2.3.4.8.1 Characteristic Data

Return to Object Data Taxonomy

Overview

Return to Top

Every Object has characteristics that contextually describe the object. For example, the Object's name, if it implements and interfaces or if it is an extension of another object.

- **Name** is a textual name that uniquely identifies the object with the context of the Namespace. For example, there are multiple objects with the name **Tank**. The first is in the namespace (i.e., context) of the oil and gas industry and refers to a fuel tank. Another **Tank** object has a military namespace and refers to a military vehicle, and another **Tank** object has a clothing namespace and refers to a “tank top” tee shirt. All the rest of the data for the object is dependent on context (i.e., namespace).
- **Implements** is a way to describe which Interfaces this object provides an implementation for. An example is the ERC-20 interface used for many Cryptocurrencies. Java defines standardized collection interfaces for things like List, Set, Queue, Maps and Iterators[1)]. C++ has similar interfaces to those described for Java [2)]. The exact way interfaces are implemented differs from language to language, and can even differ from versions of the same language.
- **Extends** is a way to describe a hierarchical tree for an object (i.e., its Inheritance). For example, a graphics program may have a base object called a **`GeographicShape`**'. Other objects within the graphics program such as **`Square`**, **`Circle`** and **`Triangle`** are said to **extend** the original definition of **`GeographicShape`**.

DIDO Specifics

Return to Top

Smart Contracts are akin to classes in Object-Oriented Programming (OOP) languages; they can have Inheritance and Interfaces. There are two very well-known examples of **interfaces** in use within Ethereum:

- EIP 20: ERC-20 Token Standard
- EIP 721: ERC-721 Non-Fungible Token Standard

However, many of the Ethereum: Ethereum Improvement Proposals (EIPs) define interfaces.

Interfaces are similar to **abstract contracts**, but they are limited to what the contract’s ABI can represent. In other words, you could convert an ABI into an interface, or vice versa, and no information would be lost. According to the Solidity docs they have a few additional restrictions. For example, Doug

Crescenzi, 13 June 2018, Accessed 3 November 2022, https://medium.com/upstate-interactive/solidity-how-to-know-when-to-use-abstract-contracts-vs-interfaces-874cab860c56)) outlines the following restrictions:

- *Interfaces cannot have any functions implemented*
- *Interfaces cannot inherit other contracts or interfaces (contracts can however inherit interfaces just as they would inherit other contracts)*
- *Interfaces cannot define a constructor*
- *Interfaces cannot define variables*
- *Interfaces cannot define structs*
- *Interfaces cannot define enums*

Interfaces are expressed using the interface keyword. Here's an example:

```
pragma solidity ^0.4.24;

interface token
  { function totalSupply() public view returns (uint256);
    function balanceOf(address who) public view returns (uint256);
    function transfer(address to, uint256 value) public returns (bool);
  } // End Token
```

☒ [char][✓ char, 2022-03-22]~~Review subsections~~

1)
Programiz.com Java Collection Frameworks, Accessed: 27 October 2021, https://www.programiz.com/java-programming/collections
2)
Geeksforgeeks.com, The C++ Standard Template Library (STL), Accessed: Accessed: 27 October 2021, https://www.geeksforgeeks.org/the-c-standard-template-library-stl/