Exclude

This lesson explains the exclude mapped type.

Exclude vs Extract

The `Exclude` mapped type has is similar to `Extract` in the sense that it builds a type by selecting several properties. However, contrary to `Extract`, `Exclude` takes all the properties from a type and removes the one specified instead of starting from nothing and adding the specified properties.

The following snippet of code produces the exact same code as the one created in the previous lesson with `Extract`.

```
interface Animal {
    name: string;
    gender: string;
    sound: string;
}
interface Human {
    name: string;
    gender: string;
    nickname: string;
}

// type LivingThing = Extract<keyof Animal, keyof Human>;
type LivingThing = Exclude<keyof Animal, "sound">;
function sayMyName(who: Record<LivingThing, string>): void {
    console.log(who.name +" is of type " + who.gender);
}
const animal: Animal = { name: "Lion", sound: "Rawwwhhh", gender: "Male" };
const human: Human = { name: "Jacob", nickname: "Jaco-bee", gender: "Boy" };
sayMyName(animal);
sayMyName(human);
```

The **line 12** is commented is the code from the `Extract` mapped type example. In this code, the **line 13** replaces the functionality of **line 12**. The **line 12** was getting the intersection of properties between the `Animal` and the `Human`.

As you can see, all `Animal` type properties are provided with `keyof` of the first generic parameter, and the second parameter subtracts the `sound` property. The end result is a new type called `LivingThing` that has two properties called `name` and `gender`. Instead of specifying these two types from the `Human` interface and to `Extract` them from `Animal`, `Exclude` goes the other way around by removing what is specified.

That being said, we could also use `keyof` on the second generic parameter and achieve the same result.

```
interface Animal {
    name: string;
    gender: string;
    sound: string;
}
interface Human {
    name: string;
    gender: string;
    nickname: string;
}

interface NoisyLivingSpecies{
    sound: string;
}
type LivingThing = Exclude<keyof Animal, keyof NoisyLivingSpecies>;
```

Exclude vs Pick

While the examples of `Extract` and `Exclude` borrow the `Record` mapped type, it is not a necessity. For example, we can create a `HumanWithoutNickname` type by using `Exclude` and `Pick`.

```
interface Animal {
    name: string;
    gender: string;
    sound: string;
}
interface Human {
    name: string;
    gender: string;
    nickname: string;
```

```
}

interface NoisyLivingSpecies{

    sound: string;
}
type LivingThing = Exclude<keyof Animal, keyof NoisyLivingSpecies>;
type HumanWithoutNickname = Pick<Human, LivingThing>;
```

The `Exclude` statement strips off the `sound` property of `Animal` . The `LivingThing` type is a union of the two remainings properties: `"name" | "gender"` .

Then, using these two strings, `Pick` creates a type composed of two named properties from the associated types in the `Human` interface. In this case, two strings.

This is interesting! Isn't it? The dynamic feature of mapped types allows for the creation of several types depending on which mapped type is used and on which type they are used against.

For example, the previous example created a `HumanWithoutNickname` of two properties defined by the `Exclude` with the type of `string` , because it was built upon type `Human` that has these two `string` properties. Let's modify the code to use `Pick` on a different human type.

```
interface Animal {
    name: string;
    gender: string;
    sound: string;
}
interface Human {
    name: string;
    gender: string;
    nickname: string;
}

interface Human2 {
    name: string;
    gender: boolean;
    intelligenceScore: number
}

interface NoisyLivingSpecies{
    sound: string;
}
type LivingThing = Exclude<keyof Animal, keyof NoisyLivingSpecies>;
type HumanWithoutNickname = Pick<Human, LivingThing>;
type HumanWithoutNickname2 = Pick<Human2, LivingThing>;
```

In that example, `Human2` has a property `gender` that is `boolean` instead of `string`. `Pick` takes this type along with the property and creates a `HumanWithoutNickname2` without the two `string` properties as before, but with `Record`.

Instead, `Pick` handles to map a string for the `name` and a boolean for the `gender`. Because the `Exclude` from `Animal` took only the name and gender members, the `intelligenceScore` is not mapped. Move your cursor over the two types (**lines 22** and **line 23**) to see the difference.