Activity 6: ERC20 applications.

Solidity basics

Libraries

Libraries are an efficient method to reuse code. Libraries are not intended for altering the state of the contract; they can only be employed to execute fundamental operations based on inputs and outputs. Libraries are similar to smart contracts, with the following differences:

- **syntax**: library key-word is used instead of contract key-word.
- **stateless:** a library does not have state variables. From an object-oriented perspective, it cannot inherit any element and cannot be inherited.
- **gas cost:** reusing code by creating a library reduces the gas cost.
- **selfdestroy**: Unlike other contracts, a library cannot be destroyed.
- Functions of the library can be called directly when they do not modify the state variables i.e. only pure and view functions can be called from outside of the library.

Libraries can be defined in the same contract file or imported with **import** statement. A single file can contain multiple libraries.

Proxy Pattern

Smart contracts are immutable, once deployed a smart contract cannot be altered. This feature, essential for Blockchain to ensure safe transfers, is an impediment when one wants to resolve bugs or add new features to existing applications. The proxy pattern helps creating upgradable contracts, helping users to migrate to new versions of DApps without changing any reference to new smart contracts. To implement proxy pattern, we must create two components:

- Proxy contract: the interface between users and other contracts. The proxy contract keeps the addresses of the business logic contracts. All calls are managed by the proxy. The proxy contract forwards the calls to the business logic contracts.
- Business logic contract: the contract with actual functionality. This component may be upgraded without changing the address of the proxy contract.

Some advantages of the proxy pattern are rollback capability (older versions can immediately be restored), version control (a proxy can interact with multiple versions of the business logic contract), availability (the upgrade process doesn't interrupt the interaction between users and dApp). On the downside managing proxy contracts add complexity to the system architecture and can lead to higher gas costs due to additional function calls.

Often Proxy Pattern is used in conjunction with a storage pattern (a storage contract holds relevant data, logic contract interacts with the storage to read or write data) and with delegatecall pattern (using the delegatecall opcode to forward function calls to a contract containing the actual logic.

Interfaces

[**TODO – insert general info**]

Delegatecall

[**TODO – insert general info and example**]

EXERCISES

1. Test functions in contract TestLib.sol. Compile with version < 0.8.0.
   - Test addition of 255 and 10 using testAddUnsafe and testAddUnsafe.
   - Test token generation.
   - Test constants initialization.

2. Work on MyERC20.sol. Replace math operations with functions from SafeMath lib.
   - Add: import "@openzeppelin/contracts/utils/math/SafeMath.sol";
   - Add: using SafeMath for uint;
   - Use add and sub functions instead of "+" and "–"
   - **OBS**: Starting from Solidity version 0.8, integer overflows or underflows will automatically revert.
3. Develop a game based on ERC token. Use the ERC20 token developed in **Lab4** or an openzeppelin implementation. Analyze and test Game.sol.
   Discuss optimizations:
   - Observe how we access a value of type array or struct associated to a key of a mapping. **storage** keyword is mandatory. Check *startGame* function.
   - Function endgame: Can high gas costs in loops be avoided?
   - Function round: Why is not safe to use timestamp in RNG. What kind of oracles can act as RNGs? etc.
   - Why is not safe to use tx.origin. Study **testGame.js**. Try truffle test --show-events and check that game with id 1 is ended by the attacker (account2) and not by the dealer (account1).
     Run

     truffle test --show-events

     Replace function endGame(uint256 _gameId) public onlyDealer(**tx.origin**, _gameId)

     with function endGame(uint256 _gameId) public onlyDealer(**msg.sender**, _gameId) and re-run the tests.

     Analyze the example in [4].

4. Deploy two versions of Game contract using Proxy Pattern and *delegatecall*.

BIBLIOGRAPHY

1. https://dl.acm.org/doi/pdf/10.1145/3276486
2. https://neptunemutual.com/blog/issues-with-authorization-using-txorigin/ or another example
3. https://medium.com/coinmonks/delegatecall-calling-another-contract-function-in-solidity-b579f804178c or another example
4. https://solidity-by-example.org/hacks/re-entrancy/