Centurion University
Shaping Lives... Empowering Communities...

School: ..... Campus: .....

Academic Year: ..... Subject Name: ..... Subject Code: .....

Semester: ..... Program: ..... Branch: ..... Specialization: .....

Date: .....

Applied and Action Learning

(Learning by Doing and Discovery)

Name of the Experiement :

* Coding Phase: Pseudo Code / Flow Chart / Algorithm

Coding Phase: Pseudo Code / Flow Chart / Algorithm

Advanced Inheritance Patterns in Solidity

Inheritance in Solidity is a mechanism that enables a smart contract (the derived or child contract) to inherit properties—such as functions and state variables—from another contract (the base or parent contract). This paradigm promotes code reuse and structured organization. For instance, if Contract B inherits from Contract A, it gains access to A's methods and data. The derived contract can also redefine, or "override," inherited functions to provide its own specific logic. Inheritance is declared using the is keyword (e.g., contract B is A).

Mechanism of Inheritance:

Inheritance functions by allowing a developer-defined contract to incorporate and build upon the variables and functions of existing parent contracts. When a contract like contract Token is ERC20 {} is created, Token inherits all the capabilities of the ERC20 standard. This eliminates the need to rewrite foundational code.

To tailor behavior, child contracts can override parent functions. For example, a custom token might override the transfer function to include additional validation before executing the standard transfer logic by calling super.transfer(). Proper initialization is critical; a child contract's constructor must correctly initialize its parent contracts to ensure inherited state variables are set up properly, preventing potential vulnerabilities.

Categories of Inheritance in Solidity

1. Single Inheritance

This occurs when a contract inherits from a single parent.

- *Concept: The child contract has access to all public and internal members of its sole parent. This establishes a straightforward hierarchical relationship, where a general-purpose base contract can be specialized by a child contract.*
- *Example: A SavingsAccount contract could inherit from a generic Account contract to reuse core deposit and withdrawal logic, adding specific features for interest calculation.*

2. Multiple Inheritance

A contract can inherit from two or more parent contracts simultaneously.

- Concept: This allows a child contract to combine functionalities from various

*As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.

sources. For instance, a ManagedToken contract could inherit from an ERC20 contract for token standards and an Ownable contract for access control.

3. Multi-Level Inheritance

This describes an inheritance chain spanning more than two levels (e.g., Child -> Parent -> Grandparent).

- Concept: It enables a hierarchy where each level adds a layer of specialization. A foundational BaseAccount might be extended by a TokenAccount, which is further specialized by a StakingTokenAccount.

4. Function Overriding

A child contract can provide a new implementation for a function inherited from a parent.

- Concept: The parent function must be marked virtual, and the child's overriding function must be marked override. This allows the child to modify or extend the parent's behavior. The super keyword is used to call the overridden function from the parent contract, enabling the child to add logic before or after the original execution.
- Example: An overriding withdraw function could add a fee deduction or a daily limit check and then call super.withdraw() to perform the actual balance update.

5. Abstract Contracts

These are incomplete contracts that cannot be deployed on their own and are designed solely to be base contracts. They are declared with the abstract keyword.

- Concept: Abstract contracts can define function interfaces without an implementation (leaving them for child contracts to define) and can also contain implemented functions. They act as templates, enforcing a specific structure or set of capabilities that all derived contracts must implement.
- Example: An abstract Voting contract could define the functions propose(), vote(), and tally(), requiring child contracts to provide the specific voting mechanics.

Software used :

Remix IDE

Testing Phase: Compilation of Code (error detection)

No error

**As applicable according to the experiment.*
Two sheets per experiment (10-20) to be used.

* Implementation Phase: Final Output (no error)

1.Single Inheritance

Single Inheritance:

solidity Copy code

```
// Parent contract
contract Parent {
    uint public value;

    function setValue(uint _value) public {
        value = _value;
    }
}

// Child contract inheriting from Parent
contract Child is Parent {
    function getValue() public view returns (uint) {
        return value;
    }
}
```

2.Multiple Inheritance

Multiple Inheritance:

solidity Copy code

```
// Parent contracts
contract Parent1 {
    uint public value1;
}

contract Parent2 {
    uint public value2;
}

// Child contract inheriting from multiple parents
contract Child is Parent1, Parent2 {
    function getChildValues() public view returns (uint, uint) {
        return (value1, value2);
    }
}
```


*As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.

Implementation Phase: Final Output (no error)

3.Multi level inheritance

3. Hierarchical Inheritance:

solidity Copy code

```solidity
// Grandparent contract
contract Grandparent {
    uint public grandparentValue;
}

// Parent contract inheriting from Grandparent
contract Parent is Grandparent {
    uint public parentValue;
}

// Child contract inheriting from Parent
contract Child is Parent {
    uint public childValue;

    function getChildValues() public view returns (uint, uint, uint) {
        return (grandparentValue, parentValue, childValue);
    }
}
```

4.Function overidding

solidity Copy code

```solidity
// Parent contract
contract Parent {
    uint public value;

    // Function that can be overridden
    function setValue(uint _value) public virtual {
        value = _value;
    }
}

// Child contract overriding function
contract Child is Parent {
    // Override the setValue function from Parent
    function setValue(uint _value) public override {
        // Add custom logic or constraints
        require(_value > 0, "Value must be greater than zero");

        // Call the parent function to set the value
        super.setValue(_value);
    }
}
```


*As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.

* Implementation Phase: Final Output (no error)

5. Abstract contracts

```solidity
// Parent contract
contract Parent {
    uint public value;

    // Function that can be overridden
    function setValue(uint _value) public virtual {
        value = _value;
    }
}

// Child contract overriding function
contract Child is Parent {
    // Override the setValue function from Parent
    function setValue(uint _value) public override {
        // Add custom logic or constraints
        require(_value > 0, "Value must be greater than zero");

        // Call the parent function to set the value
        super.setValue(_value);
    }
}
```

* Observations

1. Inheritance enables code reuse and logical organization by allowing child contracts to inherit from parent contracts.
2. It supports various patterns like single, multiple, and multi-level inheritance for different design needs.
3. Function overriding with virtual and override keywords allows child contracts to customize inherited logic.
4. Abstract contracts act as un-deployable templates, enforcing structure for derived contracts.
5. The primary benefits are modularity, reduced code duplication, and polymorphism, but overuse can complicate design.

ASSESMENT

Rubrics	Full Mark	Marks Obtained	Remarks
Concept	10		
Planning and Execution/ Practical Simulation/ Programming	10		
Result and Interpretation	10		
Record of Applied and Action Learning	10		
Viva	10		
Total	**50**		

Signature of the Student:

Name :

Regn. No. :

Signature of the Faculty:

****As applicable according to the experiment. Two sheets per experiment (10-20) to be used.***