Blockchain Study Notes Day 23:

Module 4 - Solidity Advanced
Chapter 1 - Inheritance in Solidity

Introduction to Inheritance

Inheritance in Solidity allows one contract to acquire the properties and functions of another contract. It enables code reusability, modularity, and efficient smart contract design.

---

1. What Is Inheritance?

- **Definition**:
  Inheritance allows a contract to extend the functionality of an existing contract by inheriting its properties and methods.
- **Purpose**:
  - Reuse existing code.
  - Extend or modify functionality.
  - Implement hierarchical contract structures.

---

2. Basic Syntax of Inheritance

Single Inheritance:

```
contract Parent {
    // Parent contract code
}

contract Child is Parent {
    // Child contract inherits from Parent
}
```

Multiple Inheritance:

```
contract A {
    // Contract A code
}

contract B {
    // Contract B code
}

contract C is A, B {
    // Contract C inherits from both A and B
}
```

---

3. Example Program Demonstrating Inheritance (Using Munawar)

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

// Parent contract
contract Parent {
    string public familyName;

    constructor(string memory _familyName) {
        familyName = _familyName;
    }

    function getFamilyName() public view returns (string memory) {
        return familyName;
    }
}

// Child contract
contract Child is Parent {
    string public firstName;

    constructor(string memory _familyName, string memory _firstName)
Parent(_familyName) {
        firstName = _firstName;
    }

    function getFullName() public view returns (string memory) {
        return string(abi.encodePacked(firstName, " ", familyName));
    }
}
```

Explanation:

1. **`Parent` Contract**:
   - Stores the family name.
2. **`Child` Contract**:
   - Inherits from `Parent`.
   - Adds `firstName` and a function `getFullName` to combine names.
   - Calls the `Parent` constructor.

---

4. Overriding Functions

Derived contracts can override base contract functions using the `override` and `virtual` keywords.

Example:

```
contract Base {
    function greet() public virtual pure returns (string memory) {
        return "Hello from Base";
    }
}

contract Derived is Base {
    function greet() public pure override returns (string memory) {
        return "Hello from Derived";
    }
}
```

---

5. Multiple Inheritance and `super` Keyword

When inheriting from multiple contracts, the `super` keyword can be used to call parent contract functions.

Example:

```
contract A {
    function foo() public virtual pure returns (string memory) {
        return "A";
    }
}

contract B {
    function foo() public virtual pure returns (string memory) {
        return "B";
    }
}

contract C is A, B {
    function foo() public pure override(A, B) returns (string memory) {
        return super.foo();  // Calls function from the most recent
inheritance
    }
}
```

---

6. Best Practices for Using Inheritance

- **Avoid Deep Inheritance Trees**:
    - Keep inheritance simple to maintain clarity and reduce complexity.
- **Use `override` and `virtual` Explicitly**:
    - Always use `virtual` in base contract functions and `override` in derived contracts.
- **Leverage Abstract Contracts and Interfaces**:

- Use abstract contracts for shared logic and interfaces for defining standard functions.

Home Task

1. **Extend the Example Program**:
   - Add a `GrandChild` contract that inherits from `Child` and overrides a function.
2. **Create a New Contract**:
   - Implement a multiple inheritance contract for managing different user roles (e.g., Admin, User).
3. **Research**:
   - Explore how real-world contracts (like ERC-20 tokens) use inheritance.

Conclusion

Inheritance in Solidity is a powerful tool for creating modular and reusable contracts. By mastering inheritance, developers can design scalable and maintainable blockchain applications.

.

Day 23 Notes

Prepared by Munawar Johar