JSDoc is a documentation

Next let's dive into something called JSDoc before we continue with the OOP practical example:

JSDoc is a documentation standard for writing inline comments in JavaScript code. It allows developers to describe the structure, purpose, and usage of their code in a clear and standardized way. When you use JSDoc, you write annotations inside special comment blocks, and these comments can later be used to generate HTML documentation or to simply help other developers understand the code.

Key Benefits of Using JSDoc:

1. **Clarity:** It explains the purpose of functions, parameters, and return values directly in the code, making it easier to understand what each piece of code is doing.

2. **Collaboration:** When working in teams, JSDoc makes it easier for other developers to jump into the project and understand how functions and classes are structured without needing to read the implementation in depth.

3. **Code Intelligence:** Modern code editors (like VSCode) can use JSDoc annotations to provide code hints, autocompletion, and type checking.

Why We Use It in the Code Example:

In the example I provided, JSDoc is used to:

- Explain Class and Method Responsibilities: Each class, method, and parameter has an associated description explaining its purpose.

- Parameter Annotations (`@param`): We define the types and meanings of the parameters passed to methods or constructors.

- Improved Readability: Future developers (or students) can understand what a method or constructor does without looking through all of the code logic.

```
/**
 * Constructor for creating a new User object
 * @param {string} username - The username of the user
 * @param {string} fullName - The full name of the user
 */
constructor(username, fullName) {
```

```
    this.username = username;
    this.fullName = fullName;
}
```

- `@param {string} username`: Describes the `username` parameter as a string.
- `@param {string} fullName`: Describes the `fullName` parameter as a string.
- This tells developers what types to expect when they use this class or function.

When to Use JSDoc:

- Whenever you're writing code that will be reused or shared with others.
- For complex functions or methods where the parameters or return values need clear explanation.
- When your codebase grows large and needs a structured way of describing functionality.

By using JSDoc consistently, you help maintain clean and understandable code, which benefits both current and future developers on the project.

Now we can jump into a practical example where we will be creating an example of an instagram application only using OOP and some of the fundamentals that we have learned up until now.

```
// Class to represent a User
class User {
    /**
     * Constructor for creating a new User object
     * @param {string} username - The username of the user
     * @param {string} fullName - The full name of the user
     */
    constructor(username, fullName) {
        this.username = username; // stores the user's username
        this.fullName = fullName; // stores the user's full name
        this.posts = []; // initialize an empty array to store posts
created by the user
    }
```

```
    /**
     * Create a new post and add it to the user's post list
     * @param {string} content - The content of the post
     */
    createPost(content) {
        const newPost = new Post(this.username, content); // create a new
Post object
        this.posts.push(newPost); // add the post to the user's posts
array
    }

    /**
     * Display all posts created by the user
     */
    showPosts() {
        console.log(`Posts by ${this.username}:`);
        this.posts.forEach((post) => {
            post.displayPost(); // for each post, display its content
        });
    }
}

// Class to represent a Post
class Post {
    /**
     * Constructor for creating a new Post object
     * @param {string} username - The username of the post's creator
     * @param {string} content - The content of the post
     */
    constructor(username, content) {
        this.username = username; // store the username of the person who
created the post
        this.content = content; // store the content of the post
        this.createdAt = new Date(); // store the current date and time
the post was created
    }

    /**
     * Display the post content in a formatted manner
     */
```

```
    displayPost() {
        console.log(`@${this.username} posted on ${this.createdAt}:`);
        console.log(this.content);
        console.log('------------------------'); // separator for clarity
    }
}
// Practical example
// Creating users
const user1 = new User("johnDoe", "John Doe");
const user2 = new User("janeSmith", "Jane Smith");

// Users creating posts
user1.createPost("Hello, world! This is my first post on Instagram.");
user1.createPost("Loving this new platform! #excited");

user2.createPost("Hey everyone! Just trying out this Instagram thing.");
user2.createPost("Had a great day at the beach today!");

// Displaying the posts
user1.showPosts();
user2.showPosts();
```