

TypeScript 3.2 is now available. [Download \(/#download-links\)](#) our latest version today!

Documentation ▼

Mixins

Introduction

Along with traditional OO hierarchies, another popular way of building up classes from reusable components is to build them by combining simpler partial classes. You may be familiar with the idea of mixins or traits for languages like Scala, and the pattern has also reached some popularity in the JavaScript community.

Mixin sample

In the code below, we show how you can model mixins in TypeScript. After the code, we'll break down how it works.

```

// Disposable Mixin
class Disposable {
    isDisposed: boolean;
    dispose() {
        this.isDisposed = true;
    }
}

// Activatable Mixin
class Activatable {
    isActive: boolean;
    activate() {
        this.isActive = true;
    }
    deactivate() {
        this.isActive = false;
    }
}

class SmartObject implements Disposable, Activatable {
    constructor() {
        setInterval(() => console.log(this.isActive + " : " + this.isDisposed), 500);
    }

    interact() {
        this.activate();
    }

    // Disposable
    isDisposed: boolean = false;
    dispose: () => void;
    // Activatable
    isActive: boolean = false;
    activate: () => void;
    deactivate: () => void;
}
applyMixins(SmartObject, [Disposable, Activatable]);

let smartObj = new SmartObject();
setTimeout(() => smartObj.interact(), 1000);

////////////////////////////////////
// In your runtime library somewhere
////////////////////////////////////

function applyMixins(derivedCtor: any, baseCtors: any[]) {
    baseCtors.forEach(baseCtor => {
        Object.getOwnPropertyNames(baseCtor.prototype).forEach(name => {
            derivedCtor.prototype[name] = baseCtor.prototype[name];
        });
    });
}

```

Understanding the sample

The code sample starts with the two classes that will act as our mixins. You can see each one is focused on a particular activity or capability. We'll later mix these together to form a new class from both capabilities.

```
// Disposable Mixin
class Disposable {
    isDisposed: boolean;
    dispose() {
        this.isDisposed = true;
    }
}

// Activatable Mixin
class Activatable {
    isActive: boolean;
    activate() {
        this.isActive = true;
    }
    deactivate() {
        this.isActive = false;
    }
}
```

Next, we'll create the class that will handle the combination of the two mixins. Let's look at this in more detail to see how it does this:

```
class SmartObject implements Disposable, Activatable {
```

The first thing you may notice in the above is that instead of using `extends`, we use `implements`. This treats the classes as interfaces, and only uses the types behind `Disposable` and `Activatable` rather than the implementation. This means that we'll have to provide the implementation in class. Except, that's exactly what we want to avoid by using mixins.

To satisfy this requirement, we create stand-in properties and their types for the members that will come from our mixins. This satisfies the compiler that these members will be available at runtime. This lets us still get the benefit of the mixins, albeit with some bookkeeping overhead.

```
// Disposable
isDisposed: boolean = false;
dispose: () => void;
// Activatable
isActive: boolean = false;
activate: () => void;
deactivate: () => void;
```


Finally, we mix our mixins into the class, creating the full implementation.

```
applyMixins(SmartObject, [Disposable, Activatable]);
```

Lastly, we create a helper function that will do the mixing for us. This will run through the properties of each of the mixins and copy them over to the target of the mixins, filling out the stand-in properties with their implementations.

```
function applyMixins(derivedCtor: any, baseCtors: any[]) {
    baseCtors.forEach(baseCtor => {
        Object.getOwnPropertyNames(baseCtor.prototype).forEach(name => {
            derivedCtor.prototype[name] = baseCtor.prototype[name];
        });
    });
}
```

Made with ❤ in Redmond Follow @Typescriptlang (<https://twitter.com/typescriptlang>)

Privacy (<https://go.microsoft.com/fwlink/?LinkId=521839>) ©2012-2018 Microsoft  Microsoft