Shallow copy

vs.

Deep copy

in JavaScript

SWIPE TO LEARN

RAJ BHENSDADIYA

- Are you familiar with the concepts of shallow copy and deep copy in JavaScript?

- Let's dive into this essential topic for JavaScript developers!

RAJ BHENSDADIYA

Shallow Copy

- Shallow copying creates a new object, but instead of duplicating all nested objects and their properties, **it copies references** to them.

- In simpler terms, a shallow copy doesn't go deep into the nested structures; it just **replicates the surface**.

RAJ BHENSDADIYA

Example - Shallow Copy

- Suppose you have an array of students, and you want to create a copy of that array.

JS shallowCopy.js

```
const originalStudents = [{ name: 'Alice' }, { name: 'Bob' }];
const shallowCopy = [...originalStudents];

shallowCopy[0].name = 'Alex';

console.log(originalStudents[0].name); // Output: Alex
```

Example - Shallow Copy

- In this example, changing the name in the shallowCopy array also affects the original array.

- That's because both arrays reference the same student objects.

Deep Copy

- Deep copying **creates a completely independent copy** of an object and all its nested objects.
- It means every level of **nesting is duplicated**, ensuring that changes in the copied object **won't affect the original**.

Example - Deep Copy

- Imagine you have a configuration object for a game, and you want to create a deep copy to customize it without altering the original.

Example - Deep Copy

JS deepCopy.js

```
const originalConfig = {
  gameName: 'AwesomeGame',
  settings: {
    difficulty: 'Medium',
    sound: 'On',
  },
};

// Using a library like Lodash to create a deep copy
const deepCopy = _.cloneDeep(originalConfig);

deepCopy.settings.difficulty = 'Hard';

console.log(originalConfig.settings.difficulty); // Output: Mediumd
```

RAJ BHENSDADIYA

Example - Deep Copy

- In this case, the deep copy maintains a separate set of settings, so changes don't affect the original configuration.

When to Use Which?

- Shallow Copy: Use it when you want to **replicate an object quickly** and **don't need a completely independent copy**.
- It's **memory-efficient** and suitable for many scenarios.
- Deep Copy: Opt for deep copying when you **need a truly independent duplicate** of an object, especially if the object has nested structures.
- It ensures that changes to the copy **won't impact the original data**.

RAJ BHENSDADIYA

I'm Raj.

Software Engineer;
passionate for building and shipping scalable software.

I am open for remote based contract opportunities/ Development projects.

Let's connect and expand our professional network together.

I'm excited to collaborate, exchange ideas, and contribute to impactful projects.