Cairo Crash Course

Components - Exercise 2

In this exercise, you will implement your first component and test it using a mock contract

Contract Implementation

In this exercise, you'll be working across three distinct files:

`components_exercise_2.cairo` - The component that you will implement

`mock_contract_components_2.cairo` - The mock contract for integrating the component and testing it

`test_components_2.cairo` - Testing the component

In the `components_exercise_2.cairo` file:

1. Declare a component with the name `OperationsComponent`
2. Declare and define the interface of the component, and all it's external functions:

```
fn get_value_from_storage(self: @TContractState) -> u32;
fn increment(ref self: TContractState);
fn set_to_zero(ref self: TContractState);
fn simple_addition(ref self: TContractState, x: u32, y: u32) -> u32;
fn simple_multiplication(ref self: TContractState, to_multiply: (u32,
u32)) -> u32;
```

3. Declare the storage of the components with one state variable, `value: u32`.

4. Implement the component external functions:

    1. `get_value_from_storage` -> Gets the `value` form the storage and returns it.
    2. `increment` -> Increments the `value` in storage by 1.
    3. `set_to_zero` -> Sets the `value` in storage to 0.
    4. `simple_addition` -> Adds two `u32` integers, writes the result to `value` in storage, and returns it.
    5. `simple_multiplication`-> Receives a tuple of two `u32` integers, multiplies them by each other, and returns the result.

Note: This component doesn't have events.

In the `mock_contract_components_2.cairo` file:

Integrate the component `OperationsComponent` into the mock contract

In the `test_components_2.cairo` file:

1. Declare and deploy the Mock contract
2. Create a dispatcher of the Mock contract
3. Using the dispatcher trigger the following functions and make sure the state is correct:
    1. Set `value` to `0`
    2. Retrieve and assert that value in `Storage` is `0`
    3. Call `simple_addition` with values `6` and `5`
    4. Retrieve and assert that value in `Storage` is `11`
    5. Invoke `simple_multiplication` with tuple of `6` and `5`
    6. Increment `value` by `1`
    7. Retrieve and assert the correspondent value in `Storage`
    8. The final value should be 12

Check that the exercise is completed by running the command `snforge test components_2`

Useful links

Components

Contracts