Nullable

This lesson explains the nullable mapped type.

WE'LL COVER THE FOLLOWING

Nullable Mapped Type

Another possibility with mapped type is to handle `null`. Similar to the optional with `Partial<T>` you can create your own `Nullable<T>`. To create your own type, the first step is often to build it with no generic (without `<T>`, a plain interface or type) and then adjust by adapting with a generic parameter (e.g. `<T>`).

For example, if we have the entity `Cat`, we might want to create a `NullableCat` type as a first step. The second step then would be to make it possible to null a `Dog` or any other object.

Intermediary step: Two interfaces into a specific nullable type

If the second step seems challenging, we can add an intermediary step to create the entity once with the null type (e.g. having a union that accepts the type or null) which might help to visualize the problem.

Lines 11, 12 and 13 add `null` to all fields from the former interface defined at lines 1 to 5.

```
interface Cat {
  age: number;
```

```
  weight: number;
  numberOfKitty: number;
}


const cat1: Cat = { age:1, weight: 2, numberOfKitty: 0 };

// NullableCat1 have union with the null type. It allows to visualize the dual type.
interface NullableCat1 {
  age: number | null;
  weight: number | null;
  numberOfKitty: number | null;
}
```

Intermediary Step with union of null

From the long-winded version of the `NullableCat` interface it is highly visible that every property is of the main type or null. The next step is to use the mapped type. The following code as at line 10 a mapped type where all fields of Cat (`P`) are unioned with `null`.

```
interface Cat {
  age: number;
  weight: number;
  numberOfKitty: number;
}

const cat1: Cat = { age: 1, weight: 2, numberOfKitty: 0 };

// The nullable cat is now a mapped type.
type NullableCat = { [P in keyof Cat]: Cat[P] | null };

const cat2: NullableCat = { age: 1, weight: null, numberOfKitty: null };
```

Mapped Type Strongly Associated to the interface Cat

Generic nullable mapped type

The second step is good in the sense that you avoid duplicating every field. In case of changes in the main type, `Cat`, you do no have to synchronize. However, if you need to have a similar nullable logic for an entity `Dog` the proliferation of mapped type would be cumbersome.

```
interface Cat {
  age: number;
  weight: number;
  numberOfKitty: number;
}

const cat1: Cat = { age: 1, weight: 2, numberOfKitty: 0 };

// The mapped type that goes beyond Cat with generic <T>
```

```
// The mapped type that goes beyond Cat with generic <T>
type Nullable<T> = { [P in keyof T]: T[P] | null };

const cat2: Nullable<T> = { age: 1, weight: null, numberOfKitty: null };
```

Mapped Type with Generic

The transformation is to remove all mention of a specific type by a generic identifier, e.g. `T`. Then, remove from the name the mention of the entity and add the generic bracket with the generic identifier.