Lesson 13 - Cairo contract continued / Upgradability

Upgrading Contracts

See Docs
The extensibility pattern is followed.

General Workflow

1. Declare an implementation contract class.
2. Deploy the proxy contract with the implementation contract's class hash, and the inputs describing the call to initialize the proxy from the implementation

Proxy contract

The proxy contract is used to redirect calls to an implementation contract, it is the implementation contract that we change.
Users will interact with the proxy contract, and thus an unchanging contract address.

How upgradability is implemented in the proxy contract.

There are 2 important functions

1. The `__default__` method is a fallback method that redirects a function call and associated calldata to the implementation contract.
2. The `__l1_default__` method is also a fallback method; however, it redirects the function call and associated calldata to a layer one contract. In order to invoke `__l1_default__`, the original function call must include the library function `send_message_to_l1`. See Cairo's Interacting with L1 contracts for more information.

The proxy needs to know the hash of the implementation contract, this will be passed to the proxy contract at deployment time.

Implementation Contract

The implementation contract should follow the Extensibility pattern and import directly from the Proxy library.

The implementation contract should:

- Import `Proxy` namespace.
- Provide an external initializer function (calling `Proxy.initializer`) to intialize the proxy immediately after deployment.

If the implementation is upgradeable, it should:

- Include a method to upgrade the implementation (i.e. `upgrade`).
- Use access control to protect the contract's upgradeability.

The implementation contract should NOT:

- Be deployed like a regular contract. Instead, the implementation contract should be declared (which creates a `DeclaredClass` containing its hash and abi).
- Set its initial state with a traditional constructor (decorated with `@constructor`). Instead, use an initializer method that invokes the Proxy `constructor`.

For example

```
// SPDX-License-Identifier: MIT

%lang starknet

from starkware.cairo.common.cairo_builtins import HashBuiltin

from openzeppelin.upgrades.library import Proxy

@storage_var
func value() -> (val: felt) {
}

//
// Initializer
//

@external
func initializer{syscall_ptr: felt*, pedersen_ptr: HashBuiltin*,
range_check_ptr}(
    proxy_admin: felt
) {
    Proxy.initializer(proxy_admin);
    return ();
}

@view
func getValue{syscall_ptr: felt*, pedersen_ptr: HashBuiltin*,
range_check_ptr}() -> (val: felt) {
    return value.read();
}

@view
func getAdmin{syscall_ptr: felt*, pedersen_ptr: HashBuiltin*,
range_check_ptr}() -> (
    admin: felt
) {
    return Proxy.get_admin();
}

@external
func setValue{syscall_ptr: felt*, pedersen_ptr: HashBuiltin*,
range_check_ptr}(val: felt) {
    value.write(val);
    return ();
}
```

```
@external
func setAdmin{syscall_ptr: felt*, pedersen_ptr: HashBuiltin*,
range_check_ptr}(address: felt) {
    Proxy.assert_only_admin();
    Proxy._set_admin(address);
    return ();
}
```

API

```
func initializer(proxy_admin: felt) {
}

func assert_only_admin() {
}

func get_implementation_hash() -> (implementation: felt) {
}

func get_admin() -> (admin: felt) {
}

func _set_admin(new_admin: felt) {
}

func _set_implementation_hash(new_implementation: felt) {
}
```

Further Maths libraries

1. Cairo Math 64x61
   See Repo

   Gives Signed 64.61 Fixed Point Numbers with operations `add`, `sub`, `mul`, `div`, `sqrt`, `exp`, `ln`, `log2`, `log10`, and `pow` as well as conversion to / from felts and Uint256 values, `floor`, `ceil`, `min`, `max` and assertion methods.
   Also contains Trigonometry, Hyperbolic and Vector libraries.

2. CairoMath
   See Repo

3. The Cairo bitwise integer library
   Repo

4. xoroshiro-cairo randon numbers
   Repo

5. Nethermind Cairo Safe Math
   Repo

6. Finite field operations.
   Repo

7. Maths-Physics Special Functions
   Repo

8. Multi Precision Cairo (library for 384 bit numbers)
   Repo

9. Open Zeppelin Safe Math
   Repo

Shard Labs Devnet

See Docs
Allows a ganache like devnet.
You can use it with protostar with the following lines in the configuration file

```
[profile.devnet.project]
gateway-url = "http://127.0.0.1:5050/"
chain-id = 1536727068981429685321
```

Hardhat plugin

See Docs

Installation

Install with
`npm i @shardlabs/starknet-hardhat-plugin --save-dev`

You will need to add the following to your hardhat config

```
import "@shardlabs/starknet-hardhat-plugin";
// or
require("@shardlabs/starknet-hardhat-plugin");
```

Network Configuration

You can add the network details to the config file

```
module.exports = {
  starknet: {
    network: "integrated-devnet"
  },
  networks: {
    integratedDevnet: {
      url: "http://127.0.0.1:5050",

      // venv: "active" <- for the active virtual environment with
installed starknet-devnet
      // venv: "path/to/venv" <- for env with installed starknet-devnet
(created with e.g. `python -m venv path/to/venv`)
      venv: "<VENV_PATH>",
```

```
      // or specify Docker image tag
      dockerizedVersion: "<DEVNET_VERSION>",

      // optional devnet CLI arguments
      args: ["--lite-mode", "--gas-price", "2000000000"],

      // stdout: "logs/stdout.log" <- dumps stdout to the file
      stdout: "STDOUT", // <- logs stdout to the terminal
      // stderr: "logs/stderr.log" <- dumps stderr to the file
      stderr: "STDERR"  // <- logs stderr to the terminal
    }
  }
  ...
};
```

Available networks are then

- `--starknet-network alpha-goerli` for Alpha Testnet (on Goerli)
- `--starknet-network alpha-mainnet` for Alpha Mainnet
- `--starknet-network integrated-devnet` for integrated Devnet

Some available commands

Compile

```
npx hardhat starknet-compile [PATH...] [--cairo-path "<LIB_PATH1>:
<LIB_PATH2>:..."] [--account-contract] [--disable-hint-validation]
```

Deploy

```
npx hardhat starknet-deploy [--starknet-network <NAME>] [--wait] [--
gateway-url <URL>] [ARTIFACT_PATH...] [--inputs <CONSTRUCTOR_ARGUMENTS>]
[--salt <SALT>]
```

Invoke

```
npx hardhat starknet-invoke [--starknet-network <NAME>] [--gateway-url
<URL>] [--contract <CONTRACT_NAME>] [--address <CONTRACT_ADDRESS>] [--
function <FUNCTION_NAME>] [--inputs <FUNCTION_INPUTS>] [--signature
<INVOKE_SIGNATURE>] [--wallet <WALLET_NAME>]
```

Testing with Starknet

Example

```
import { expect } from "chai";
import { starknet } from "hardhat";


describe("My Test", function () {

this.timeout(30_000); // 30 seconds - recommended if used with starknet-devnet
```

```
it("should work with old-style deployment", async function () {
    const account  = ...;
    const contractFactory = await starknet.getContractFactory("MyContract");

    await account.invoke(contract, "increase_balance", { amount: 10 }); // invoke method by name and pass arguments by name
    await account.invoke(contract, "increase_balance", { amount: BigInt("20") });

    const { res } = await contract.call("get_balance"); // call method by name and receive the result by name
    expect(res).to.deep.equal(BigInt(40)); // you can also use 40n instead of BigInt(40)
  });

  it("should load a previously deployed contract", async function () {
    const contractFactory = await starknet.getContractFactory("MyContract");
    const contract = contractFactory.getContractAt("0x123..."); // address of a previously deployed contract
  });

  it("should declare and deploy", async function() {
    const contractFactory = await starknet.getContractFactory("MyContract");
    const account = await starknet.getAccountFromAddress(...);

    // You are expected to have a Deployer contract with a deploy method
```

```
    const deployer = await starknet.getContractFactory("Deployer");
    const classHash = await account.declare(contractFactory, { maxFee: ...
});
    const opts = { maxFee: BigInt(...) };
    const txHash = await account.invoke(deployer, "my_deploy", {
class_hash: classHash }, opts);
    const deploymentAddress = ...; // get the address, e.g. from an event
emitted by deploy
    const contract = contractFactory.getContractAt(deploymentAddress);
  });
```

Tayt - Contract Fuzzer

See Repo

Writing invariants

Invariants are StarkNet view functions with names that begin with `tayt_`, have no arguments, and return a felt. An invariant is considered failed when it returns 0.

```
@view
func tayt_flag{
        range_check_ptr,
        syscall_ptr: felt*,
        pedersen_ptr: HashBuiltin*
    }() -> (res: felt):
    let (flag_result) = flag.read()
    if flag_result == 1:
        return (0)
    end
    return (1)
end
```

Allows testing the invariants and has a coverage option.

Starkscan Source Code Verifier

See Repo
Anyone can upload any ABI to Starknet and block explorers will assume it is correct when it doesn't have to be.
See this community post

You can use it with Protostar or Nile.

Install as a npm package
`npm install -g starkscan`

Starklings Tutorial

See repo

Installation

```
git clone --branch stable --single-branch
https://github.com/onlydustxyz/starklings.git
```

and install with

```
curl -L
https://raw.githubusercontent.com/onlydustxyz/starklings/master/install.sh |
bash
```

Running the tutorial

```
starklings --watch
```

L3 - Slush SDK

See Repo
The project allows you to create L3 nodes (based on Tendermint consensus).
You need to link the L3 to nodes on L2 (and thereby to the L2 contract)
Still a work in progress.

Authentication

Sign in with Starkware

See article
See Docs
Sign-in with StarkWare brings on-chain authentication to Web 2.
A user can log in to a Web2 application using their StarkWare account.
See Demo

JWT on Starknet

An implementation of JSON Web Tokens.
See Repo