Certainly! Let me provide a summary based on the provided transcript:

### Summary:

1. \*\*Chaining Methods in Class:\*\*

- Similar to chaining array methods, you can make class methods chainable.

- To achieve this, return the object (`this`) at the end of a method.

- This allows for method chaining, where one method can be called immediately after another on the same object.

2. \*\*Example Chaining:\*\*

- Demonstrated an example where multiple methods are chained on an `account` object.

- Methods included `deposit`, `withdraw`, and `requestLoan`.

- The methods modified the `movements` array and returned the account object to enable chaining.

3. \*\*Implementation:\*\*

- Emphasized that chaining makes the most sense for methods that modify or set some property.

- Applied the `return this;` statement at the end of each method to make them chainable.

4. \*\*Final Outcome:\*\*

- Executed the chained methods on an account object without errors.

- Verified the success by checking the movements array and receiving a loan approval message.

5. \*\*Tool in Toolbox:\*\*

- Encouraged experimentation with chaining methods in your own classes.

- Highlighted the flexibility and power this approach brings to designing class APIs.

6. \*\*Conclusion of OOP Section:\*\*

- Wrapped up the Object-Oriented Programming (OOP) section in JavaScript.

- Acknowledged the comprehensive coverage of class syntax and its practical application.

7. \*\*Upcoming Content:\*\*

- Teased a final overview of the entire class syntax.

- Mentioned a concluding coding challenge to reinforce learning.

### Final Thoughts:

- The section covered a comprehensive exploration of OOP in JavaScript, providing a solid foundation for building and working with classes.

- Encouraged practical application through experimentation and hands-on coding challenges.

With this summary, you have a concise overview of the key concepts and practical examples presented in the transcript. If you have any specific questions or if there's anything else you'd like to explore, feel free to ask!

Code:

Certainly! It looks like the transcript provided some code examples related to chaining methods in a class. Here's the code based on the context of the transcript:

```javascript

class Account {

#movements = [];

// Public Methods

deposit(val) {

this.#movements.push(val);

return this; // Make the method chainable

}

withdraw(val) {

this.deposit(-val); // Using deposit method for simplicity

return this; // Make the method chainable

}

requestLoan(val) {

if (this.approveLoan(val)) {

this.deposit(val);

console.log('Loan approved');

} else {

console.log('Loan denied');

}

return this; // Make the method chainable

}

// Private Methods

#approveLoanThreshold() {

return this.#movements.reduce((acc, mov) => acc + mov, 0) \* 0.1;

}

// Public API Method

getMovements() {

return this.#movements;

}

// Additional Method for Chaining Example

exampleMethod() {

// Some logic here

return this; // Make the method chainable

}

}

// Example Usage

const account1 = new Account();

account1.deposit(300).deposit(500).withdraw(35).requestLoan(25000).withdraw(4000);

// Checking Movements

console.log(account1.getMovements());

```

In this code:

- The `deposit`, `withdraw`, `requestLoan`, and `exampleMethod` methods are designed to be chainable by returning `this`.

- The example usage demonstrates how these methods can be chained on an `account1` instance.

Feel free to ask if you have any questions or if there's anything specific you'd like to explore further!