Using Advanced Type Features

Brice Wilson

@brice_wilson www.BriceWilson.net

Overview

Polymorphic *this* types

Declaration merging

Type guards

Symbols

Polymorphic *this* Types

polymorphic *this* types

A polymorphic *this* type represents a type that is the *subtype* of the containing class or interface.

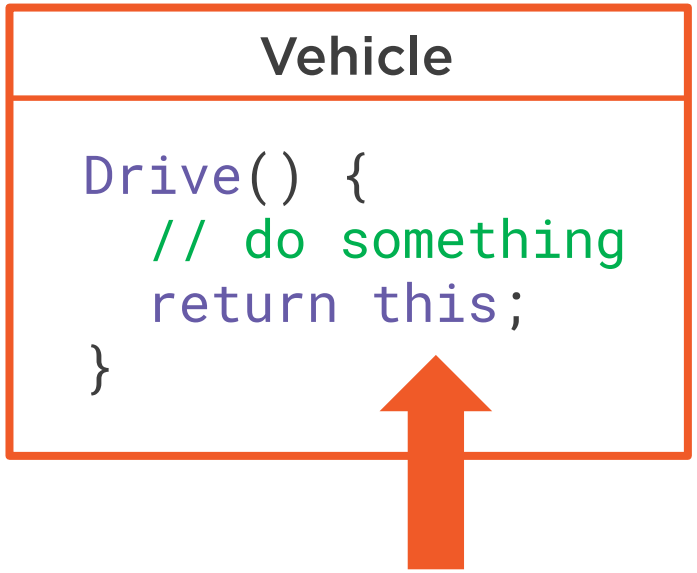
TypeScript Handbook
http://www.typescriptlang.org/docs/handbook/advanced-types.html

Polymorphic *this*

Vehicle

```
Drive() {
  // do something
  return this;
}
```

Polymorphic *this*




```
let v = new Vehicle();
v.Drive(); // returns Vehicle
```

Polymorphic *this*

Vehicle

```
Drive() {
  // do something
  return this;
}
```

Car

```
CarryPeople() {
  // do something
  return this;
}
```

```
let c = new Car();
c.Drive(); // returns Car
```

Polymorphic *this*

Vehicle

```
Drive() {
  // do something
  return this;
}
```

Car

```
CarryPeople() {
  // do something
  return this;
}
```

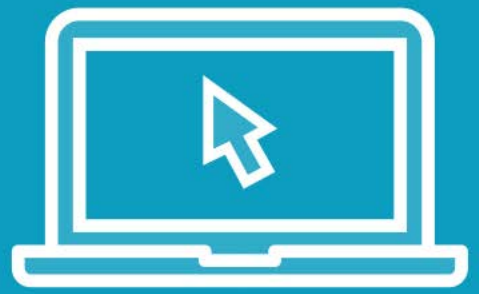
Truck

```
CarryCargo() {
  // do something
  return this;
}
```

```
let t = new Truck();

t.Drive(); // returns Truck
```

Demo



Creating a fluent API with polymorphic *this* types

Declaration Merging

declaration merging

The compiler merges two separate declarations declared with the same name into a single definition.

TypeScript Handbook
http://www.typescriptlang.org/docs/handbook/declaration-merging.html

Declaration Merging

```
interface Employee {
    name: string;
    doWork: () => void;
}
```

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

TypeScript Compiler

```
interface Employee {
    name: string;
    doWork: () => void;
    title: string;
    phone: string;
}
```

Can I Merge It?

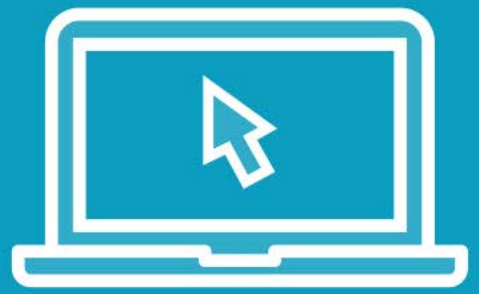
Allowed merges

- Interfaces
- Enums
- Namespaces
- Namespaces with classes
- Namespaces with functions
- Namespaces with enums

Disallowed merges

- Classes with classes

Demo



Interface merging and module augmentation

Type Guards

```
let x: string | number = 123;
```

typeof Type Guards

Uses JavaScript *typeof* operator

Compares result of *typeof* operator to a type name

Type name may only be “string”, “number”, “boolean”, or “symbol”

```
let x: string | number = 123;
if (typeof x === 'string') {
    // x is a string
}
else {
    // x is a number
}
```

typeof Type Guards

Uses JavaScript *typeof* operator

Compares result of *typeof* operator to a type name

Type name may only be “string”, “number”, “boolean”, or “symbol”

instanceof Type Guards

```
class Phone {
    callSomeone() { console.log('make call'); }
}
class Tablet {
    watchMovie() { console.log('watch movie'); }
}
let device: Phone | Tablet = new Phone();
```

instanceof Type Guards

```
class Phone {
    callSomeone() { console.log('make call'); }
}
class Tablet {
    watchMovie() { console.log('watch movie'); }
}
let device: Phone | Tablet = new Phone();
if (device instanceof Phone) {

}
```

instanceof Type Guards

```
class Phone {
    callSomeone() { console.log('make call'); }
}
class Tablet {
    watchMovie() { console.log('watch movie'); }
}
let device: Phone | Tablet = new Phone();
if (device instanceof Phone) {
    device.callSomeone();
}
```

User-Defined Type Guards

```
interface Vehicle { numberOfWheels: number; }
function isVehicle(v: any): v is Vehicle {


}
```

User-Defined Type Guards

```
interface Vehicle { numberOfWheels: number; }
function isVehicle(v: any): v is Vehicle {


}
```

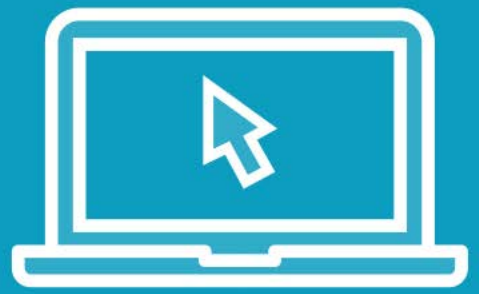
User-Defined Type Guards

```
interface Vehicle { numberOfWheels: number; }
function isVehicle(v: any): v is Vehicle {
    return (<Vehicle>v).numberOfWheels !== undefined;
}
```

User-Defined Type Guards

```
interface Vehicle { numberOfWheels: number; }
function isVehicle(v: any): v is Vehicle {
    return (<Vehicle>v).numberOfWheels !== undefined;
}
let c = new Car();
if(isVehicle(c)) {
    // it's a Vehicle
}
```

Demo



Creating and using type guards

Symbols

What Is a Symbol?

ES2015 feature

Primitive data type

Unique

Immutable

Why?

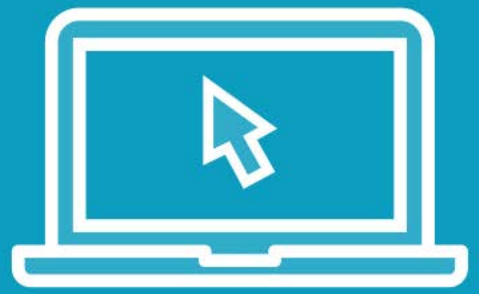
Symbol Use Cases

Unique Constants

Computed Property Declarations

Customize Internal Language Behavior

Demo



Experimenting with symbols

Summary

Fluent APIs with polymorphic *this* types

Declaration merging to augment modules

Writing better code with type guards

Practical use of Symbols