Chapter 12: User-defined Type Guards

Section 12.1: Type guarding functions

You can declare functions that serve as type guards using any logic you'd like.

They take the form:

```
function functionName(variableName: any): variableName is DesiredType {
    // body that returns boolean
}
```

If the function returns true, TypeScript will narrow the type to `DesiredType` in any block guarded by a call to the function.

For example (try it):

```
function isString(test: any): test is string {
    return typeof test === "string";
}

function example(foo: any) {
    if (isString(foo)) {
        // foo is type as a string in this block
        console.log("it's a string: " + foo);
    } else {
        // foo is type any in this block
        console.log("don't know what this is! [" + foo + "]");
    }
}

example("hello world");          // prints "it's a string: hello world"
example({ something: "else" });  // prints "don't know what this is! [[object Object]]"
```

A guard's function type predicate (the `foo is Bar` in the function return type position) is used at compile time to narrow types, the function body is used at runtime. The type predicate and function must agree, or your code won't work.

Type guard functions don't have to use **`typeof`** or **`instanceof`**, they can use more complicated logic.

For example, this code determines if you've got a jQuery object by checking for its version string.

```
function isJQuery(foo): foo is JQuery {
    // test for jQuery's version string
    return foo.jquery !== undefined;
}

function example(foo) {
    if (isJQuery(foo)) {
        // foo is typed JQuery here
        foo.eq(0);
    }
}
```

Section 12.2: Using instanceof

`instanceof` requires that the variable is of type `any`.

This code (try it):

```
class Pet { }
class Dog extends Pet {
    bark() {
        console.log("woof");
    }
}
class Cat extends Pet {
    purr() {
        console.log("meow");
    }
}

function example(foo: any) {
    if (foo instanceof Dog) {
        // foo is type Dog in this block
        foo.bark();
    }

    if (foo instanceof Cat) {
        // foo is type Cat in this block
        foo.purr();
    }
}

example(new Dog());
example(new Cat());
```

prints

```
woof
meow
```

to the console.

Section 12.3: Using typeof

`typeof` is used when you need to distinguish between types `number`, `string`, `boolean`, and `symbol`. Other string constants will not error, but won't be used to narrow types either.

Unlike `instanceof`, `typeof` will work with a variable of any type. In the example below, `foo` could be typed as `number | string` without issue.

This code (try it):

```
function example(foo: any) {
    if (typeof foo === "number") {
        // foo is type number in this block
        console.log(foo + 100);
    }

    if (typeof foo === "string") {
```

```
        // foo is type string in this block
        console.log("not a number: " + foo);
    }
}

example(23);
example("foo");
```

prints

```
123
not a number: foo
```