Assignment 2: Dating with Ruby

Due: 19:00, Fri 18 Mar 2016 Full marks: 100

1 Introduction

In this assignment, you will gain experience in *Ruby*, which is a pure object-oriented language. The assignment consists of three parts. Task 1 is to implement a game called *Gomoku* (五子棋) in Ruby with some simple artificial intelligence. Task 2 is to rewrite a given Ruby program (a Hospital Administration System) in either Java or C++ of your choice. Task 3 is to write a one-page report to tell us your experience in using Ruby. The aim of the assignment is to let you understand and appreciate the flexibility of *dynamical typing* and *duck typing*.

2 Task 1: Gomoku

In this task, you will write a program in Ruby for the game *Gomoku*. The following subsections describe the game rules and the object-oriented design.

2.1 Game Description

The game is played on a game board with 15 rows and 15 columns. Two players O and X take turns to put their game discs into one unoccupied square of the board. The player who first forms a line of five or more consecutive discs horizontally —, vertically |, or diagonally ╲╱ wins the game. ("Go" in Gomoku means "five" in Japanese.) The game is a draw when the board is full but no player wins. Figure 1 shows an example configuration of Gomoku. If Player X puts a disc to the square in row 7, column 6, then a diagonal line ／ of five consecutive X's will be formed and thus X will win.

```
                     1 1 1 1 1
   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4
 0 . . . . . . . . . . . . . . .
 1 . . . . . . . . . . . . . . .
 2 . . . . . . . . . . . . . . .
 3 . . . . . . . . . . . . . . .
 4 . . . . . . . . . . . . . . .
 5 . . . . . . . X X . . . . . .
 6 . . . . . . . X O . . . . . .
 7 . . . . . O . O O . . . . . .
 8 . . . . . X X O O . . . . . .
 9 . . . . X O O X X . . . . . .
10 . . . . . . . . . . . . . . .
11 . . . . . . . . . . . . . . .
12 . . . . . . . . . . . . . . .
13 . . . . . . . . . . . . . . .
14 . . . . . . . . . . . . . . .
```

Figure 1: An Example Game Configuration of Gomoku

2.2 Program Design and Specification

Your program file name should be `gomoku.rb`. You have to write your program in the classes `Gomoku`, `Player`, `Human`, and `Computer` defined below. *You are free to add other extra members*

(instances variables, methods, etc.) to these classes. *You are free to add extra classes* as well. But you *cannot have public instance variables in all classes*.

2.2.1 Class `Gomoku`

This class models the Gomoku game, with the following members:

Instance Variables

`@board`

A two dimensional array of size 15x15 representing the game board.

`@player1`, `@player2`

Player O and Player X respectively.

`@turn`

The player in the current turn. The first turn is Player O.

Instance methods

`initialize`

A constructor for initializing all the instance variables.

`startGame`

Starts a new Gomoku game and play until the end of game. The flow of this method is as follows:

1. Prompt the user to choose the player type (computer or human) for Player O and then Player X.
2. The game starts with an empty game board. Player O takes the first turn.
3. Obtain a position from the current player for putting the disc to.
4. Update the board by putting a disc to the appropriate square.
5. Determine if the current player has connected five or above.
6. Repeat Steps 3– 5 until the game is finished. Alternate Players O and X in each round.
7. Once the game is finished, display the message “Player O wins!”, “Player X wins!”, or “Draw game!” accordingly.

`printBoard`

Prints out the game board in the console, in the format shown in Figure 1.

2.2.2 Class `Player`

This class is an abstract superclass for modeling players of the Gomoku game.

Instance Variable

`@symbol`

The symbol of the player. It is either “O” or “X”.

Instance Methods

`initialize(symbol [, …])`

A constructor for initializing the player symbol as the parameter `symbol`. You can design extra parameters for this constructor if necessary.

`nextMove`

An abstract method *to be implemented in subclasses*. In all subclass implementations, this method should return an array (that is, a list) of length two, in which the first and second elements (indices 0

and 1 respectively) are the row and column numbers of the next move respectively. For example, suppose the next move by the player is row 6, column 8, then calling the method should return the array `[6, 8]`. The position returned by this method must be a *valid* move (that is, row and column numbers within 0–14 and square is unoccupied).

2.2.3 Class Human

The Human class models a human Gomoku player and is a *subclass of* `Player`.

Instance Method

`nextMove`

This method obtains the next move of a human player from user input. The flow of the method is as follows:

1. Prompt the user to enter a row number and a column number for placing a disc. You can assume that *the user input is always an integer, followed by a space, followed by another integer*.
2. In case the player makes an invalid input, display a warning message and go back to Step 1.
3. Return the input position as an array of length two (e.g., `[6, 8]` for row 6, column 8).

2.2.4 Class Computer

The `Computer` class models an AI Gomoku player and is a *subclass of* `Player`.

Instance Method

`nextMove`

This method determines the next move of a computer player using the following simple AI.

1. In the game board, if there is an unoccupied square where placing a disc there will result in the computer player winning the game (connecting five or above), then this position will be returned. If there is more than one such square, you can pick any one of them to return.
2. If there is no such square, then this method returns a randomly generated position. Note that *the generated position must be a valid move*.

2.2.5 "Main" Program

The whole program should be started using the statement:

```
Gomoku.new.startGame
```

Some sample program outputs are provided in Blackboard for your reference.

3 Task 2: Hospital Administration System

This task is to implement a Hospital Administration System (HAS). A Ruby program of the system is given. You have to understand its behavior and re-implement it in either Java or C++ of your choice. After finishing this task, you should have experienced a special feature of Ruby called *duck typing*. Since Java and C++ do not directly support duck typing, you will also see that simple things in Ruby will sometimes become more complicated things in Java/C++.

3.1 Duck Typing

The following synopsis of duck typing is summarized from:

- http://en.wikipedia.org/wiki/Duck_typing
- http://www.sitepoint.com/making-ruby-quack-why-we-love-duck-typing

In standard statically-typed object-oriented languages, objects' classes (which determine an object characteristics and behavior) are essentially interpreted as the objects' types. Duck typing is a new typing paradigm in dynamically-typed languages that allows us to dissociate typing from objects' characteristics. It can be summarized by the following motto by the late poet James Whitcombe Riley:

When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck.

The basic premise of duck typing is simple. If an entity looks, walks, and quacks like a duck, for all intents and purposes it is fair to assume that one is dealing with a member of the species *anas platyrhynchos*. In practical Ruby terms, this means that it is possible to try calling any method on any object, regardless of its class.

An important advantage of duck typing is that we can enjoy *polymorphism without inheritance*. An immediate consequence is that we can write more generic codes, which are cleaner and more precise.

3.2 Task Description

The Hospital Administration System in Ruby is provided in Blackboard. It involves various units of a hospital such as medical departments, pharmacy, canteens, etc., and also users such as doctors, patients, and visitors. Different users have different characteristics and behaviors. For examples, doctors receive salaries from the hospital but patients do not; both doctors and patients can see a doctor (doctors can be sick too!) but visitors cannot; all doctors, patients, and visitors can buy food in the canteen; and so on. You have to study the provided Ruby program to understand all these characteristics and behaviors.

You have to replicate all the behaviors of the Ruby program using either Java or C++ of your choice. We will evaluate your program not just on execution correctness but also program design. Specific requirements for Java and C++ are listed below.

3.2.1 Java

Your program will be graded in Java SE 8. You cannot use the “reflection” and “generics” features in your program. The main class of the system should be named HAS (that is, we can run your system with the command “`java HAS`”.)

3.2.2 C++

Your program will be graded in Visual Studio 2013+. You cannot use the “template” feature in your program. The main function of your system should be written in the source file `has.cpp`.

4 Task 3: Written Report

Your written report (`report.pdf`) should answer the following questions within one A4 page (*all questions in one page total*).

1. Using your codes for Task 2, compare polymorphism obtained with duck typing and with inheritance.
2. Using your codes for Task 2, give a scenario where the Ruby implementation is better than the Java/C++ implementation. Briefly state your reason.

5 Submission and Marking

- Prepare a zip file `assg2.zip` containing the following files: (a) `gomoku.rb`, (b) all `.java` files (for Java) or `.cpp` and `.h` files (for C++), and (c) `report.pdf`.
- Submit the file `assg2.zip` in Blackboard (https://elearn.cuhk.edu.hk/). Besides, your *report* has to be submitted to VeriGuide (http://www.veriguide.org/) as well.
- You can submit your assignment multiple times. Only the latest submission counts.
- *Plagiarism is strictly monitored and heavily punished if proven.* Lending your work to others is subjected to the same penalty as the copier. You have to insert the following statement as comments in your Ruby and Java/C++ programs.

CSCI3180 Principles of Programming Languages
--- Declaration ---
I declare that the assignment here submitted is original except for source material explicitly acknowledged. I also acknowledge that I am aware of University policy and regulations on honesty in academic work, and of the disciplinary guidelines and procedures applicable to breaches of such policy and regulations, as contained in the website http://www.cuhk.edu.hk/policy/academichonesty/
Assignment 2
Name:
Student ID:
Email Addr: