CIS*3260 F20 – Assignment 1

Purpose

- Implement classes and methods in Ruby from a given (simple) type/interface design provided.
- The system being designed models a player with a cup full of dice/coins. The cup is thrown and the results recorded.

Main idea behind the system to be built

- Each "player" has a bag and a cup.
- Dice and coins are created in a factory and placed in a player's bag.
- Dice/coins can be removed from a bag (using a "selection description" described below) and placed in a cup.
- When moving dice/coins they are placed in an object called a clutch that functions as a stack/queue etc. (i.e. is iterable). The implementation is your choice.
- Dice/coins are randomized by "throwing" the cup.
    - This returns a "throw" object which initially holds all the dice/coins. The cup becomes empty.
- The items in the throw should be emptied back into the player's bag, but a record of the contents of the throw must be kept in the throw object for reporting purposes.
- Each player stores the result of each throw, and the player can be asked for a "report".

The classes and methods to be implemented in Ruby

Randomizer Classes

RandomizerFactory
- create_die(sides:Int, colour:Enum)
- create_coin(denomination:Enum)

Randomizer
- randomize() *both randomizes and returns the randomizer itself (for method chaining)*
- result() *returns the result of the randomization, or nil if never randomized*
- randomize_count() *returns the number of randomizations performed*
- reset() *sets the result to nil and number of randomizations performed to 0*

Coin *<inherits from Randomizer>*
- denomination() *returns the denomination of the coin (does not set it)*
- flip() *flips the coin and returns the "flipped" coin (for method chaining)*
- sideup() *returns :H or :T (the result of the last flip) or nil (if no flips yet done)*

Die *<inherits from Randomizer>*
- colour() *returns the colour of the die (does not set it)*
- sides() *returns the number of sides (does not set it)*
- roll() *randomizes and returns the "rolled" die (for method chaining)*
- sideup() *returns 1..sides or nil*

RandomizerContainer Classes

RandomizerContainer

- store(r:Randomizer) *stores a randomizer in the container*
- add(rc:RandomizerContainer) *gets each randomizer in rc & stores It in the new container*
- ~~add(*objects)~~ ~~*a series of coins or dice, separated by commas*~~
- empty() *removes all members of the container*

Bag *<Inherits from RandomizerContainer>*

when store() or add() invoked, Bag makes sure that all randomizers added are reset

- select(description:Hash, amt=:all) *selects items from Bag based on the description and returns a Clutch object that is holding the items up to the number entered into amount*
- empty() *returns a Clutch of all items from the Bag*

Clutch *<Inherits from RandomizerContainer>*

- next() *removes and returns the last objected added if no objects stored, return nil*
- empty() *returns nil (items are "spilled on the ground") i.e. the pointers to the contained objects are lost (and the objects will be garbage collected by the system)*

Cup *<Inherits from RandomizerContainer>*

- throw() *each item in the cup is rolled or flipped, all items are removed and stored in a Throw object, the newly created Throw object is returned*
- load(c:Clutch) *enters each randomizer from a clutch (synonym of add())*
- empty() *returns a Clutch object to be returned to the bag, leaves the cup empty*

High Level Classes

Throw

- return_items() *returns the items in the Throw as a Clutch a record is made of the items so tally() and sum() continue to give the same answers to the same input description before and after return is called*
- tally(description:Hash) *counts the items in the Throw that match the description and returns the value*
- sum(description:Hash) *totals the value of the randomizer items in the Throw that match the description, where the value equals the number that is "up" (for coins, :H = 1 and :T = 0), and returns the value*
- report() *returns the value from the last tally or sum method call*

Player

- name() — *returns the name of the player (does not set it)*
- store(item:Randomizer) — *stores the item in the player's bag*
- add(rc:RandomizerContainer) — *gets each item in rc and stores it in the player's bag*
- load(description:Hash) — *loads items from the player's bag to the player's cup based on the description*
- throw() — *throws the (previously loaded) cup, replaces the items in the cup to the bag, both returns the throw and stores it internally*
- clear() — *clears all stored throws*
- tally(description:Hash) — *calls tally(description) on each stored throw and returns the combined values as an array*
- sum(description:Hash) — *calls sum(description) on each stored throw and returns the combined values as an array*
- report() — *returns the values as an array from the last tally or sum method call*

Enumerations

- enum RandomizerItems = :coin, :die
- enum Denomination = 0.05, 0.10, 0.25, 1, 2
- enum Colour = :red, :green, :blue, :yellow, :black, :white
- enum CoinSide = :H, :T

Description examples

- {item: :die, sides: 4, colour: :yellow, up: 4}
- {item: :die, colour: :red}
- {item: :die, sides: 6}
- {item: :die}
- {item: :coin, denomination: 0.25 }
- {item: :coin, up: :H}
- {item: :coin}

Grading

- [6 pts] The ruby code i.e. how much functionality does your written code try to encapsulate
- Create and run tests
    - [6 pts] Create use cases to base your tests on
        - i.e. reverse engineer use cases based on the type design provided
    - [12 pts] Execute your test cases and report the results
        - i.e. for each test
            - state purpose of test,
            - state expected output
            - report actual output
            - report pass/fail
        - Your tests must test at least each method of each class
- [6 pts] Passes our tests
- [5 pts] OO Style
- [1 pt] Ruby Style

Submit in Courselink

- Each class is kept in its own file (className.rb)
- The testing script should be called tests.rb
- Provide a read.me on how to use the testing script
- Provide a file that has the results of your tests (see grading), called mytest.pdf or mytest.txt
- Archive all files together (e.g. as a .zip file)