HW1: DSSL2 Warmup

Due: Thursday, October 11, at 11:59 PM, via GSC

You must work on your own for this assigment. Future assignments will allow partners, but not this one.

The purpose of this assignment is to get you programming fluently in DSSL2, the language that we'll be using for the course.

In `warmup.rkt`[1] I've supplied starter code including most of a class, with headers for the functions that you'll need to write, along with some tests.

Part I: Installing DSSL2

To complete this homework assignment, you will first need to install the DrRacket programming environment, version 7 (available from `racket-lang.org`). Then you will need to install the DSSL2 language within DrRacket.

Once you have DrRacket installed, open it and choose "Package Manager" from the "File" menu. Go to the "Do What I Mean" tab, and paste the package URL, `https://github.com/tov/dssl2.git`, into the text box. Then click the "Install" button and wait for installation to finish. When it's finished, the "Install" button should change to "Update"; then close the window.

Part II: Class practice

In `warmup.rkt` we define a class for representing bank accounts as follows:

```
#lang dssl2

# An AccountId is a Natural

# An Account is Account(AccountId, String, Number)
class Account:
    let id
```

[1] `http://goo.gl/YBv9gS`

```
    let owner
    let balance

    # Account: AccountId String Number -> Account
    # Constructs an account with the given ID number,
    # owner name, and balance. The balance cannot be
    # negative.
    def __init__(self, id, owner, balance):
        if balance < 0: error('Account: negative balance')
        self.id = id
        self.owner = owner
        self.balance = balance

    # ... additional methods omitted ...

# Examples:
let ACCOUNT0 = Account(0, "Alan Turing",    16384)
let ACCOUNT1 = Account(1, "Grace Hopper",   32768)
let ACCOUNT2 = Account(2, "Ada Lovelace",   32)
let ACCOUNT3 = Account(3, "David Parnas",   2048)
let ACCOUNT4 = Account(4, "Barbara Liskov", 8192)
```

The `Account` class has several methods, including:

- `Account.clone`, which copies an `Account` object,
- getters `Account.get_balance`, `Account.get_id`, and `Account.get_owner`, and
- `Account.deposit`, which adds an amount to the balance of the account.

It contains headers for two additional methods, which you must write:

1. `Account.withdraw: Number -> Void`, which subtracts the given amount from the balance if the balance would not go negative. If the requested withdrawal exceeds the balance, this method must call `error`.
2. `Account.__eq__: Account -> Boolean` compares two accounts for equality by comparing their three fields. Two accounts are equal if all three fields are equal, pairwise.

Note that the `__eq__` method implements the `==` operator for its class. That

is, comparing two `Account`s with `==` calls your `__eq__` method, which must be defined to compare objects of its class appropriately.

Then, there is one free (non-method) function to write for dealing with accounts:

3. `account_transfer : Number Account Account -> Void` withdraws the given number from the first account's balance and deposits it in the second.

You may want to write additional tests for the above methods and function. Note that the provided tests for `account_transfer` and `Account.withdraw` clone the accounts that it they modifying before modifying them—this is so that subsequent tests can also rely on the original values of the accounts.

Part III: Vector practice

Write these four functions:

4. `vec_swap : VecC[X] Natural Natural -> Void` takes a vector and two indices, and swaps the vector's values at the indices.
5. `vec_copy : VecC[X] -> VecC[X]` copies a vector. That is, it makes a new vector of the same length containing the same elements. (Use a vector comprehension for the easiest way to do this and the next one.)
6. `vec_copy_resize : Natural VecC[X] -> VecC[X]` copies and resizes a vector. In particular, it takes as its first parameter the length for the new vector to create and then copies over as many elements as will fit. If the new vector is shorter than the old then it won't contain all of the old vector's elements. If the new vector is longer then the remaining elements should be filled with `False`.
7. `find_largest_account : VecC[Account] -> Account` takes a non-empty vector of `Account`s and returns the account with the largest balance. You may assume that the vector has at least one element, and you don't need to worry about ties for the largest balance.

Deliverable

The provided file `warmup.rkt`, containing definitions of the two methods and five functions described above, and sufficient tests to be confident of your code's correctness. You will be graded for correctness, efficiency, style, and adequacy of tests.

Submission

Your homework must be submitted via the online system GSC, which can be found at `https://eecs214.cs.northwestern.edu/`.

Before you can submit, you will have to sign up for an account. **Your user name must be your NetID,** which is three letters followed by three or four digits. If you use anything else as your username, we will not know who gets credit for your work. Please choose a secure password and do not share it.

When logging into GSC, you will usually want to check the "Keeps login for 2 weeks" box, as otherwise refreshing the page will lose your session.

Upon signing into GSC, you should see a list of assignments with due date. Select the assignment you want to submit and upload the file(s). You do not need to do anything to indicate that your submission is complete—whatever you have uploaded as of the due date will be considered your submission.