SOLIDITY

DECENTRALIZED SUPPLY CHAIN MANAGEMENT SYSTEM

BY:

SRIVARSHA KAMISHETTY

AAYUSH KUMAR SAKINETI

TULASI MALLAMPATI

PRIYA GANTA

NIKHIL DERANGULA

TABLE OF CONTENTS:

INTRODUCTION:

- Solidity is a high-level programming language used to create smart contracts on the Ethereum blockchain.
- It is statically typed, contract-oriented, and designed to be able to support dApps or decentralized applications.
- Solidity is an object-oriented programming language developed specifically for the creation and design of smart contracts on Blockchain platforms by the Ethereum Network team.
- It finds its application in developing smart contracts that realize business logics and build a chain of records of transactions within the blockchain system.
- It serves as a utility tool for constructing machine-level code and subsequently compiling it on the Ethereum Virtual Machine.
- It carries many similarities with C and C++. Solidity is relatively easy to learn and grasp; for instance, the "main" in C would be the "contract" equivalent in Solidity.

NAMES, BINDING , AND SCOPES:

- Names and Identifiers: Solidity sticks to the naming convention for variables and functions.
- Binding: The variables are statically bound at compile time itself.

Scope:

- Global Scope: msg.sender, block.timestamp, etc., are variables available globally.
- Local Scope: Variables declared inside functions/code blocks

Syntax:

```
uint public count; // global scope
function increment() public
{
        uint localVar = 1; // local scope
        count += localVar;
}
```

DATA TYPES:

- Data types are one of the essential concept in programming language.
- In general, they define what type of data the variables can hold and how the actions on them would be performed.
- In Solidity, data types are categorized into three main types:
  1. Value Types,
  2. Reference Types, and
  3. Complex Types.

- Value types hold their data directly in memory. When you assign a value type to a new variable, a copy of the value is created.

  Common Value Types: Integers, Booleans, Addresses, Fixed-Size Byte Arrays

Syntax:

```
bool isComplete = true; //Boolean
int variableName; and uint variableName; //Integer
address owner = 0x1234567890abcdef1234567890abcdef12345678; // Addresses
function setFixedData(bytes32 _data) public {
        fixedData = _data;
}
```

- Reference types store references to the actual data rather than the data itself. When you assign a reference type to a new variable, both variables point to the same data.

  Common Reference Types: Arrays, Structs, Mappings.

Syntax:

```
uint[] numbers = [1, 2, 3];  // Dynamic array
struct Person {
        string name;
        uint age;
} // Struct
 mapping(address => uint) public balances; //Mappings
```

- Complex types are combinations of value and reference types, often used for more advanced data structures.

  Common Complex Types: Nested Arrays, Nested Struct.

EXPRESSIONS AND ASSIGNMENT STATEMENTS:

- In Solidity, expressions and assignment statements are fundamental components for manipulating data and performing calculations.
- Expressions: An expression is a combination of variables, constants, operators, and function calls that evaluates to a value. Expressions can be simple (involving single variables or constants) or complex (involving multiple variables and operators).
- Types of Expressions:
  1. Arithmetic Expressions: Use arithmetic operators like +, -, *, /, and %.
  2. Logical Expressions: Use logical operators like Logical AND (&&), Logical OR (||), and Logical NOT (!).
  3. Comparison Expressions: Use comparison operators like ==, !=, >, <, >=, and <=.
  4. Bitwise Expressions: Use bitwise operators like Bitwise AND (&), Bitwise OR (|), Bitwise XOR (^), Bitwise NOT (~), Left Shift (<<), and Right Shift (>>).
  5. Assignment Statements: An assignment statement is used to set a variable with a specific value. In Solidity, you typically use the = operator for assignments. Solidity also supports compound assignment operators like +=, -=, *=, and so on.

EXAMPLE:

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

contract ExpressionAndAssignment {
    uint256 public total;       // Variable to store total
    uint256 public count;       // Variable to store count
    bool public isGreater;      // Variable to store comparison result

    // Function to demonstrate arithmetic expressions and assignment
    function calculateTotal(uint256 a, uint256 b) public {
        total = a + b; // Arithmetic expression: addition
    }

    // Function to demonstrate logical expressions and assignment
    function checkGreater(uint256 value1, uint256 value2) public {
        isGreater = value1 > value2; // Comparison expression
    }
```

```
     // Function to demonstrate compound assignment
    function incrementCount() public {
        count += 1; // Compound assignment
    }

   // Function to demonstrate multiple expressions
    function calculateAverage(uint256 a, uint256 b) public view returns (uint256) {
        return (a + b) / 2; // Average calculation using expressions
    }

    // Function to demonstrate a more complex expression
    function evaluateCondition(uint256 a, uint256 b) public view returns (bool) {
        return (a > b && a % 2 == 0); // Logical and arithmetic expressions
    }
}
```

SUPPORT TO OO PROGRAMMING:

- Solidity is a smart contract language that caters for dealing with the Ethereum blockchain and, though commercialized, contains feature, Object-Oriented Programming (OOP).

- Features concerning using Solidity for OO Programming

1. **Polymorphism:** In Solidity, Polymorphism is achieved by function overriding and interfaces. With solid, break-run-time principles. In roots contracts, derived contracts are able to implement functions which were already present in the base contracts giving them special behavior.

2. **Encapsulation (Visibility Modifiers):** Encapsulation in Sol is accomplished by introducing visibility modifier like public, private, internal, external which restricts access to the functions and state variables. Encapsulation in that sense ensures that contracts can have their implementations behind a curtain from other contracts or external users, a feature in OOP.

3. **Constructors:** Constructor is a function within the context of Solidity, and it is advanced and used during deployment to allow a contract state to be formed in a similar way as OOP constructors.

4. **Multiple inheritance and C3 Linearization:** Solidity supports multiple inheritance and that is the possibility of one contract inheriting many parent contracts. This is because Solidity uses C3 linearization to solve the method conflict problems which result from multiple inheritance and therefore produces a proper method resolution order.

EXAMPLE:

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

// Base contract representing an Animal
contract Animal {
    // Private state variable encapsulating the species
    string private species;

    // Constructor to initialize the species of the animal
    constructor(string memory _species) {
        species = _species; // Setting species
    }
// Public function to get the species (demonstrating encapsulation)
    function getSpecies() public view returns (string memory) {
        return species;
    }

    // A virtual function that can be overridden in derived contracts
    function sound() public virtual pure returns (string memory) {
        return "Some sound";
    }
}
```

OUTPUT:

Deploy Dog Contract:	Deploy Cat Contract:
Call: getSpecies()	Call: getSpecies()
Output: "Dog"	Output: "Cat"
Call: sound()	Call: sound()
Output: "Woof!“	Output: "Meow!"

CONCURRENCY:

- Concurrency is a fundamental aspect as it allows multiple transactions to interact with the Ethereum blockchain concurrently.
- It is a major component of the design of smart contracts and may present multiple challenges especially as concerns state race and these transactions order.
- The following section addresses concurrency on Solidity providing in depth information on the subject.

1. **Transaction Ordering:** Ethereum implies a one-by-one processing model for each transaction with each transaction having a unique nonce. Its implication is that the transaction is executed as it was when submitted.
2. **Re-entrancy:** This pertains to another contract being called from within a contract or from amongst its functions and the called contract is allowed to modify state before exiting the calling contract, thereby creating potential security issues.
3. **State Changes:** By invoking functions, Solidity provides you with a way to change the state of your contract. However, if state changes are induced in several transactions for the same state variable, this may result in unpredictable behavior.

EXAMPLE:

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

contract SimpleToken {
    mapping(address => uint256) public balances;

    // Function to mint tokens to an address
    function mint(address _to, uint256 _amount) public {
        balances[_to] += _amount; // Add tokens to the recipient's balance
    }

    // Function to transfer tokens
    function transfer(address _to, uint256 _amount) public {
        require(balances[msg.sender] >= _amount, "Insufficient balance."); // Check balance
        balances[msg.sender] -= _amount; // Subtract from sender's balance
        balances[_to] += _amount; // Add to recipient's balance
    }
}
```

OUTPUT:

Operation	Output
User A mints 10 tokens	User A's balance: 10 tokens
User A transfers 5 tokens to User B	User A's balance: 5 tokens
	User B's balance: 5 tokens
User A attempts to transfer 7 tokens to User C	Error: "Insufficient balance."

EXCEPTION HANDLING AND EVENT HANDLING

In Solidity, exception handling and event handling are crucial for building robust smart contracts. They enable developers to manage errors gracefully and provide a way to log important contract activity, respectively.

1. Exception Handling: Exception handling in Solidity is primarily done using require, assert, and revert. These mechanisms allow developers to enforce conditions and handle errors effectively.

a. require: Used to validate inputs and conditions before executing a function.

If the condition fails, it reverts the transaction and provides an optional error message.

b. assert: Used to check for conditions that should never be false (invariants).

If the assertion fails, it throws an exception and consumes all gas.

c. revert: Used to revert the transaction with a custom error message.

Useful for more complex conditions or to revert from nested calls.

2. Event Handling: Events in solidity provide a logging mechanism that allows contracts to communicate with external application. They are crucial for monitoring state changes and contract activity.

a. Defining Events: Events are defined using the event keyword. They can include indexed parameters to allow for efficient searching and filtering

EXAMPLE:

```
pragma solidity ^0.8.0;
contract EventHandling {
    event Deposit(address indexed user, uint256 amount);
    event Withdraw(address indexed user, uint256 amount);
    mapping(address => uint256) public balances;

    // Function to deposit funds
    function deposit() public payable {
        require(msg.value > 0, "Deposit amount must be greater than 0");
        balances[msg.sender] += msg.value; // Update balance
        emit Deposit(msg.sender, msg.value); // Emit event
    }
// Function to withdraw funds
    function withdraw(uint256 amount) public {
        require(balances[msg.sender] >= amount, "Insufficient balance");
        balances[msg.sender] -= amount; // Update balance


        emit Withdraw(msg.sender, amount); // Emit event
        payable(msg.sender).transfer(amount); // Transfer funds
    }
}
```

OUTPUT:

Event: Deposit	Event: Withdraw
Alice calls deposit() and sends 1 Ether	Alice calls withdraw(0.5 ether)
Parameters:	Parameters:
- user: Alice's address (e.g., 0xABC...)	- user: Alice's address (e.g., 0xABC...)
- amount: 1 Ether	- amount: 0.5 Ether

FUNCTIONAL PROGRAMMING

- Functional Programming (FP) is a style concentrating on utilizing functions effectively as the programming constructs and avoiding any kind of mutations and external variables. The language of Solidity, originally procedure, supports environment concept and design in programs, the essence of pure and functions and of not changing the state of the transaction and passing around functions in the code.
- Key Functional Programming Concepts in Solidity;

1. **Pure Functions:** For the pure functions, they got no right to change the public state or data and do not read it. Such functions don't have any other significant impact and return the same output value given the same input. This coincides with the functional programming model of the pure functions.
2. **Immutability:** Data present in a function call in Solidity is by default unchangeable if it is not marked mutable. In the same vein, immutability is essential in FP which does not allow the transformation of functions and data.

3. View Functions: View functions can read the state of the contract but are unable to change it. Indeed, since these methods do not modify the state of the program, they can be viewed as an aspect of functional programming and can be thought of as equivalents of ‘read-only’ functions.

EXAMPLE:

```
pragma solidity ^0.8.0;
contract FunctionalExample {
        // Pure function: No state change, deterministic output based on inputs
        function add(uint256 a, uint256 b) public pure returns (uint256) {
                return a + b;
        }
}
```

INPUT:

A=3, B=5

OUTPUT:

8

OBJECTIVE OF THE PROJECT:

- This project offers an overview of developing a blockchain-based decentralized supply chain management system that will improve the supply chain in terms of transparency, traceability, and efficiency by allowing stakeholders to track the origin and status of goods, verify their authenticity, and audit trails.

Enhanced Supply Chain Tracking:

- **Real-time Tracking:** Allow all the concerned persons to view in real time the location of the goods and their conditions right from manufacturing up to delivery.Visibility and Transparency: Allow full transparency in the supply chain, enabling the stakeholder to see every step of the process right from getting raw materials to the final delivery.

Provenance Verification:

- **Authenticity Verification:** It authenticates a product since it verifies its origin and movement within the supply chain.
- **Anti-Counterfeiting:** There is less likelihood of counterfeit products within the supply chain. This is because the blockchain guarantees that the origin and movement history of goods are tamper-proof and secure.

Testing and Transparency

- Immutable Audit Trails: Make a non-reversible record of the transaction and movement taking place along the supply chain. Data accuracy is guaranteed with responsibility.
- Regulatory Compliance: Assist in achieving regulatory compliance based on auditable records of each supply chain activity.

Stakeholder Collaboration

- Decentralized Data Sharing: It provides stakeholders-suppliers, manufacturers, distributors, or retailers-with the security and transparency of data sharing.
- Smart Contracts: These are used in the automation of implementation and enforcement of agreements between different parties. This reduces the influence of middlemen; hence disputes are minimal.

Efficiency and Cost Reduction

- Enhance the supply chain process in eliminating unnecessary steps that delay the team's performance and create lateness. Reduce paperwork, smoothen operations, thereby reducing costs through improving inventory management.

CONSTRAINTS

- Statements on the issues of a decentralized supply chain management system. Linear restrictions on centralization

Scalability:

- **Block chain scalability:** most block chain networks often deal with disproportion swelling. Especially, with degree of transaction executions and time delays where volume is of the essence, constructing such networks may be very tasking to any supplier.
- **Duplicating data:** information stored in block chain always tends to be large by repeating the same information. Storage and retrieval is bound to be a challenge because of the tendency to keep one copy of the same data in one place.

Cross platform tanzac control:

- It also should become a hard task to have a proper blockchain and other existing supply chain systems in place (ERP, WMS, TMS), sometimes even new ones, in the network.
- Data standards: different entities may have their own data sets and standards for internal use, and this causes barriers in data integration that needs to be done in a consistent manner.

Security – Ethical and Legal Issues:

- **Information privacy:** Concerning the business secrets without being too moribund at the same time and resorting to the public or a consortium blockchain in order not to overcome the desired, clear and legible tracking of the happy papers.
- **General Cybersecurity:** Resistance against the unlawful intrusion of the blockchain network in the form of cyber-assaults like hacking, data breaching, DDoS and more.

Limitation to Smart Contract Adoption:

- **Complications:** It is difficult to create and apply smart contracts especially in cases that involve a lot of business for instance regulatory and complex business requirements.
- **Exploitative:** The security of smart contracts is undermined due to presence of bugs and design faults warranting an audit to be conducted.
- **Operational Constraints**

Cost:

- **Implementation Costs:** The start-up budget for a block chain-based supply chain system including hardware, software, requirements, developers and acceptance rates is bound to be expansive.
- **Maintenance Costs**: The other monetary obligations for the development of the network or renewing the functioning of smart contracts and system safeguarding.
- **Admission and Training**
- **Stakeholder switch**

FEATURES

Characteristics of the decentralized supply chain management system

1. Real-time supply chain tracking:

- **Integration of GPS and iot:** always monitoring the movement and well-being of products around the clock through GPS and iot sensors.
- **Visualization via dashboards:** providing end users with graphical representations of the movement of goods through the supply chain which helps in performance management and choke point identification.

2. Authenticity of product:

- **Use of block chain:** this ensures that manufacturing information of products are not only from raw materials to the company but also from agreements to finished products which create quite a lot of controversy over authenticity.
- **Issuing digital certificates:** in confirming the sources of each product from the very beginning, digital certificates are provided to all parties involved in the particular product.

3. Auditing and Transparency:

- **Immutable Auditor’s Flow:** Everything taking place in the supply chain, even the slightest occurrence in a transaction is captured on blockchain to produce a detailed and permanent record of activity thus enhancing the level of openness and framing of accountability.
- **Legislative Requirements:** acceptance of the rules and regulations of the industry is made easier by incorporations of these contracts and compliance checks there are included and performed.

4. Smart Contracts:

- **Automated Contract Calls:** To systematize the transaction engaged by the parties, the use of the smart contracts is made hereby reducing the human involvement and disputes.
- **Contract adjustments using code however, temporary also possible:** It is also possible to allow the contract execution terms to be altered temporarily solely under a certain condition and based on present information so as to allow rooms of strategies within extraordinary circumstances.

TECHNOLOGIES USED:

Blockchain technology:

Platforms: This includes the use of blockchain technologies offered by services such as ethereum, hyper ledger fabric, or quorum to develop a record of all supply chain related processes.

Smart contracts: It is obvious that agreements need to be enforced during course of business dealing and thus smart contracts can be utilized to deliver this target. The system eliminates the intermediary and ensures that once deployed, the contract cannot be tampered with.

Consensus mechanisms: another strategic way of formulating and approving transactions is the use of consensus algorithm. This base attack logic includes proof of stake (pos) or practical byzantine fault tolerance (PBFT)

Artificial intelligence (ai) and machine learning (ml):

- Predictive analytics: how to use AI and machine learning algorithms in demand forecasting, risk management and advanced maintenance by studying historical and real-time data.
- Primary data – here, ml models help to analyze the vast sources of supply chain data and provide recommendations tightly i.E., Decision making.

PROJECT APPLICATIONS :

1. Food and Beverage Industry

- **Traceability:** Knowing the origin and keeping a record of ingredients is an important process in quality and safety management. The process enables easier identification and isolation of the lot when there is any foodborne illness.
- **Sustainability:** The monitoring and verification of environmental farming practices enhance compliance with environmental regulations and meet consumer demands for greener products.

2. Pharmaceutical Industry

- **Prevention against Counterfeit:** Verification at critical points along the supply chain is very important in the field of pharmaceuticals to prevent fraudulent drugs from reaching consumers.
- **Regulatory Compliance:** Records should be maintained in such a fashion that they make any regulatory audit easy concerning health-related regulations, for instance the Drug Supply Chain Security Act (DSCSA) and EU Falsified Medicines Directive (FMD).

3. Automotive Industry

- **Parts Traceability:** Since quality and authenticity depend on it, the origin and movement of the auto parts need to be tracked correctly. It helps in smooth recall of defective parts and handling of warranty claims.
- **Supplier Collaboration:** Better collaboration will help suppliers cut lead times and improve procurement efficiency.

4. Fashion and Apparel Industry

- **Ethical Sourcing:** Materials shall be sourced ethically, considering sustainable labor practices. That is adding value to a brand and creating goodwill, especially when consumers support ethically produced products.
- **Inventory Management:** Better visibility and management of inventory will avoid overproduction, resulting in less waste by aligning it to demand.

5. Electronics Industry

- **Component Authentication:** This becomes very important because a great number of electronic parts are counterfeit and may lead to product failure and serious safety issues.