Certainly! Here are the key points and notes from the continuation of the lecture:

**1. \*\*Encapsulation and Data Privacy Recap:\*\***

- Encapsulation involves keeping properties and methods private within a class, exposing only a limited public interface or API.

- Two main reasons for encapsulation:

- Preventing unintentional manipulation of data from outside the class.

- Facilitating confident changes to internal methods without breaking external code that relies on the class.

**2. \*\*JavaScript Limitations:\*\***

- JavaScript classes do not currently support real data privacy and encapsulation.

- A proposal for adding private class fields and methods exists but is not yet finalized.

**3. \*\*Convention for Protection:\*\***

- In the absence of true encapsulation, developers often use conventions to indicate that certain properties or methods are intended for internal use only.

- A common convention is to prefix names with an underscore (`\_`), signifying that the property or method is protected but not truly private.

**4. \*\*Protected Properties Example:\*\***

- The `movements` array is considered mission-critical data and is protected using the underscore convention.

- Developers should refrain from directly manipulating protected properties from outside the class.

**5. \*\*Public Method for Access:\*\***

- If there's a need to provide external access to a protected property, a public method can be implemented (e.g., `getMovements` method).

- The method serves as part of the public API, ensuring controlled access to the protected property.

**6. \*\*Protecting Other Properties:\*\***

- In the example, the `pin` property is also protected, as it doesn't make sense for it to be accessible from the outside.

- The decision to protect properties depends on the design and requirements of the application.

**7. \*\*Protecting Internal Methods:\*\***

- The `\_approveLoan` method, which is intended for internal use only, is also protected using the underscore convention.

- This ensures that external code does not rely on or attempt to call this internal method.

**8. \*\*Public API Summary:\*\***

- The public API of the class now consists of methods like `deposit`, `withdraw`, `requestLoan`, and `getMovements`.

- Developers are encouraged to use this API for interactions, and direct access to protected properties and methods is discouraged.

**Code:**

class Account {

constructor(owner, currency, pin) {

this.owner = owner;

this.currency = currency;

this.\_pin = pin; // Protected property using underscore convention

this.\_movements = []; // Protected property for movements array

this.\_locale = navigator.language;

console.log(`Thanks for opening an account, ${this.owner}!`);

}

// Public method for deposit

deposit(value) {

this.\_movements.push(value);

}

// Public method for withdrawal

withdraw(value) {

this.deposit(-value); // Reusing deposit method to handle withdrawal

}

// Public method for requesting a loan

requestLoan(value) {

if (this.\_approveLoan(value)) {

this.deposit(value);

console.log(`Loan approved! ${value} deposited into your account.`);

} else {

console.log(`Loan request denied.`);

}

}

// Public method to get movements array

getMovements() {

return this.\_movements;

}

// Protected method for loan approval

\_approveLoan(value) {

// For simplicity, always approve the loan

return true;

}

}

// Example usage:

const acc1 = new Account('Jonas', 'Euro', 1234);

// Access protected properties (not recommended, but still possible)

console.log(acc1.\_pin); // Accessing protected property

console.log(acc1.\_movements); // Accessing protected property

// Use public methods to interact with the account

acc1.deposit(250);

acc1.withdraw(140);

acc1.requestLoan(1000);

// Use public method to access protected property

console.log(acc1.getMovements()); // Using public method to access protected property