Learning Object-Oriented Programming, Design and TDD with Pharo

Stéphane Ducasse

March 11, 2019

Layout and typography based on the sbabook LaTeX class by Damien Pollet.

Contents

Illustrations

CHAPTER 1

Electronic wallet solution

Here are the possible solutions of the implementation we asked for the Wallet Chapter **??**.

1.1 Using a bag for a wallet

```
Wallet >> add: anInteger coinsOfValue: aCoin
  "Add to the receiver, anInteger times a coin of value aNumber"

  bagCoins add: aCoin withOccurrences: anInteger
```

We can add elements one by one to a bag using the message `add:` or specifying the number of occurences of the element using the message `add:withOccurrences:`.

```
Wallet >> coinsOfValue: aNumber

  ^ bagCoins occurrencesOf: aNumber
```

1.2 Testing money

```
Wallet >> money
  "Return the value of the receiver by summing its constituents"
  | money |
  money := 0.
  bagCoins doWithOccurrences:
      [ :elem : occurrence |
        money := money + ( elem * occurrence ) ].
  ^ money
```

1.3 Checking to pay an amount

```
Wallet >> canPay: amounOfMoney
  "returns true when we can pay the amount of money"
  ^ self money >= amounOfMoney
```

1.4 Biggest coin

```
Wallet >> biggest
  "Returns the biggest coin with a value below anAmount. For
    example, {(3 -> 0.5) . (3 -> 0.2) . (5-> 0.1)} biggest -> 0.5"

  ^ bagCoins sortedElements last key
```

1.5 Biggest below a value

```
Wallet >> biggestBelow: anAmount
  "Returns the biggest coin with a value below anAmount. For
    example, {(3 -> 0.5) . (3 -> 0.2) . (5-> 0.1)} biggestBelow:
    0.40 -> 0.2"

  bagCoins doWithOccurrences: [ :elem :occurrences |
      anAmount > elem ifTrue: [ ^ elem ] ].
  ^ 0
```

1.6 Improving the API

```
Wallet >> addCoin: aNumber
  "Add to the receiver a coin of value aNumber"

  bagCoins add: aNumber withOccurrences: 1
```

```
Wallet >> removeCoin: aNumber
  "Remove from the receiver a coin of value aNumber"

  bagCoins remove: aNumber ifAbsent: []
```

1.7 Coins for paying: First version

```
Wallet >> coinsFor: aValue
  "Returns a wallet with the largest coins to pay a certain amount
    and an empty wallet if this is not possible"
  | res |
  res := self class new.
```

```
  ^ (self canPay: aValue)
    ifFalse: [ res ]
    ifTrue: [ self coinsFor: aValue into2: res ]
```

```
Wallet >> coinsFor: aValue into2: accuWallet
  | accu |
  [ accu := accuWallet money.
  accu < aValue ]
    whileTrue: [
      | big |
      big := self biggest.
      [ big > ((aValue - accu) roundUpTo: 0.1) ]
        whileTrue: [ big := self biggestBelow: big ].
      self removeCoin: big.
      accuWallet addCoin: big ].
  ^ accuWallet
```

Bibliography