Type Guards

This lesson introduces the concept of type guards.

WE'LL COVER THE FOLLOWING

Overview

Type guard is not really a *type* so much as it is a mechanism that *narrows types*.

TypeScript soars at analyzing your code and deducing information based on it. Oftentimes, TypeScript can analyze a conditional expression (or an `if` statement) and narrow the type of the tested value inside the branches of the condition. We've already seen this happen with discriminated unions. This mechanism is referred to as *type guard*.

Nullable type guard

We've already seen the nullable type guard in action when discussing `strictNullChecks`. Nullable type guard removes optionality from the type of some value if this value is previously tested for nullability. In the below example, if the `name` is truthy (i.e., not `null` or `undefined`), its type is narrowed to `string` inside the positive branch of the `if` statement.

```
function sayHello(name?: string) {
  if (name) {
    console.log(`Hello, ${name.toUpperCase()}!`);
  } else {
    console.log("Hello!");
```

```
  }
}
```

Hover over `name` inside the first `if` branch to see how its type is narrowed.

`typeof` type guard

Sometimes it's possible to check the type of a value at runtime. This type guard narrows the type of checked variable to the type it's checked against. This trick only works with JavaScript primitive types (string, number, undefined, null, Boolean, and symbol). You cannot use it to check the type of an interface because this information is erased at runtime.

In the below example, `width` can be either a `number` or a `string`. Not only the type of `width` is narrowed to a `number` in the first branch, but it's also narrowed to a `string` in the second branch.

```
class Properties {
  width: number = 0;

  setWidth(width: number | string) {
    if (typeof width === "number") {
      this.width = width;
    } else {
      this.width = parseInt(width);
    }
  }
}
```

Hover over `width` within both branches of the `if` statement to see the inferred types.

`instanceof` guard

This type guard can be used with classes. Contrary to interfaces, the type information for classes is retained at runtime. In other words, you can check if some object is an instance of some class at runtime. This type guard will narrow the type of that object to this class inside the conditional expression.

```
class Person {
  constructor(public name: string) {}
}

function greet(obj: any) {
  if (obj instanceof Person) {
    console.log(obj.name);
  }
}
```

Hover over `obj` within the `if` statement to see the inferred type.

The next lesson discusses how to create your own type guards.