CSU2510H Exam 2 – Spring 2011

Name: _____

Student Id (last 4 digits): _____

- You may use the usual primitives and expression forms of any of the `class` languages or `Java`; for everything else, define it.
- You may write `c` $\rightarrow$ `e` for tests in both `class` **and Java** programs.
- To add a method to an existing class definition, you may write just the method and indicate the appropriate class name rather than re-write the entire class definition.
- We expect data *and* interface definitions (although interfaces may be defined in comments rather than with `define-interface`).
- If an interface is given to you, you do not need to repeat the contract and purpose statements in your implementations. Likewise, you do not need to repeat any test cases given to you, but you should add tests wherever appropriate.
- Some basic test taking advice: Before you start answering any problems, read *every* problem, so your brain can be thinking about the harder problems in background while you knock off the easy ones.

Problem	Points	/out of
1		/ 18
2		/ 19
3		/ 22
4		/ 20
5		/ 11
Total		/ 90

Good luck!

Problem 1 You've just been hired by Adobe to write their next generatation PDF rendering engine. As your first task, you've been assigned to write an *outline viewer* that displays the overall structure of a document. A document can consist of any number of sections, and a section has a title and any number of subsections, and a subsection has a title and any number of subsubsection, and so on.

18 POINTS

1. Design a representation of documents that can handle documents like this:

```
1. OO = struct + fun
1.1. Rocket
1.2. Moon
2. Design
2.1. Announce
2.1.1. Assign
3. End
```

   Note that the section numbers are *NOT* part of the document!

2. Using your design, give the representation of the document in part 1.

3. Design a method `outline` that produces a list of strings, which are the sectional headings in order *and with section numbers*.

   In other words, if `d` represents the document in part 1, `outline` should produce the following:

```
> (d . outline)
'("1. OO = struct + fun"
  "1.1. Rocket"
  "1.2. Moon"
  "2. Design"
  "2.1. Announce"
  "2.1.1. Assign"
  "3. End")
```

[Here is some more space for the previous problem.]

Problem 2 The Programming Research Lab has been acquiring some new decorative objects, in the form of rectangles and rectangular solids. Unfortunately, they've all come unpainted, and now the graduate students have to paint them. Dan has been trying to calculate how much this will cost, and he's enlisted you to help. 19 POINTS

First, some basic facts. Rectangles have an area which is the width times height, and must be painted on both sides. Rectangular solids have a surface area which is the sum of the areas of all of the sides, or $2 \cdot (w \cdot d + d \cdot h + w \cdot h)$. Red paint costs 2 dollars per square foot, and blue paint costs 5 dollars per square foot. Each decorative object has a color, and the appropriate dimensions, measured as an integer number of feet.

Design a representation for these decorative objects, and implement the `expense` method. You should use *overriding* to avoid repeating code—in particular, you should only have one version of the `expense` method.

[Here is some more space for the previous problem.]

Problem 3 Tobin-Hochstadt and Van Horn have both been traveling a lot recently, so they've decided to develop software to manage their trips. Since they're busy with writing the exam, they've asked you to help them out. 22 POINTS

Additionally, since they want their program to be as verbose as possible, they've asked you to write it in Java. In each of the following implementations, you must support the `howLong` method.

1. Initially, they've settled on the following kinds of data to represent trips. A `Trip` is either a `PlaneFlight`, with an airline and a distance (in miles), or a `Connection`, composed of two trips.

   Write a data definition for trips, and implement the `howLong` method, which computes the total time of the trip (in minutes), assuming that airplanes fly at 10 miles per minute.

2. Van Horn has discovered that getting around New England is easier if you take the train, even though it takes 2 minutes to go 1 mile. Add a new representation for this new kind of trip, which should have a distance and an indication of whether the trip is on the Acela, in which case the train goes twice as fast (1 minute per mile).

3. Tobin-Hochstadt also likes to take the train, but unfortunately, he's been taking the commuter rail instead, which goes as fast as regular trains but is always half an hour late. Add to your data definition of trains a representation for commuter rail trips.

[Here is some more space for the previous problem.]

[Here is some more space for the previous problem.]

Problem 4 Here are data, class, and interface definitions for Complex Numbers: 20 POINTS

```
;; A Complex is (cplx% Number Number)
;; dist : -> Number
;; How far is this point from the origin.
;; =? : Complex -> Boolean
;; Determine if this Complex is the same as the given one

(define-class cplx%
  (fields real imag)
  (define/public (dist)
    (sqrt (+ (sqr (real)) (sqr (imag)))))
  (define/public (=? that)
    (and (= (real) (that . real))
         (= (imag) (that . img)))))
```

1. Design a `UPair` data definition for an unordered pair of `Complex`, with an `=?` method. Because these are unordered pairs,

   ```
   ((upair% (cplx% 1 1) (cplx% 2 2)) . =?
    (upair% (cplx% 2 2) (cplx% 1 1)))
   ```

   should produce `#t`.

2. Revise your data, class, and interface definitions so that your unordered pairs can store other kinds of data, not just `Complex`es. Make your solution as general as possible, but no more.

3. Since `UPair` is unordered, switching the two components of a pair should produce a pair that's equal to the original one. Define a predicate that expresses this property, and use it to state two test cases.

Problem 5 Here's an implementation of lists that supports visitors: 11 POINTS

```
;; A [Listof X] is one of
;; - (mt%)
;; - (cons% X [Listof X])
;;
;; and implements:
;;
;; Accept the visitor and visit this list's data.
;; visit : [IListVisitor X Y] -> Y

(define-class mt%
  (define/public (visit v)
    (v . mt)))

(define-class cons%
  (fields first rest)
  (define/public (visit v)
    (v . cons (field first)
              ((field rest) . visit v))))
```

The `[IListVisitor X Y]` interface represents a computation over a `[Listof X]` that produces a `Y`:

```
;; An [IListVisitor X Y] implements:

;; Visit an empty list
;; mt : -> Y

;; Visit a cons list
;; cons : X Y -> Y
```

1. Design a class that implements `[IListVisitor Number Number]` that sums a list of numbers.

2. Recall our previous discussion of functions-as-objects, which relied on an interface for objects representing functions:

```
;; An [IFun X Y] implements:

;; Apply this function to given argument.
;; apply : X -> Y
```

Design a class `map%` that implements `[IListVisitor [Listof X] [Listof Y]]`.

For example, here is a use of the `map%` visitor that maps the representation of an add1 function over a list of numbers.

```
;; An Add1 is a (add1%), implements [IFun Number Number].
(define-class add1%
  (define/public (apply x) (add1 x)))

> ((cons% 1 (cons% 2 (cons% 3 (mt%)))) . visit (map% (add1%)))
(cons% 2 (cons% 3 (cons% 4 (mt%))))
```

[Here is some more space for the previous problem.]