CS51 **Spring 2024**

Code Review 8: Object-Oriented Programming

1 Practice Problems

Problem 1.1 (Object-Oriented Counters).

```
class loud_counter : counter_interface =
  object (this)
     inherit counter as super
     method! bump n =
     super#bump n;
     Printf.printf "State is now %d\n" this#get_state
  end ;;


class type reset_counter_interface =
    object
        inherit counter_interface
        method reset : unit
    end ;;

class loud_reset_counter : reset_counter_interface =
    object (this)
        inherit loud_counter
        method reset =
        this#bump (-this#get_state)
    end ;;
```

Problem 1.2 (Social Network). "ConnectU" is a social media site for college students. Suppose we want to define a `user` class.

1. Define a class interface for a user `user` that includes the following methods:
   (a) Get a user's username
   (b) Get a user's student id (a string)
   (c) Add a friend to a user's list of friends; note that friendships are mutual, so if Alice is a friend of Bob, then Bob is a friend of Alice.
   (d) Get a user's friend list
   (e) Add a post (a string) to a user's list of posts
   (f) Remove a post
   (g) Retrieve all the user's posts

2. Define an implementation that satisfies the class interface which takes in a `username` and an `id`.
3. Define a class interface called `student` which has the same functionality as the `user` class but an additional method which outputs the school a student attends.
4. Define an implementation that satisfies the `student` interface which takes in a `username`, `id`, and the `school` the student attends.
5. Define a function `form_friend_group` that takes in a list of `users`, and for each user in the list, sets their friends to be everyone else in the list.
6. Define a list of four `students` and use `form_friend_group` on the `student` list.

Solution

```
class type user_type =
 object

  method get_username : string

  method get_id : string

  method add_friend : user_type -> unit

  method get_friends : user_type list

  method add_post : string -> unit

  method remove_post : string -> unit

  method get_posts : string list

 end ;;

 class user (username : string) (id : string) : user_type =
  object (this)

    val mutable friends : user_type list = []
    val mutable posts : string list = []

    method get_username = username

    method get_id = id

    method add_friend (friend : user_type) : unit =
        if
          List.mem friend friends
        then
```

```
          ()
        else
          (friends <- friend :: friends;
          friend#add_friend (this :> user_type))

  method get_friends = friends

  method add_post post = posts <- post :: posts

  method remove_post post = posts <- List.filter ((<>) post) posts

  method get_posts = posts

 end ;;

class type student_type =
 object
 inherit user_type
 method get_school : string

 end ;;

class student (username : string) (id : string) (school : string) =
 object

 inherit user username id
 method get_school = school

 end ;;


let rec form_friend_group (users : user_type list) : unit =
  match users with
  | [] -> ()
  | head :: tail ->
      List.iter (fun user -> head#add_friend user) tail;
      form_friend_group tail ;;

let jayden = new student "jaydenp" "1" "Harvard" ;;
let gerson = new student "gersonp" "2" "Harvard" ;;
let kwee = new student "kwee" "3" "Harvard" ;;
let victoria = new student "victoria" "4" "Harvard" ;;

let user_list : student list = [jayden; gerson; kwee; victoria] ;;

form_friend_group (user_list :> user_type list) ;;
```

Problem 1.3 (Polynomial)**.** In this problem, you will define classes that work with (univariate) polynomials.

A polynomial is an expression consisting of coefficients and variables.

We define a class interface `polynomial_type`.

Listing 1: polynomial_type

```
1
2  class type polynomial_type =
3    object
4      method get_coeffcients : float list (* int list returns the
           coefficients of the polynomial in order from lowest degree to
           highest degree. For example, [5, 3, 2] would represent the
           polynomial 5 + 3x + 2x^2. *)
5
6      method evalulate : float -> float (* evalulates a polynomial f(x)
           *)
7
8
9      method solve : float -> float list option (* solve c returns real
           solutions for x when a polynomial equals c. For example if a
           polynomial is 5 + 3x + 2x^2, polynomial_object#solve 10 returns
           a list of solutions to 5 + 3x + 2x^2 = 10  *)
10   end
```

Now we define the class definition for a `polynomial class`.

Listing 2: polynomial

```
class polynomial (coefficients : float list) : polynomial_type =
 object
  method get_coefficients : float list = coefficients

  method evaluate (x : float) : float = failwith "not yet implemented"

  method solve (c : float) : float list option = None
 end ;;
```

1. Implement the `evaluate` method for the polynomial class. For example, for a polynomial $5 + 3x + 2x^2$ with coefficients $[5.0, 3.0, 2.0]$, the evaluate method called on 5 should return $5 + 3(5) + 2(5)^2 = 70$.

2. Implement a `linear_polynomial` class satisfying the `polynomial_type` class interface. A linear polynomial is an expression that has **at most two coefficients**. For example, $5x + 2$ (with coefficients as $[2.0, 5.0]$), $3x$ ($[0., 3.]$), and 11 ($[11.]$) are all polynomial functions. *Hint*: You may define a type so that users can only define at most two coefficients.

   For a constant polynomial c, `solve a` should return `[Float.infinity]` if $c = a$, `None` otherwise.

3. Implement a `quadratic_polynomial` class satisfying the `polynomial_type` class interface.

   Quadratic polynomials have at most three coefficients. To solve quadratic polynomials, we can use the quadratic formula.

   The solutions to a quadratic $ax^2 + bx + c = 0$ is given by the formula: $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$.

   Remember to properly handle cases when there are no real solutions or infinitely many solutions.

Solution

```
class type polynomial_type =
 object

  (* returns the coefficients of the polynomial in order from lowest
     degree to highest
     degree *)
  method get_coefficients : float list

  (* 'evaluate' x evalutes a polynomial f at x, i.e. computes f(x) *)
  method evaluate : float -> float
```

```
  (* solve c returns real solutions for x when a polynomial equals c.
      In other words,
     it should return the real solution solutions x such that f(x) =
       c *)
  method solve : float -> float list option

 end ;;

class polynomial (coefficients : float list) : polynomial_type =
 object
  method get_coefficients : float list = coefficients

  method evaluate (x : float) : float =
     List.fold_right (fun coeff acc -> x *. acc +. coeff)
        coefficients 0.0

  method solve (c : float) : float list option = None
 end ;;

type linear_coefficients =
 | Constant of float
 | Linear of float * float (* first element of constructor should be
    intercept, second should be slope *)

let rec extract_linear_coeffs (coeffs : linear_coefficients) =
  match coeffs with
  | Constant c -> [c]
  | Linear (intercept, slope) -> if slope = 0. then
     extract_linear_coeffs (Constant intercept) else [intercept;
     slope]

class linear_polynomial (coefficients : linear_coefficients) :
   polynomial_type =
 object (this)

 inherit polynomial (extract_linear_coeffs coefficients)

 method! solve (c : float) : float list option =
    let coeffs = this#get_coefficients in
     match coeffs with
     | [constant] -> if c = constant then Some [Float.infinity] else
        None
     | [intercept; slope] -> Some [(c -. intercept) /. slope]
     | _ -> failwith "solve: Invalid linear polynomial"

 end ;;
```

```
type quadratic_coeffs =
  | Constant of float
  | Linear of float * float (* first element is intercept, second is
      slope *)
  | Quadratic of float * float * float (* Quadratic (c, b, a)
      corresponds to ax^2 + bx + c *)

let rec extract_quad_coeffs (coeffs: quadratic_coeffs) =
   match coeffs with
  | Constant c -> [c]
  | Linear (intercept, slope) -> if slope = 0. then
      extract_quad_coeffs (Constant intercept) else [intercept; slope]
  | Quadratic (c, b, a) -> if a = 0. then extract_quad_coeffs (Linear
      (b, c)) else [c; b; a] ;;


class quadratic_polynomial (coefficients: quadratic_coeffs) :
   polynomial =
 object (this)
 inherit polynomial (extract_quad_coeffs coefficients)

 method !solve (c : float) : float list option =
    let coeffs = this#get_coefficients in
     match coeffs with
     | [constant] -> if c = constant then Some [Float.infinity] else
        None
     | [intercept; slope] -> Some [(c -. intercept) /. slope]
     | [c_prime; b; a] ->
        let inner = b ** 2. -. 4. *. a *. (c_prime -. c) in
        if
          inner < 0.
        then None
        else
          let root = sqrt inner in
          let left = (-.b -. root) /. (2. *. a) in
          let right = (-.b +. root) /. (2. *. a) in
          if left = right then Some [left]
          else Some [left; right]
     | _ -> failwith "solve: Invalid quadratic polynomial"
 end ;;
```