Final Project Design Patterns and Security Measures

This document will list the design patterns and attack vectors in the course.

In the final project requirements, available in the course here, there is a requirement:

> **Use at least two design patterns from the "Smart Contracts" section**
>
> **Protect against two attack vectors from the "Smart Contracts" section with its SWC number**

Below is a list of **design patterns** in the Smart Contract chapter, along with a short description and the title of the lesson where it's mentioned. To meet the requirement, you need only **two** of the following, documented in your `design_pattern_decisions.md`:

- **Inter-Contract Execution** (Calling functions in external contracts) *Inter-Contract Execution, Part 1 and Part 2*
- **Inheritance and Interfaces** (Importing and extending contracts and/or using contract interfaces) *Inheritances and Interfaces* — ***(note: this is already a requirement in the final project, so you can simply describe which library or interface you use)***
- **Oracles** (retrieving third-party data) *Off-Chain Oracles* and *Chapter 5: Second-Order Effects — Oracles Revisited*
- **Access Control Design Patterns** (Restricting access to certain functions using things like Ownable, Role-based Control) *Access Control Design Patterns*
- **Upgradable Contracts** (Ways to update a deployed contract's logic or data) *Upgradable Contracts* and *Additional Material: Upgradable Contracts*

- **Optimizing Gas** (Creating more efficient Solidity code) *Optimizing Gas*

Below is a list of attack vectors and / or security measures from the course, specifically *Solidity Pitfalls and Attacks* and *Smart Contract Pitfalls and Attacks*. It is okay for some of these to overlap with design patterns, but you can list **at least two** of them in `avoiding_common_attacks.md`:

From *Solidity Pitfalls and Attacks*

- **Using Specific Compiler Pragma**
- **Proper Use of Require, Assert and Revert**
- **Use Modifiers Only for Validation**
- **Pull Over Push** (Prioritize receiving contract calls over making contract calls)
- **Checks-Effects-Interactions** (Avoiding state changes after external calls)
- **Proper use of .call and .delegateCall**

From *Smart Contract Pitfalls and Attacks*

- Not everything can be avoided, but you can write if you're taking protection against:
    - **Re-entrancy**
    - **Timestamp Dependence**
    - **Forcibly Sending Ether**
    - **Tx.Origin Authentication**