

CS1301: Project 1

Martian Currency

This project is worth 3% of your final grade.

Goals

- Apply CS1 concepts (loops, if-else, arrays, inheritance) in a moderately open-ended project

Introduction

In this and the other projects, I will tell you what your completed program *should* do, but not how to do it. You will need to write your own code based on the requirements I give you. In this project I will, however, give you several hints on what you need to do; be aware that I won't do this for the final project.

You may ask your instructor or a TA for help, but they will not tell you what to write. You may not give, nor receive, help from other students on this project.

Project Overview

You will write a program that will examine a set of coins and tell the user what the value of those coins is. The problem is that these coins are from the planet Mars, who use a form of currency called the *rofl*. Unfortunately the Martians, being a highly artistic race, developed their system of currency with more emphasis on aesthetics than efficiency.

All Martian coins come in red, yellow, or blue denominations. In addition to their colors, the coins have the following properties:

- Triangular Coins can have a face value of 1, 2, or 3 rofls
- Circular Coins have no value, but they negate the sum of the triangular coins of the same color
- Square Coins have no value, but they double the sum of the triangular coins of the same color

Thus, for example, if I have the following:

- Red coins:
 - triangles: 2, 2, 3
 - circles: none
 - squares: 1
- Blue coins:
 - triangles: 3, 1
 - circles: 1
 - squares: 2

- Yellow coins:
 - triangles 3, 3, 2, 1
 - circles: 2
 - squares: none

Then my total wealth is $(2+2+3)*2 = 14$ red, $(3+1)*(-1)*2*2 = -16$ blue, and $(3+3+2+1)*(-1)*(-1) = 9$ yellow for a total of $14+(-16)+9 = 7$ rofls. Note that both the $x2$ and $x(-1)$ modifiers are cumulative. In particular, if I have an odd number of (-1) modifiers, the sum for that color will be negative; if I have an even number, the sum will be positive.

Note also that it is possible to have negative wealth! (This strange occurrence has led to odd business practices where the seller actually pays the buyer to buy his goods! No wonder all the Earth economists went insane when we first made contact with the Martians.)

Project Requirements

Your project will contain the following classes:

- *Coin*
 - is abstract
 - has the protected instance variables
 - `int value` (can be 0, 1, 2, or 3)
 - `char color` (can be 'r', 'y', or 'b', for red, yellow, or blue, respectively)
 - has the non-abstract constructor `Coin(int value, char color)` which sets the corresponding instance variables
 - has the methods:
 - `abstract int getValue()`
 - `abstract int getModifier()`
 - `abstract char getColor()`
- *TriangularCoin*
 - extends *Coin*
 - implements *Coin*'s methods in the appropriate way
 - has a constructor with a parameter to take in the value (1, 2, or 3) of the coin and another parameter to take in the color ('r', 'y', or 'b'). It passes these values, along with the appropriate modifier and color, to the superclass constructor
- *SquareCoin*
 - extends *Coin*
 - implements *Coin*'s methods in the appropriate way
 - has a constructor with a single parameter to set the color; this constructor calls the superclass constructor with the appropriate values
- *CircularCoin*
 - extends *Coin*

- implements *Coin*'s methods in the appropriate way
- has a constructor with a single parameter to set the color; this constructor calls the superclass constructor with the appropriate values
- *MartianAddingMachine*
 - contains a single method, named `tally`, that takes in an array of *Coin* objects and sums their value, returning an `int`

You will write a test class for *MartianAddingMachine* that will test the `tally` method. It should handle the following cases:

- If the array of *Coins* is empty (ie, null), tally should return zero
- combination of red, blue, and yellow coins
- combination of positive and negative modifiers

You should properly document the tally method with a Javadoc comment. Be sure to include all parameters and return values, as well as pre-conditions and post-conditions.

It is recommended (but not required) that you write the Javadoc comments and the test class **before** you write the `tally` method. This will aid you in thinking about how `tally` should work.