Creating and Using Custom Types

---



**Brice Wilson**

@brice\_wilson [www.BriceWilson.net](http://www.BriceWilson.net)



# Overview



**Interfaces**

**Classes**

**Supporting multi-file projects**



# Interfaces vs. Classes

## Interfaces

Define a new type

Properties (signatures)

Methods (signatures)

Cannot be instantiated

## Classes

Define a new type

Properties (with implementation)

Methods (with implementation)

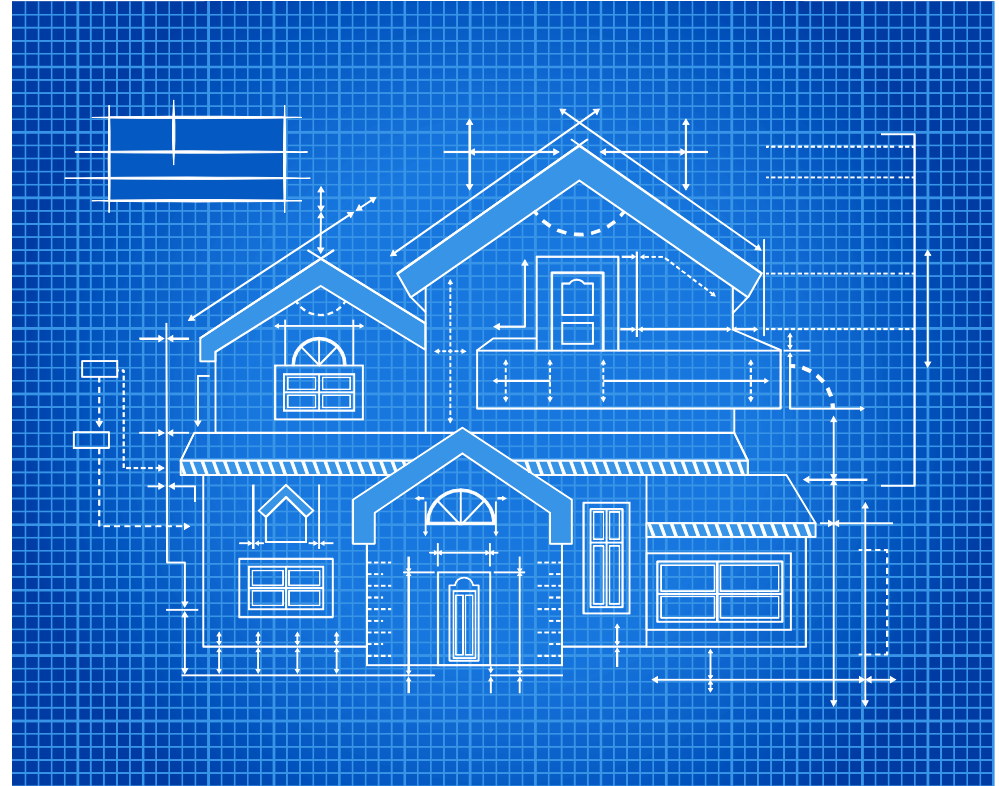
Can be instantiated



# Interfaces vs. Classes



**Interface**



**Class**

# Creating an Interface

```
interface Employee {
```



```
}
```



# Creating an Interface

```
interface Employee {  
    name: string;  
    title: string;  
}
```



# Creating an Interface

```
interface Employee {  
    name: string;  
    title: string;  
}
```


```
interface Manager extends Employee {  
    department: string;  
    numOfEmployees: number;  
}
```



# Creating an Interface

```
interface Employee {  
    name: string;  
    title: string;  
}
```

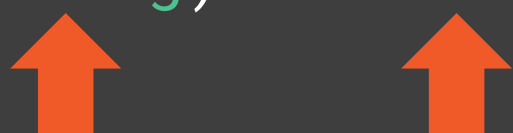
```
interface Manager extends Employee {  
    department: string;  
    numOfEmployees: number;  
}
```





# Creating an Interface

```
interface Employee {  
    name: string;  
    title: string;  
}  
  
interface Manager extends Employee {  
    department: string;  
    numOfEmployees: number;  
    scheduleMeeting: (topic: string) => void;  
}
```



# TypeScript's Structural Type System

```
interface Employee {  
    name: string;  
    title: string;  
}  
  
let developer = {  
    name: 'Michelle',  
    title: 'Senior TypeScript Developer',  
    editor: 'Visual Studio Code'  
}
```



# TypeScript's Structural Type System

```
interface Employee {  
    name: string;  
    title: string;  
}
```

```
let developer = {  
    name: 'Michelle',  
    title: 'Senior TypeScript Developer',  
    editor: 'Visual Studio Code'  
}
```



# TypeScript's Structural Type System

```
interface Employee {  
    name: string;  
    title: string;  
}  
  
let developer = {  
    name: 'Michelle',  
    title: 'Senior TypeScript Developer',  
    editor: 'Visual Studio Code'  
}
```

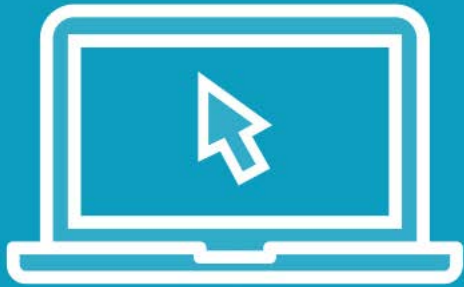


# TypeScript's Structural Type System

```
interface Employee {  
    name: string;  
    title: string;  
}  
  
let developer = {  
    name: 'Michelle',  
    title: 'Senior TypeScript Developer',  
    editor: 'Visual Studio Code'  
}  
  
let newEmployee: Employee = developer;
```



# Demo



## Creating interfaces



# Class Members

**Method implementations**

**Property implementations**

**Accessors (getters and setters)**

**Access modifiers**

- Public
- Private
- Protected



# Class Members

```
class Developer {  
    department: string;  
    private _title: string;
```

```
}
```





# Class Members

```
class Developer {  
    department: string;  
    private _title: string;  
    get title(): string {  
        return this._title;  
    }  
    set title(newTitle: string) {  
        this._title = newTitle.toUpperCase();  
    }  
    documentRequirements(requirements: string): void {  
        console.log(requirements);  
    }  
}
```



# Class Members

```
class Developer {  
    department: string;  
    private _title: string;  
    get title(): string {  
        ↑ return this._title;  
    }  
    set title(newTitle: string) {  
        this._title = newTitle.toUpperCase();  
    }  
    documentRequirements(requirements: string): void {  
        console.log(requirements);  
    }  
}
```



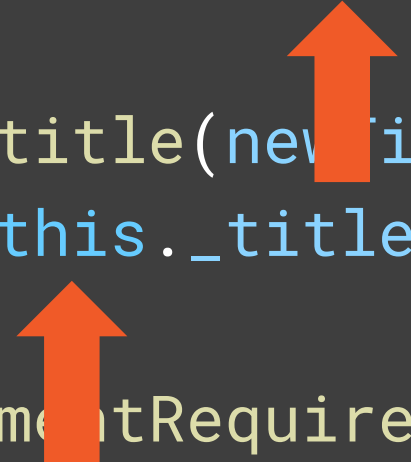
# Class Members

```
class Developer {  
    department: string;  
    private _title: string;  
    get title(): string {  
        return this._title;  
    }  
    set title(newTitle: string) {  
        ↑ this._title = newTitle.toUpperCase();  
    }  
    documentRequirements(requirements: string): void {  
        console.log(requirements);  
    }  
}
```



# Class Members

```
class Developer {  
    department: string;  
    private _title: string;  
    get title(): string {  
        return this._title;  
    }  
    set title(newTitle: string) {  
        this._title = newTitle.toUpperCase();  
    }  
    documentRequirements(requirements: string): void {  
        console.log(requirements);  
    }  
}
```



# Extending a Class

```
class WebDeveloper extends Developer {
```



```
}
```



# Extending a Class

```
class WebDeveloper extends Developer {  
    favoriteEditor: string;  
    writeTypeScript(): void {  
        // write awesome code  
    }  
}
```



# Extending a Class

```
class WebDeveloper extends Developer {  
    favoriteEditor: string;  
    writeTypeScript(): void {  
        // write awesome code  
    }  
}  
  
let webdev: WebDeveloper = new WebDeveloper();
```



# Extending a Class

```
class WebDeveloper extends Developer {  
    favoriteEditor: string;  
    writeTypeScript(): void {  
        // write awesome code  
    }  
}  
  
let webdev: WebDeveloper = new WebDeveloper();  
webdev.department = 'Software Engineering';  
webdev.favoriteEditor = 'Visual Studio Code';
```





# Implementing an Interface

```
interface Employee {  
    name: string;  
    title: string;  
    logID: () => string;  
}
```



# Implementing an Interface

```
interface Employee {  
    name: string;  
    title: string;  
    logID: () => string;  
}  
class Engineer implements Employee {  
    name: string;  
    title: string;  
  
}
```



# Implementing an Interface

```
interface Employee {  
    name: string;  
    title: string;  
    logID: () => string;  
}
```

```
class Engineer implements Employee {  
    name: string;  
    title: string;
```



```
}
```

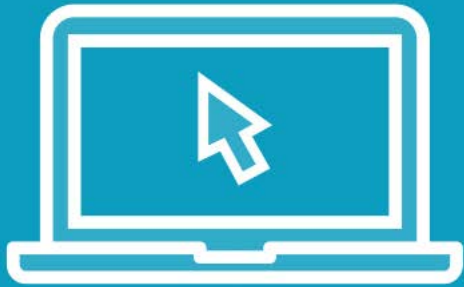


# Implementing an Interface

```
interface Employee {  
    name: string;  
    title: string;  
    logID: () => string;  
}  
  
class Engineer implements Employee {  
    name: string;  
    title: string;  
    logID() {  
        return `${this.name}_${this.title}`;  
    }  
}
```



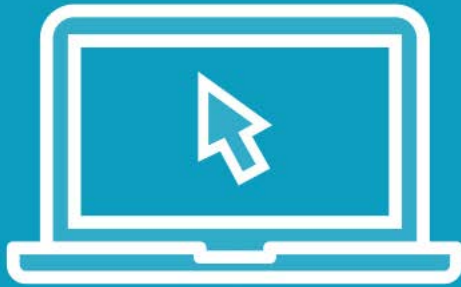
# Demo



## Creating classes



# Demo



Configuring a project with multiple source files



# Static Members

```
class WebDeveloper extends Developer {  
    static jobDescription: string = 'Build cool things!';  
    static logFavoriteProtocol() {  
        console.log('HTTPS, of course!');  
    }  
}
```



# Static Members

```
class WebDeveloper extends Developer {  
    → static jobDescription: string = 'Build cool things!';  
    → static logFavoriteProtocol() {  
        console.log('HTTPS, of course!');  
    }  
  
}
```





# Static Members

```
class WebDeveloper extends Developer {  
    static jobDescription: string = 'Build cool things!';  
    static logFavoriteProtocol() {  
        console.log('HTTPS, of course!');  
    }  
    logJobDescription(): void {  
        console.log(WebDeveloper.jobDescription);  
    }  
}
```



# Static Members

```
class WebDeveloper extends Developer {  
    static jobDescription: string = 'Build cool things!';  
    static logFavoriteProtocol() {  
        console.log('HTTPS, of course!');  
    }  
    logJobDescription(): void {  
        console.log(WebDeveloper.jobDescription);  
    }  
}
```



# Static Members

```
class WebDeveloper extends Developer {  
    static jobDescription: string = 'Build cool things!';  
    static logFavoriteProtocol() {  
        console.log('HTTPS, of course!');  
    }  
    logJobDescription(): void {  
        console.log(WebDeveloper.jobDescription);  
    }  
}  
  
WebDeveloper.logFavoriteProtocol();
```



# Constructors

```
class Developer {  
    constructor() {  
  
    }  
}
```



# Constructors

```
class Developer {  
    constructor() {  
    }  
}
```



# Constructors

```
class Developer {  
    constructor() {  
        console.log('Creating a new developer.');    }  
}
```



# Constructors

```
class Developer {  
    constructor() {  
        console.log('Creating a new developer.');    }  
}  
  
class WebDeveloper extends Developer {  
    readonly favoriteEditor: string;  
    constructor(editor: string) {  
        super();  
        this.favoriteEditor = editor;  
    }  
}
```



# Constructors

```
class Developer {  
    constructor() {  
        console.log('Creating a new developer.');    }  
}  
  
class WebDeveloper extends Developer {  
    readonly favoriteEditor: string;  
    constructor(editor: string) {  
        super();  
        this.favoriteEditor = editor;  
    }  
}
```





# Constructors

```
class Developer {  
    constructor() {  
        console.log('Creating a new developer.');    }  
}  
  
class WebDeveloper extends Developer {  
    readonly favoriteEditor: string;  
    constructor(editor: string) {  
        → super();  
        this.favoriteEditor = editor;  
    }  
}
```



# Constructors

```
class Developer {  
    constructor() {  
        console.log('Creating a new developer.');    }  
}  
  
class WebDeveloper extends Developer {  
    readonly favoriteEditor: string;  
    constructor(editor: string) {  
        super();  
        this.favoriteEditor = editor;  
    }  
}
```



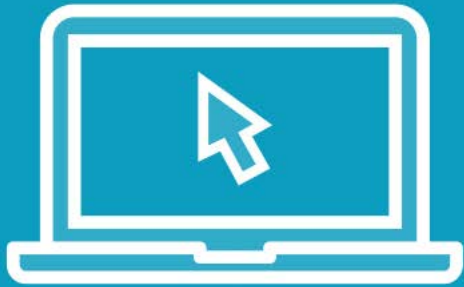
# Constructors

```
class Developer {  
    constructor() {  
        console.log('Creating a new developer.');    }  
}
```

```
class WebDeveloper extends Developer {  
    readonly favoriteEditor: string;  
    constructor(editor: string) {  
        super();  
        this.favoriteEditor = editor;  
    }  
}
```



# Demo



## Refactoring the demo app with classes



# Summary



**Interfaces and Classes**

**Structural type system**

**Supporting multiple source files**

**Flexibility**

