Pakistan Blockchain Institute

PAKISTAN BLOCKCHAIN INSTITUTE

MODULE-2
BLOCKCHAIN AND SMART CONTRACT BASICS

Class-11

Raja Rizwan Saleem
Lead Blockchain Trainer

Edversity.

Inheritance

Inheritance

1. Inheritance is the process of defining multiple contracts that are related to each other through parent-child relationships.
2. The contract that is inherited is called the **parent contract** and the contract that inherits is called the **child contract**.
3. Similarly, the contract has a parent known as the **derived class** and the parent contract is known as a **base contract**.
4. Inheritance is mostly about code-reusability.

5. There is a is-a relationship between base and derived contracts and all public and internal scoped functions and state variables are available to derived contracts.

6. Solidity compiler copies the base contract bytecode into derived contract bytecode.

7. The is keyword is used to inherit the base contract in the derived contract.

Inheritance

“Solidity copies the base contracts into the derived contract and a single contract is created with inheritance. A single address is generated that is shared between contracts in a parent-child relationship.”

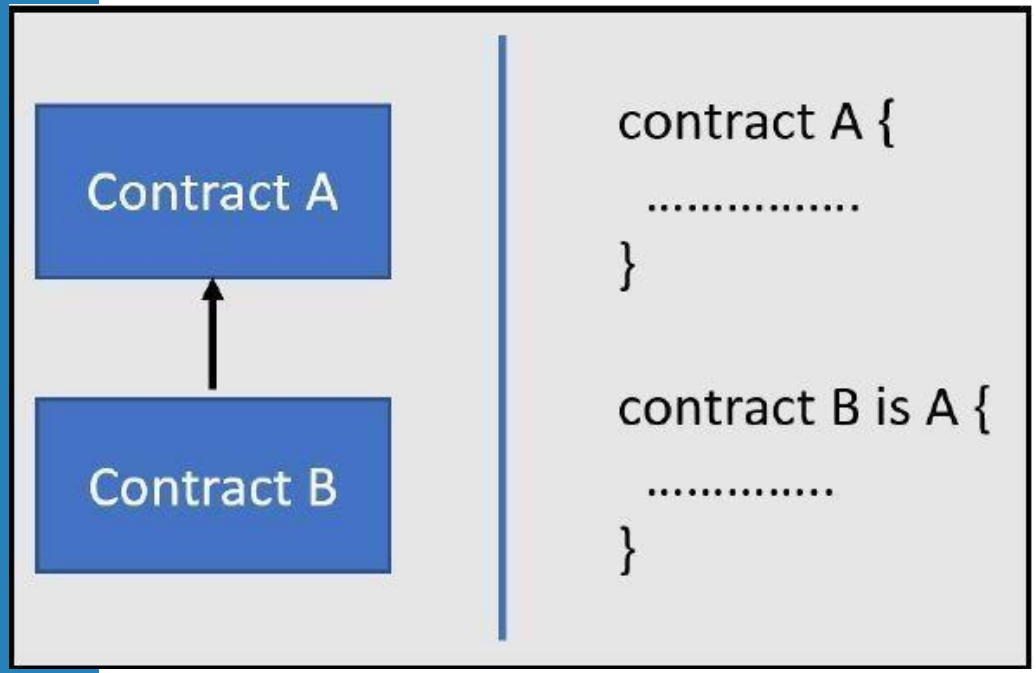
https://www.geeksforgeeks.org/solidity-inheritance/?ref=leftbar-rightbar

Inheritance

Single Inheritance: Inherits from one base contract.
Multiple Inheritance: Inherits from multiple base contracts.
Hierarchical Inheritance: Multiple derived contracts inherit from a single base contract.
Multilevel Inheritance: A contract is derived from a derived contract, forming a chain.

Single inheritance

Single inheritance helps in inheriting the variables, functions, modifiers, and events of base contracts into the derived class.



Inheritance -- Example 1a

```
contract Human { uint
    internal age;
    function setAge(uint a) public {
        age = a;

    }}
contract Student is Human {
    function getAge() public returns (uint) {
        return age;

    }}
```

Inheritance -- Example 1b

1. setAge function is inherited from parent Human contract

```
contract Client {
    function createObjects() public {
        Student st = new
        Student(); st.setAge(20);

        st.getAge();
    }
}
```

Inheritance -- Example 1b

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

contract Parent {
    uint public value;

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

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

Inheritance -- State Variable shadowing is not allowed

1. State variable shadowing is not allowed.
2. A derived contract can only declare a state variable x, if there is no visible state variable with the same name in any of its bases.
3. If parent contract has state variable with **public** or **internal** visibility, then child contact cannot declare variable with same name
4. If parent contract has state variable with private visibility, then child contact can declare variable with same name

Inheritance -- Example 2

```
contract Human { uint
    internal age;
    function setAge(uint a) public {
        age = a;
}}
contract Student is Human {
    uint internal age;  // Compile time Error, Not
    allowed   function getAge() public returns (uint)
    {
        return age;
}}
```

Inheritance -- Example 3

```
contract Human {
    uint private age;// Parent class have private state
    function setAge(uint a) public {

        age = a;
}}
contract Student is Human {
    uint internal age;//Works fine- public, internal, private
    function getAge() public returns (uint) {

        return age;
}}
```

Inheritance -- No constructor in child and parent

1. If there is no constructor in parent and child contract then child contract’s object can be created with default zero argument constructor.

Inheritance -- Example 4a

```
contract Human {
    uint public age;

}
contract Student is Human {
    function getAge() public returns (uint) {
        return age;

}}
```

Inheritance -- Example 4b

1. Create object with default constructor works fine

```
Student st  = new
st.getAge();Student();
```

Inheritance -- Default constructor in parent

1. If there is constructor in parent contract with default zero argument then child contract’s object can be created with default zero argument constructor

Inheritance -- Example 5a

```
contract Human {
    uint public age;

    constructor () public {  }
}

contract Student is Human {
    function getAge() public returns (uint) {

        return age;

}}
```

Inheritance -- Example 5b

1. Create object with default constructor works fine

```
Student st  = new Student();
st.getAge();
```

Inheritance -- Constructor with parameter in parent

1. If there is constructor in parent contract with one or more parameter then child must call parent constructor and provide required arguments
2. Child contract will not compile if it will not provide call to parent constructor

Inheritance -- Example 6a

```
contract Human {
    uint public age;

    constructor (uint _a) public {
        age = _a;
    }}
contract Student is Human {
    constructor(uint _b) Human(_b) public {}

    function getAge() public returns (uint) {

        return age;

    }}
```

Inheritance -- Example 6b

1. Create object with parameter will call constructor of parent contract and set value of age in parent contract

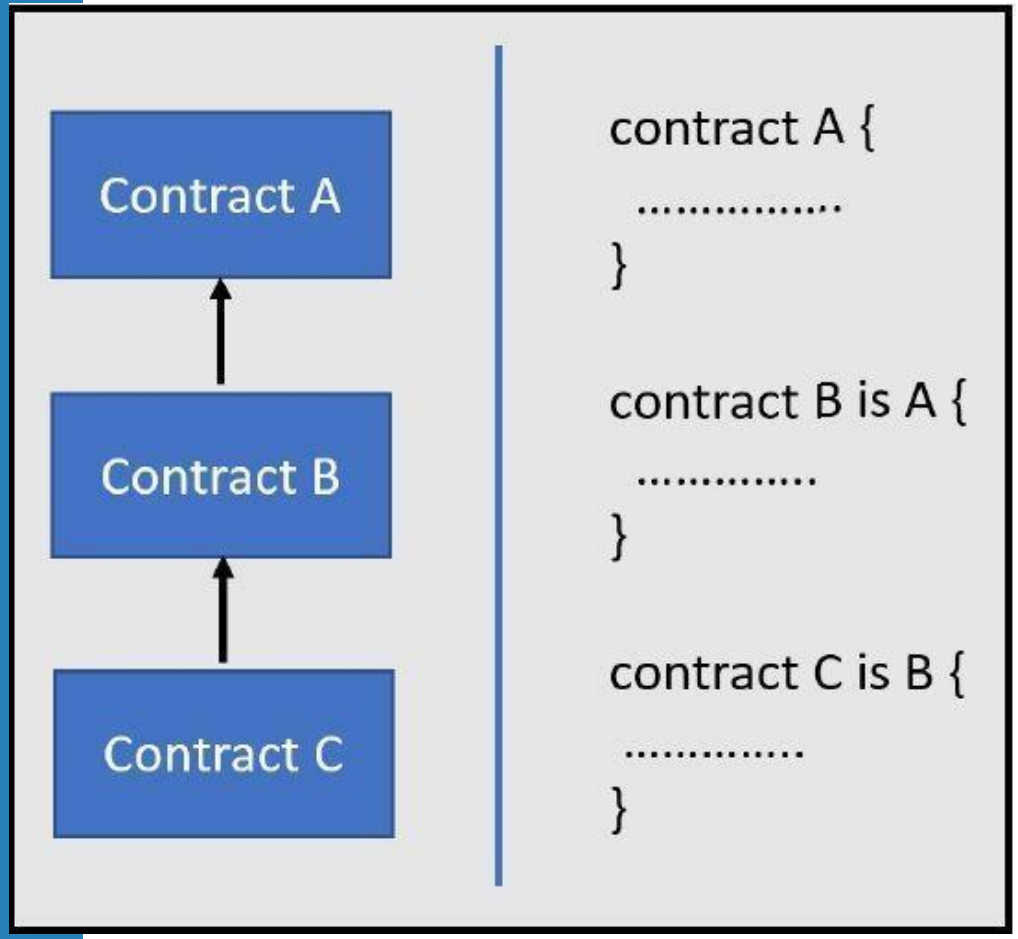
```
Student st = new
Student(5); st.getAge();//
returns 5
```

Inheritance -- Example 6b

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract A {
    uint public a;
    function setA(uint _a) public {
        a = _a;
    }}
contract B {
    uint public b;

    function setB(uint _b) public {
        b = _b;
    }}
contract C is A, B {
    function getValues() public view returns (uint, uint) {
        return (a, b);
    }}
```

Multi-level Inheritance

Multi-level inheritance is very similar to single inheritance; however, instead of just a single parent-child relationship, there are multiple levels of parent-child relationship.



Multi-level Inheritance -- Example 7a

```
contract Human { uint
    internal age;
    string internal name;
    function setAge(uint _a) public {
        age = _a;

    }
    function getName() public returns (string memory){
        return name;

    }
}
```

Multi-level Inheritance -- Example 7b

```
contract Student is Human {
    function getAge() public returns (uint) {
        return age;
    }
    function setName(string memory _name) public {
        name = _name;
    }
}
```

Multi-level Inheritance -- Example 7c

```
contract BlockchainStudent is Student {
    function getCourseName() public returns (string memory) {
        return "Blockchain";
    }
}
```

Multi-level Inheritance -- Example 7d

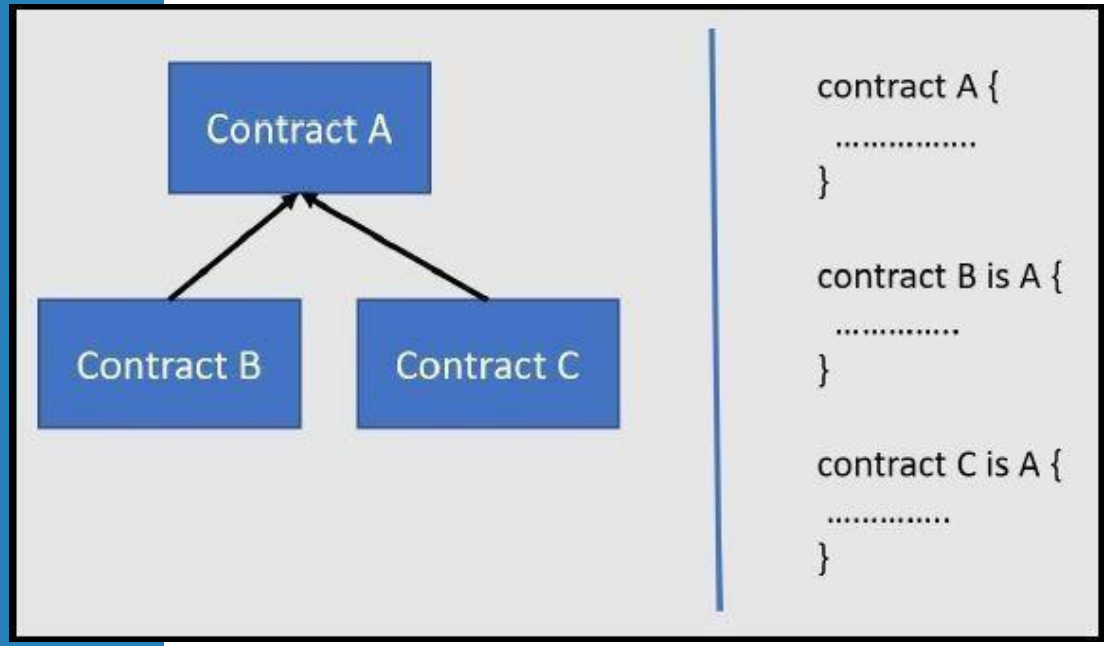
```
contract Client {
    function createObjects() public {
        Human h = new Human();
        Student s =  new
        Student();
        BlockchainStudent bs = new BlockchainStudent();
        bs.getName();
        bs.getAge();
        bs.getCourseName();

    }
}
```

Multi-level Inheritance -- Example 7d

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract Grandparent {
    uint public gValue;
    function setGValue(uint _gValue) public {
        gValue = _gValue;
    }}
contract Parent is Grandparent {
    uint public pValue;
    function setPValue(uint _pValue) public {
        pValue = _pValue;
    }}
contract Child is Parent {
    function getAllValues() public view returns (uint, uint) {
        return (gValue, pValue);
    }}
```

Hierarchical Inheritance

Hierarchical inheritance is again similar to simple inheritance. Here, however, a single contract acts as a base contract for multiple derived contracts.



Hierarchical Inheritance -- Example 8a

```
contract Human { uint
    internal age;
    function setAge(uint _a) public {
        age = _a;
    }
    function getAge() public returns (uint) {
        return age;
    }
}
```

Hierarchical Inheritance -- Example 8b

```
contract Student is Human {
    function getCourseName() public returns (string memory){
        return "Blockchain";
    }
}
contract Teacher is Human {
    function getQualification() public returns (string memory){
        return "Masters";
    }
}
```

Hierarchical Inheritance -- Example 8c

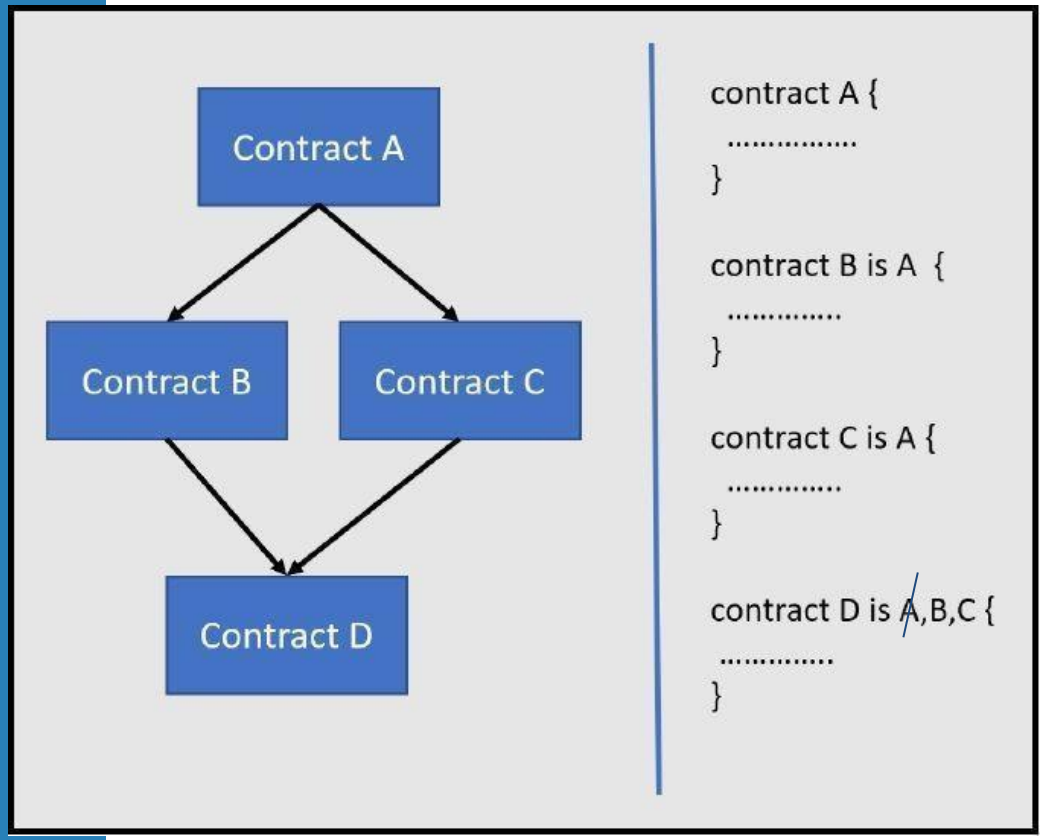
```
contract Client {
    function createObjects() public {
        Human h = new Human();
        Student s =  new
        Student(); Teacher t = new
        Teacher();
        s.getCourseName();
        t.getQualification();

    }
}
```

Hierarchical Inheritance -- Example 8c

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

contract Parent {
    uint public value;
    function setValue(uint _value) public {
        value = _value;
    }}
contract Child1 is Parent {
    function getValue1() public view returns (uint) {
        return value;
    }}
contract Child2 is Parent {
    function getValue2() public view returns (uint) {
        return value;
    }}
```

Multiple Inheritance

Solidity supports multiple inheritance.there can also be multiple contracts that derive from the same base contract. These derived contracts can be used as base contracts together in further child classes. When contracts inherit from such child contracts together, there is multiple inheritance



Multiple Inheritance -- Example 9a

```
contract Human {
}
contract Student is Human {
}
contract Teacher is Human {
}
contract BlockchainStudentAndTeacher is Student, Teacher
{
}
```

Hybrid Inheritance

Hybrid inheritance is a combination of multiple types of inheritance. It may involve a combination of single, multiple, and multilevel inheritance.

Multiple Inheritance -- Example 9a

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract A {
    uint public aValue;
    function setAValue(uint _aValue) public {
        aValue = _aValue;
    }}
contract B is A {
    uint public bValue;
    function setBValue(uint _bValue) public {
        bValue = _bValue;
    }}
contract C {
    uint public cValue;
    function setCValue(uint _cValue) public {
        cValue = _cValue;
    }}
contract D is B, C {
```

ADVANTAGES & DISADVANTAGES OF INHERITANCE IN SOLIDITY

SOLIDITY

Advantages of Inheritance

Code Reusability:

- Inheritance allows you to reuse existing code, reducing redundancy. Common functionalities can be defined once in a base contract and reused in derived contracts.

Modularity:

- Contracts can be split into smaller, manageable pieces. This modularity makes the code easier to maintain, test, and debug.

Extendibility:

- New functionalities can be added to existing contracts without modifying the original code. This makes the contract more adaptable to changes and upgrades.

Advantages of Inheritance

Organization and Readability:

- Inheritance helps in organizing code logically by grouping related functions and variables into a hierarchy. This enhances code readability and comprehension.

Security:

- By inheriting from well-tested and audited contracts (e.g., OpenZeppelin libraries), you can build secure smart contracts with reduced risk of vulnerabilities.

Abstraction:

- Inheritance enables abstraction by allowing base contracts to define common interfaces that can be implemented differently by derived contracts.

Dis-Advantages of Inheritance

Complexity:

- Inheritance can lead to complex and deep inheritance chains, making the code difficult to understand and maintain. This complexity can also introduce bugs that are hard to track down.

Gas Consumption:

- Each inherited contract increases the size of the derived contract, which can lead to higher gas costs when deploying and interacting with the contract.

Diamond Problem:

- When multiple inheritance is used, and contracts have methods with the same name, it can cause conflicts and ambiguities, known as the Diamond Problem, which Solidity partially mitigates using specific rules but still requires careful management.

Dis-Advantages of Inheritance

Security Risks:

- If a base contract has vulnerabilities, all derived contracts inherit those vulnerabilities. Overriding functions or not fully understanding the inherited code can introduce new security risks.

Unintended Behavior:

- Improper use of inheritance, such as unintentionally overriding functions, can lead to unintended and unpredictable behavior in smart contracts.

Dependency on Base Contracts:

- Derived contracts are tightly coupled with their base contracts, making them dependent on the base contract's design and functionality. Any changes in the base contract can have cascading effects on all derived contracts.

Thanks

End of Module-2 (Class-11)