

Strict Property Initialization

This lesson introduces another compiler flag, `strictPropertyInitialization`, which brings more type safety to code that uses classes.

WE'LL COVER THE FOLLOWING ^

- Definition
- Fixing the error
 - Constructor initialization
 - In-place initialization
 - Adjusting the type

Definition

The flag we're talking about is called `strictPropertyInitialization`. As usual, let's look at the definition from [the documentation](#). I think it's rather self-explanatory.

Ensure non-undefined class properties are initialized in the constructor.
This option requires `--strictNullChecks` be enabled in order to take effect.

Enabling the flag will force you to always initialize class properties or make it explicit in the type annotation that the property can be `undefined`.

Fixing the error

In the piece below, the `header` property produces an error because it's initialized neither in-place nor in the constructor.

```
class HeaderComponent {  
  // ⚠ Property 'header' has no initializer and is not definitely assigned in the constructor  
  header: string;  
}
```

```
render() {  
  return `<h1>${this.header.toUpperCase()}</h1>`;  
}  
}
```



Run the code to see the error caused by enabling `strictPropertyInitialization`.

There are three ways to get rid of this error. First, you can initialize the property in the constructor.

Constructor initialization

```
class HeaderComponent {  
  constructor(private header: string) {}  
  
  render() {  
    return `<h1>${this.header.toUpperCase()}</h1>`;  
  }  
}
```



Run the code to verify that there are no compile errors. `strictPropertyInitialization` enabled.

In-place initialization

Second, you can initialize `header` in-place.

```
class HeaderComponent {  
  header: string = 'hello';  
  
  render() {  
    return `<h1>${this.header.toUpperCase()}</h1>`;  
  }  
}
```



Run the code to verify that there are no compile errors. `strictPropertyInitialization` enabled.

Adjusting the type

Finally, you can explicitly mark `header` as possibly undefined. Note that this

will result in an error inside `render` because now you're trying to call a method on a possibly undefined object.

```
class HeaderComponent {
  header: string | undefined;

  render() {
    // ⚠ Error! Object is possibly `undefined`
    return `<h1>${this.header.toUpperCase()}</h1>`;
  }
}
```



Run the code to verify that line 6 results in an error. `'strictPropertyInitialization'` enabled.

The third option helps understand why this flag is related to `strictNullChecks`. Leaving a property uninitialized is essentially equal to lying about its type. We say that `header` is a `string`, but if we don't initialize it, it is `undefined` and not a `string`.

One could argue that this compiler behavior should be part of the `strictNullChecks` flag. However, strict property initialization checks could lead to cumbersome code. We'll see an example of this in the next lesson.

In the next lesson, we'll talk about an escape hatch for `strictNullChecks` and `strictPropertyInitialization` flags.