**Unit 4 – Blockchain and Decentralized Applications**

**Lab 6 Manual**

**Objective:**

In this lab, students will learn and implement inheritance and mappings across inherited contracts for data management.

By the end of this lab, students will be able to:

* Understand the role of events and emits in Solidity smart contracts
* Explore the structure of event logs and how to retrieve event data from the blockchain
* Gain hands-on experience with filtering and subscribing to events

**Events:**

Events are logs or records created within a smart contract that allow external entities (e.g., user interfaces) to react to specific occurrences. Emits are keywords used to trigger events in Solidity. Events facilitate transparency and auditable tracking of important contract interactions. They log event data onto the blockchain as part of the transaction history.

* 1. **Defining and Emitting Events**

**A screen shot of a computer program

Description automatically generated**

* 1. **Retrieving Events (A)**

A screen shot of a computer program

Description automatically generated

* 1. **Retrieving Events (B)**

A screen shot of a computer program

Description automatically generated

* 1. **Event Filters**

A screen shot of a computer program

Description automatically generated

* 1. **Subscribing to Events**

A screen shot of a computer program

Description automatically generated

* Some more about events (i)

A screenshot of a computer program

Description automatically generated

**Byte32 name:** 0x546f6b656e4e616d650000000000000000000000000000000000000000000000

Output:

A screenshot of a computer program

Description automatically generated

* Some more about events (ii)

A computer screen shot of a program

Description automatically generated

Output

A screenshot of a computer program

Description automatically generated

**Modifier:**

A computer screen shot of a program code

Description automatically generated

**Payable: A**

A computer screen shot of a program code

Description automatically generated

In the above example we send to the contract address but in the below contract we will send it on the owner address:

**Payable: B**

A computer screen shot of code

Description automatically generated

**Transfer:**

A computer screen shot of text

Description automatically generated

**Task:**

Make a token and deploy it with the MetaMask account. Here is the code, ERC-20-Token.sol, which is successfully deployed and also imported it token into Metamask account. See GitHub

Interface: In Solidity, an **interface is a way to define a set of function signatures that other contracts must follow to if they want to be considered as implementing** that interface. Interfaces are used to establish a common communication **protocol between different contracts, allowing them to interact seamlessly without knowing the implementation details of each other.**

Save the following code in ./IERC20.sol

All these functions are implanted in ERC-20-Token, here we only declare the function.

//SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.8.0;

interface ERC20Interface {

    function transfer(address to, uint256 value) external ;

    function approve(address spender, uint256 value) external;

    function getAllowance(address from, address spender) external view ;

    function minting(address \_initAdd, address \_toAdd, uint \_mintAmount) external ;

     function burning(address \_initAdd, address \_toAdd, uint \_burAmount) external ;

}

Now we call any of this function in another contract, name as ERC20-InterfaceCall.sol. But we need first to deploy the ERC20-token file, and pass his contract address to the below code, then it will transfer the amount to another address.

// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.8.0;

import "./IERC20.sol";

contract xzy{

ERC20Interface token;

address public owner=msg.sender;

constructor(address \_tokenAddress){

token=ERC20Interface(\_tokenAddress);

}

function sellToken(address \_to, uint \_amount)public {

token.transfer(\_to,\_amount);

}}