```
class RaceService {

  @Log()
  getRaces() {
    // call API
  }

  @Log()
  getRace(raceId) {
    // call API
  }
}
```

To define it, we have to write a method returning a function like this:

```
const Log = () => {
  return (target: any, name: string, descriptor: any) => {
    logger.log(`call to ${name}`);
    return descriptor;
  };
};
```

Depending on what you want to apply your decorator to, the function will not have exactly the same arguments. Here we have a method decorator that takes 3 parameters:

- `target`: the method targeted by our decorator
- `name`: the name of the targeted method
- `descriptor`: a descriptor of the targeted method (is the method enumerable, writable, etc.)

Here we simply log the method name, but you could do pretty much whatever you want: interfere with the parameters, the result, calling another function, etc.

So, in our simple example, every time the `getRace()` or `getRaces()` methods are called, we'll see a trace in the browser logs:

```
    raceService.getRaces();
    // logs: call to getRaces
    raceService.getRace(1);
    // logs: call to getRace
```

As a user, let's look at what a decorator in Angular looks like: