Lab 8 - Generics

Preparations

If you couldn't finish the previous exercise, you can copy and paste the pervious solution from the *labsSolutions* folder.

In this lab, we'll implement an audit log function.

Exercise 1 - Create an `auditLog` function

1. Make sure both `Iban` and `Customer` have a `format` method. It should be without parameters and return a string. Copy it from the labSolutions if it is missing.
2. Inside the main.ts file, add an `auditLog` function. It should take an argument called "subject" of type `Iban` and an argument called "action" of type string.
3. Inside the `auditLog` function, log the message in this form "[subject]: action". Log it to console for now (we'll have to implement an actual audit log later on). Make sure you call the `format` method on `subject` when you're logging it.
4. Call the `auditLog` function from inside the `BankAccount` constructor (just after you've created the `Iban`).

   ```
   auditLog(this.iban, 'created');
   ```

5. Now also use the `auditLog` function to log when a customer is assigned to a bank account. Add `auditLog(customer, 'assigned');` to the `createAccount` method of the `Bank` class. In order to make it work, make the `auditLog` function generic. Change the type of `subject` to be of generic type `T`. **Hint:** You might need a type constraint to make this work.

Exercise 2 - My very own `call` - if time permits

This is a fun exercise. Try to make your own `call` function. It takes a function as it's first argument and a list of parameters as the next arguments. It should execute the function for you. This is what it looks like without generic type arguments:

```
function call(fn: any, ...args: any[]): any {
  return fn(...args);
}
```

Now introduce generic type arguments to make the function type safe.

When you're done, you should be able to use it like this:

```
function increment(n: number) {
  return ++n;
```

```
}

console.log(call(increment, 41));
call(console.log, 'test');
```

However, this should result in compile errors:

```
call(increment, '42'); // => ERROR!
const str: string = call(increment, 42); // => ERROR!
```