

```

let nos: number[] = [7, 6, 4, 3, 1];
function map (elems, transformFn) {
}
x => x * 2
[14, 12, 8, 6, 2]
06000
products: Product[] = [
  { name: 'Samsung Washing Machine', price: 45000.00, category: 'electronics' },
  { name: 'iPhone', price: 95000.00, category: 'mobile' },
  { name: 'Samsung OLED', price: 245000.00, category: 'tv' },
  { name: 'Samsung Fold', price: 124000.00, category: 'mobile' },
  { name: 'Wacom', price: 6000.00, category: 'electronics' }
]
c => e.name
'samsung washing machine'
'iPhone'
'Samsung OLED'
'Samsung Fold'
'Wacom'
sing[1]

```

Diagram illustrating the execution flow of the `map` function:

- The code starts with `let nos: number[] = [7, 6, 4, 3, 1];` and `function map (elems, transformFn) {`.
- An arrow points from the array `nos` to the parameter `elems` in the `map` function.
- The transformation function `x => x * 2` is passed to `map` as `transformFn`.
- The resulting array `[14, 12, 8, 6, 2]` is shown.
- A green checkmark is placed next to the first line of code.
- Below the code, there is a list of product names: 'Samsung Washing Machine', 'iPhone', 'Samsung OLED', 'Samsung Fold', and 'Wacom'.
- A green bracket labeled `sing[1]` groups the first four items: 'Samsung Washing Machine', 'iPhone', 'Samsung OLED', and 'Samsung Fold'.
- Handwritten annotations include '06000' and 'c => e.name'.

Below this section, another diagram shows the implementation of the `map` function:

```

let nos: number[] = [7, 6, 4, 3, 1]; (1)
let output: number[] = map(nos, no => no * 2);

```

```

export function map<T, R>(data: T[], transformFn: (elem: T) => R): R[] {
  let result: R[] = [];
  data.forEach(elem => {
    result.push(transformFn(elem));
  });
  return result;
}
no | no * 2

```

Annotations include circled '1' over the first line of code and circled 'no' and 'no * 2' under the implementation code.

```

let products: Product[] = [
  { name: 'Samsung Washing Machine', price: 45000.00, category: 'electronics' },
  { name: 'iPhone', price: 95000.00, category: 'mobile' },
  { name: 'Samsung OLED', price: 245000.00, category: 'tv' },
  { name: 'Samsung Fold', price: 124000.00, category: 'mobile' },
  { name: 'Wacom', price: 6000.00, category: 'electronics' }
]

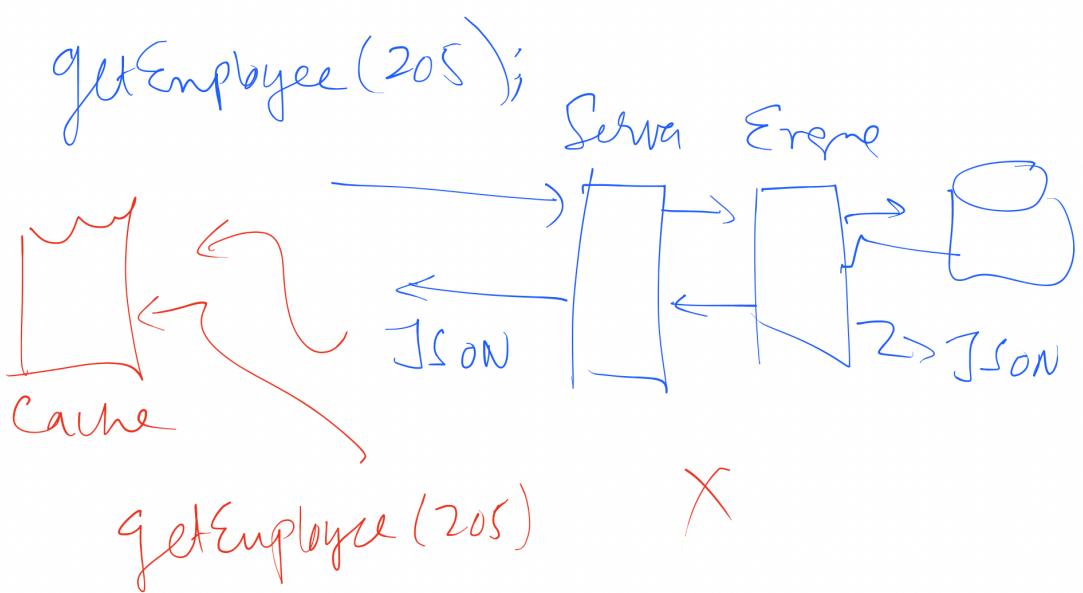
```

Diagram illustrating the execution flow of the `map` function:

- The code starts with `let products: Product[] = [...];` and `function map<T, R>(data: T[], transformFn: (elem: T) => R): R[] {`.
- An arrow points from the array `products` to the parameter `data` in the `map` function.
- The transformation function `let result: R[] = []` is passed to `map` as `transformFn`.
- The resulting array `R` is shown.
- A green circle labeled '1' is placed over the first line of code.
- Below the code, there is a list of product names: 'Samsung Washing Machine', 'iPhone', 'Samsung OLED', 'Samsung Fold', and 'Wacom'.
- A green circle labeled '1' is placed over the first item in the list.
- On the right, a function `toCard(p: Product)` is defined, which returns an HTML string for a card: `

<h1>\${p.name}</h1><h4>\${p.price}</h4>

`.
- A green circle labeled '2' is placed over the second item in the list.
- Below the function definition, the code `let cards: string[] = map(products, toCard);` is shown, with a green circle labeled '2' placed over it.



```
export default function Memo(target: any, methodName: string, descriptor: PropertyDescriptor) {
  let fn:Function = descriptor.value; // actual method code --> fibonacci

  let cache:Map<String, any> = new Map();

  descriptor.value = function(...args:number[]) {
    let key = args.join("-");
    if(cache.has(key)) {
      const result = fn.apply(this, args);
      cache.set(key, result);
    }
    return cache.get(key);
  }
}

@Memo
fibonacci(no: number): number {
  return (no === 0 || no === 1) ? no : this.fibonacci(no - 1) + this.fibonacci(no - 2);
}
```

