Lab 4

1. Constant vs Immutable Variables

In Solidity, there are two main types of variables that can't be changed after being set: constant and immutable.

a. Constant Variables:

i. Can be viewed as compile-time constants. They cannot be modified and their values have to be set at compile time.
ii. Suitable for values which will never change across all instances.
iii. Example :

```
pragma solidity ^0.8.0;
contract ConstantExample {
      uint256 constant MY_CONSTANT = 12345;
}
```

b. Immutable Variables:

i. These variables can be assigned once during the contract creation but cannot be modified after that.
ii. The actual value of immutable variables is set in the constructor and gets stored in the bytecode of the contract.
iii. Example :

```
pragma solidity ^0.8.0;
contract ImmutableExample {
      uint256 public immutable creationTimestamp;
      constructor() {
            creationTimestamp = block.timestamp;
      }
}
```

2. Optimization using Constant or Immutable Variables

Constant and immutable variables can optimize gas usage and performance in Solidity contracts:

- They reduce storage costs. In the case of constant, the value doesn't need to be stored on-chain at all. For immutable, the value gets baked into the contract's bytecode and doesn't need a separate storage slot.

Example: Suppose you have a contract that needs to know the maximum number of tokens for its logic. Instead of storing it as a state variable, you can use an immutable variable if it's decided at deployment or constant if it's a pre-defined value.

```
pragma solidity ^0.8.0;

contract Token {
    uint256 public immutable MAX_SUPPLY;

    constructor(uint256 maxSupply) {
        MAX_SUPPLY = maxSupply;
    }
}
```

3. Diamond Pattern

The Diamond pattern addresses Ethereum's contract size limits by dividing a contract's functionality into smaller parts called "facets".
Key Components:

- Diamond Contract (Dispatcher): Users interact with this main contract. It doesn't hold business logic but delegates calls to the appropriate facet based on the function invoked.
- Facets: These are smaller contracts containing specific logic. The Diamond contract delegates function calls to these facets.

Advantages:

- Enables scalability beyond contract size limits.
- Facilitates contract upgradability.
- Offers modular development, where facets can be independently managed or upgraded.

In essence, the Diamond contract acts as a router, directing function calls to the appropriate facet for execution, thus allowing for modular and upgradable contract structures.

Github link : LINK

Reference :

ERC-2535: Diamonds, Multi-Facet Proxy
Diamond Standard with Nick Mudge | Solidity Fridays
GitHub - mudgen/diamond: Information about three diamond reference implementations.
Louper - Diamond