Generic Comparison

This lesson explains the difference between comparing two generic types and two primitive types.

WE'LL COVER THE FOLLOWING

Generic and `typeof`

The generic code does not allow the use of `typeof` on T, type `new T` or `instanceof T`.

The reason is that the transpilation step removes all trace of type – TypeScript transpiles `typeof` and `instanceof` operations into JavaScript, and during that step, it erases types. The absence of type at runtime justifies the lack of generic type verification at runtime as well.

The *Generic* solution

To compare two generics, we need to keep in mind that the types are erased (removed) when TypeScript is transpiled to JavaScript. Thus, we need to create a custom generic array with a unique identifier (or many) for comparison. For example, we can add an `id` field and have that value, not erased at runtime, be compared.

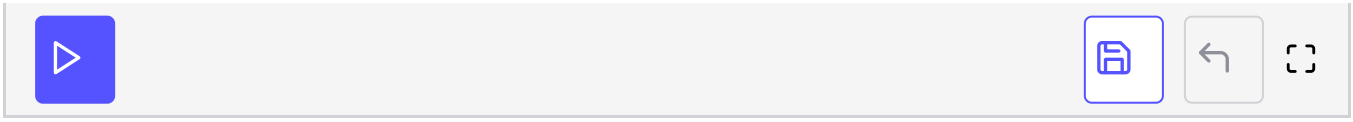
```
class IdentificatedGeneric<S> extends Array<S> {
    public id: string; // Enhancement of Array class
    public constructor(id: string) {
        super();
        this.id = id;
    }
}
function concatenate<S, T1 extends IdentificatedGeneric<S>>(list1: T1, list2: T1): T1 {
    if (list1.id === list2.id) { // Comparison to ensure from the same id, possible because b
        const oneList = [...list1, ...list2] as T1;
        return oneList;
```

```
    }
    throw Error("Must be the same id");
}

const l1 = new IdentificatedGeneric<string>("l1");
const l2 = new IdentificatedGeneric<string>("l2");
const l3 = new IdentificatedGeneric<number>("l1");
const l4 = new IdentificatedGeneric<string>("l1");

l1.push("1", "2");
l2.push("100", "200");
l3.push(5, 6);
l4.push("500", "600");

// const c1 = concatenate(l1, l2); // Error 1
// console.log(c1);
// const c2 = concatenate(l1, l3); // Error 2
// console.log(c2);
const c3 = concatenate(l1, l4);
console.log(c3);
```



The example above contains a custom generic that extends `Array` which is also generic, see **line 1**. In that code, the custom generic array has a field `id` at **line 2** and a function, at **line 9**, checks if the other `Array` has the same `id` before performing concatenation. the same id before performing the concatenation. The idea is to illustrate how to compare two generic types. In this case, by a field value that is available at runtime.

Uncommenting **lines 26** and **27** throws an exception because they have a different `id`. Even if they have the same type, it simulates two different generics.

Uncommenting **lines 28** and **29** causes a transpilation error because the type is different even if they have the same `id`.