DRAFT: Clarity traits

Part 6

Phillip G. Bradford*

November 29, 2024

Abstract

Clarity is a functional language rather than an *Object Oriented (OO)* language. OO languages have polymorphism and inheritance. These two characteristics of OO languages can cloak things. This can add a lot of value to handle complex systems. Although polymorphism and inheritance may also contracts confusing.

Clarity is not OO, but functional. Clairty implements traits which is structured functional composition, rather than inheritance.

1 Introduction

There are several smart contract languages that are primarily *object oriented (OO)*. The OO paradigm is attractive since inheritance allows for common contract structure. However, OO languages also have polymorphism. Polymorphism and inheritance can cloak behaviors. This can add a lot of value to handle complex systems, though it may also make it very confusing for a contract.

Clarity is not OO, but functional. Clairity has traits to give structure to contracts. This is based on function composition. This gives common behaviors for different contracts.

Listing 1: A general trait from the Clarity-of-Mind book

```
(define-trait my-trait
    (
        (function-name-1 (param-types-1) response-type-1)
        (function-name-2 (param-types-2) response-type-2)
        ;; And so on...
    )
)
```

*phillip.bradford@uconn.edu, phillip.g.bradford@gmail.com, University of Connecticut, School of Computing, Stamford, CT USA

Listing 2: A trait we can use

```
(define-trait locked-wallet-trait
    (
        (lock (principal uint uint) (response bool uint))
        (bestow (principal) (response bool uint))
        (claim () (response bool uint))
    )
)
```

2 Basic traits

A simple trait from the book *Clarity of mind* is,

Listing 3: Simple multiply trait in contract c2.clar

```
(define-trait multiplier
    (
        (multiply (uint uint) (response uint uint))
                (add (uint uint) (response uint uint))
    )
)
```

where `(response <success type> <failure type>)`.

Listing 4: Applicaiton of the multiply trait in contract c1.clar

```
(impl-trait  .c2.multiplier)

(define-read-only (multiply (a uint) (b uint))
    (ok (* a b))
)

(define-read-only (add (a uint) (b uint))
    (ok (+ a b))
)
```

Listing 4 shows a function conforming to the trait in Listing 3.
Two steps: (1) importing a trait, then (2) implementing the trait:

```
(use-trait ExampleTrait .trait-example) ;; Import the trait

(impl-trait .ExampleTrait) ;; Declare the implementation
```

It is most interesting when several functions are implemented in traits.

Listing 5: Simple transfer trait

```
(define-trait simple-transfer-trait
  (
    ;; Transfer from the sender to a new principal
    (transfer (uint principal principal) (response bool uint))
  )
)
;;
;;(impl-trait
   'SP2PABAF9FTAJYNFZH93XENAJ8FVY99RRM50D2JG9.nft-trait.nft-trait)
;;
```

3 A cookbook example

The text-book example is from the Clarity Cookbook [1].

Listing 6: Fungible trait from [1]

```
(define-trait nft-trait
  (
    ;; Last token ID, limited to uint range
    (get-last-token-id () (response uint uint))

    ;; URI for metadata associated with the token
    (get-token-uri (uint) (response (optional (string-ascii
       256)) uint))

     ;; Owner of a given token identifier
    (get-owner (uint) (response (optional principal) uint))

    ;; Transfer from the sender to a new principal
    (transfer (uint principal principal) (response bool uint))
  )
)
;;
;;(impl-trait
   'SP2PABAF9FTAJYNFZH93XENAJ8FVY99RRM50D2JG9.nft-trait.nft-trait)
;;
```

Listing 7: Implementing fungible trait

```
(impl-trait
   'SP3FBR2AGK5H9QBDH3EEN6DF8EK8JY7RX8QJ5SVTE.sip-010-trait-ft-standard.sip-01

(define-constant contract-owner tx-sender)
```

```
(define-constant err-owner-only (err u100))
(define-constant err-not-token-owner (err u101))

(define-fungible-token clarity-coin)

(define-public (transfer (amount uint)
                         (sender principal)
                         (recipient principal)
                                                 (memo
                                                        (optional
                                                           (buff
                                                           34))))
    (begin
        (asserts (is-eq tx-sender sender) err-not-token-owner)(try
                      (ft-transfer? clarity-coin amount sender
                         recipient))
        (match memo to-print (print to-print) 0x)
        (ok true)
    )
)

(define-read-only (get-name)
    (ok "Clarity␣Coin"))

(define-read-only (get-symbol)
    (ok "CC"))

(define-read-only (get-decimals)
    (ok u6))

(define-read-only (get-balance (who principal))
    (ok (ft-get-balance clarity-coin who)))

(define-read-only (get-total-supply)
    (ok (ft-get-supply clarity-coin)))


(define-read-only (get-token-uri)
    (ok none))


(define-public (mint (amount uint) (recipient principal))
    (begin
        (asserts(is-eq tx-sender contract-owner) err-owner-only)(ft-mint?
           clarity-coin amount recipient)))
```

4 Exercises

1. How are traits different than Java Inerfaces?
2. How are traits different than OO abstract classes?

References

[1] https://docs.stacks.co/docs/cookbook/creating-an-ft

[2] Clarity Book, https://book.clarity-lang.org/ 2023-04-11.

[3] Kenny Rogers: *Building an NFT with Stacks and Clarity*, https://blog.developerdao.com/building-an-nft-with-stacks-and-clarity, 2022-09-01.

[4] Kenny Roger, Joe Bender: Stacks developer workshop: *Web3 for Bitcoin: The What, Why, and How of Building on Stacks*. Web3 for Bitcoin. Wed, Jun 29, 2022.