**Blockchain Study Notes Day 9:**

**Module 2 - Solidity Basics**  
**Chapter 5 - Access Modifiers in Solidity**

**Introduction to Access Modifiers**

Access modifiers in Solidity define the visibility and accessibility of functions and state variables. They control who can call a function or access a variable, enhancing contract security and organization.

**1. Types of Access Modifiers**

**1.1. Public**

* **Description**:  
  Functions and state variables marked as public can be accessed from anywhere, both within the contract and externally.
* **Default for Functions**: If not explicitly specified, functions default to public.
* **Example**:

uint public myPublicVariable = 100;

function getPublicValue() public view returns (uint) {

return myPublicVariable;

}

**1.2. Private**

* **Description**:  
  Accessible only within the contract where they are declared.  
  Prevents external access, even from derived contracts.
* **Example**:

uint private myPrivateVariable = 200;

function getPrivateValue() public view returns (uint) {

return myPrivateVariable;

}

**1.3. Internal**

* **Description**:  
  Similar to private but allows access from derived contracts.  
  Useful for sharing functionality across inherited contracts.
* **Example**:

uint internal myInternalVariable = 300;

function getInternalValue() internal view returns (uint) {

return myInternalVariable;

}

**1.4. External**

* **Description**:  
  Can only be called from outside the contract.  
  Cannot be called internally using this.
* **Example**:

function setExternalValue(uint \_value) external {

myPublicVariable = \_value;

}

**2. Example Program Using Access Modifiers (Using Munawar)**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract AccessModifiers {

// Public variable

string public name = "Munawar";

// Private variable

uint private secretCode = 1234;

// Internal variable

uint internal internalValue = 500;

// External function to set a new name

function setName(string calldata \_newName) external {

name = \_newName;

}

// Public function to get secret code indirectly

function getSecretCode() public view returns (uint) {

return secretCode;

}

// Internal function to get internal value

function getInternalValue() internal view returns (uint) {

return internalValue;

}

}

// Derived contract to demonstrate `internal` access

contract DerivedAccess is AccessModifiers {

function accessInternalValue() public view returns (uint) {

return getInternalValue();

}

}

**Explanation**:

1. **Public Modifier**:
   * name can be accessed from anywhere.
2. **Private Modifier**:
   * secretCode is only accessible within AccessModifiers.
3. **Internal Modifier**:
   * internalValue can be accessed within AccessModifiers and its derived contracts like DerivedAccess.
4. **External Modifier**:
   * setName can only be called from outside the contract.

**3. Best Practices for Access Modifiers**

* Use **private** for sensitive data to prevent external access.
* Use **internal** for shared functionality across derived contracts.
* Use **public** for functions or variables meant for external and internal use.
* Use **external** for functions intended solely for external calls.

**Home Task**

1. **Modify the Example Program**:
   * Add a private function getSecretCodeWithMultiplier that multiplies secretCode by a given number.
2. **Create a New Contract**:
   * Write a contract demonstrating the use of external and internal functions in inheritance.
3. **Experiment with Visibility**:
   * Try accessing private and internal variables/functions in a derived contract to see how visibility works.

**Conclusion**

Access modifiers in Solidity are crucial for controlling function and variable visibility. By understanding and correctly applying these modifiers, developers can enhance the security and structure of their smart contracts.

Day 9 Notes

***Prepared by Munawar Johar***