Principles of Distributed Ledgers: Tutorial 2

In this tutorial, the goals are the following:

1. Implement an ERC-20 token representing a currency
2. Implement an ERC-721 token (NFT) that can be purchased using the ERC-20 token created in part 1

These tasks are described in detail below.

0 Initial setup

0.1 Setting up coding environment

In this course, we will use Foundry to develop smart contracts. The simplest way to install Foundry is to use their official installation script for `foundryup` (a tool to manage Foundry's installation):

```
curl -L https://foundry.paradigm.xyz | bash
```

and then run

```
foundryup
```

For those who are (rightfully) uncomfortable piping a shell script directly from the Internet, the content of the official installation script can be inspected and the installation steps reproduced very easily.

If the installation is successful, the `forge` command should become available. It can be checked using

```
forge --version
```

which returns `forge 0.2.0` at the time of writing.

0.2 Using the skeleton

A skeleton project is provided here: `https://gitlab.doc.ic.ac.uk/podl/2023/tutorial-2-skeleton`. The skeleton contains contract files as well as a set of tests.

The following commands will fetch and build the projects:

```
git clone https://gitlab.doc.ic.ac.uk/podl/2023/tutorial-2-skeleton
cd tutorial-2-skeleton
forge build
forge test
```

The above run of `forge test` will fail, as the contracts are not implemented yet. While implementing the contracts, the tests can be run using `forge test` to check that the implementation is correct. Below are some useful flags for `forge test`:

```
forge test --mc ERC20Test   # run all tests in the ERC20Test contract
forge test --mc ERC20Test --mt testName  # run test called testName in ERC20Test
forge test --nmc Bonus  # run all tests apart from the bonus ones
forge test --mc ERC20Test -vvv  # run tests in ERC20Test with a verbose output
```

More information can be found in the Foundry documentation.

1 Implementing an ERC-20 token

ERC-20 is a token standard often used to implement tokens representing a currency. These tokens are fungible, which means that each token has the same value as any other token.

The ERC-20 token standard defines the following functions:

`totalSupply` Retrieve the total number of tokens in existence

`balanceOf` Retrieve the balance of a user

`transfer` Transfer tokens to another user

`approve` Approve another user to transfer tokens on the user's behalf

`allowance` Retrieve the number of tokens that a user is allowed to transfer on another's half

`transferFrom` Transfer tokens on another user's behalf

Although not mandatory in the standard, most tokens (including ours) also allow querying the following metadata:

`name` The name of the token

`symbol` The symbol (ticker) of the token, e.g. DAI

`decimals` The number of decimals by which the amounts are scaled, e.g. 18 if the amounts are scaled by 10^{18}

More information about each of these functionalities can directly be found in the ERC-20 standard.

In this tutorial, our ERC-20 token will have a few particularities:

- The name and symbol are passed into the constructor
- The total supply is initially set to 0
- The contract creator (and only the contract creator) is allowed to mint an arbitrary number of tokens to any user. The function should have the following signature: `mint(address to, uint256 amount)`
- The number of decimals is fixed to 18 (this is returned by the `decimals` function but not used in the contract itself)

Note that the different error messages that should be used when reverting can be found in the test files of the skeleton provided.

2 Implementing an ERC-721 token

ERC-721 is a token standard used to implement a non-fungible token (NFT). NFTs are often used to represent digital assets, where each token represents a different asset. For example, in the case of a collection of images, each token would represent a single image in the collection.

In this tutorial, the goal is to implement the following functions defined by the ERC-721 standard:

`balanceOf` Retrieve the number of tokens that a user owns

`ownerOf` Retrieve the owner of a given token

`transferFrom` Transfer a token to another address

`approve` Approve another user to transfer a given token on the user's behalf

`getApproved` Retrieve the user approved to transfer a given token

Like the ERC-20 standard, the ERC-721 standard also has some extensions. In this tutorial, we will also implement the metadata extension that defines the following other functions:

`name` The name of the token

`symbol` The symbol of the token

`tokenURI` The URI of the given token. The URI typically points to a JSON file containing metadata about the token

More information can be found in the ERC-721 standard.

In addition to the ERC-721 standard, our token will have the following extra specs:

- The constructor takes 5 arguments:
  1. The name of the token (type: `string memory`)
  2. The symbol of the token (type: `string memory`).
  3. The base URI of the token (type: `string memory`). Typically, the base URI is an URL such as `https://example.com/nft/`
  4. The ERC-20 token used pay for the ERC-721 token (type: `IERC20`)

5. The initial price (in terms of ERC-20 token) of a token (type: `uint256`)

- To form the `tokenURI`, the base URI (passed in the constructor) should be concatenated with the token ID (using `string.concat`). A function to convert an integer to a string is provided in the `src/libraries/StringUtils.sol` of the skeleton.
- No tokens exist initially
- A new token can be minted by anyone by "burning" the amount of ERC-20 tokens corresponding to the current price. The price starts at the initial price passed in the constructor. To burn the ERC-20 tokens, this contract transfers the ERC-20 tokens from the user and sends them to the zero address. Note that this will revert if the user has not approved this contract to spend his ERC-20 tokens beforehand.
- The first token has an id of 1 and the id is incremented each time a token is minted
- Each time a token is minted, the price increases by 10%

3 Bonus

In the bonus section, the goal is to finish the implementation of the ERC-721 token so that it is completely compliant with the standard. To do so, the following extra functionalities should be implemented:

- The ERC-721 standard specifies that the ERC-165 standard must be implemented. This standard stipulates that a contract must have a function `supportsInterface(bytes4 interfaceID)` that returns true if the given `interfaceID` is supported. For this tutorial `supportsInterface` should return true if and only if the `interfaceID` equals `0x80ac58cd`, which is the interface ID of the ERC-721 interface.
- The ERC-721 standard also allows accounts to have control over all the tokens of a user. This is handled by the two following functions

  `setApprovalForAll` Approve another user to manage (transfer or approve) all tokens owned by the user

  `isApprovedForAll` Retrieve whether a user is approved to manage all tokens owned by another user
- To avoid losing tokens, the ERC-721 has a function named `safeTransferFrom`. It does the same thing as `transferFrom` but performs some extra checks when transferring tokens to a contract address.