Programming Languages Midterm 4

Winter 2011
Problems 1-8

1.
“Hello World
Message: You slime You done good You yellow slime”

2.
(a).
msg1: String msg, cur, deflt
msg2: String msg, cur, deflt
msg3: String msg, cur, deflt

(b).
No -- each new object instance creates its own table of private/protected instance variables from its class(es). The only way those objects could share would be by providing accessor methods

(c).
msg1 :
& String msg [from MeanMsg]
& String cur, deflt [from Msg]
& VMT of MeanMsg
msg2:
& String msg [from NiceMsg]
& String cur, deflt [from Msg]
& VMT of NiceMsg
msg3:
& String msg [from ReallyMeanMsg]
& String cur, deflt [from Msg]
& VMT of ReallyMeanMsg

VMT of MeanMsg, NiceMsg, and ReallyMeanMsg:
toString() [from Msg]

Wasn't very sure how to do this -- the book isn't very explicit about how to lay it out. I assumed that since the objects' fields, the Strings, are all instance variables, they would not lie in the VMTs but in the memory tables for the instances of the objects.

3.
MeanMsg, ReallyMeanMsg, and NiceMsg all have *is-a* relations to Msg. All four classes have *has-a* relations to String.

4.
(a).
Polymorphism (either parametric or subtype)
(b).
Subtype polymorphism in OO languages is checked at run time, while parametric polymorphism, such as generics in Java, is type checked during compile time.
(c).
Subtype:
List l = new ArrayList(); // new list of type Object
l.add(1);

Parametric:
List l<Int> = new ArrayList();
or in haskell:
length :: [a] -> Int
length [] = 0
length (l:ls) = 1 + (length ls)

(I couldn't see any distinction in the question in 4a between parametric and subtype polymorphism... did I miss something?)

5.
RTTI identifies the class of an object, while Reflection will provide different kinds of information about a given class.

6.
f = (\n . take(n,[1...])) where [1...] is an infinite list

In eager evaluation, the function would never complete, because we can never finish evaluating the infinite list. In lazy evaluation, we would receive a list of length n starting at 1.

7.
Yes. Monadic programming in Haskell using the state monad can be very imperative, yet there are no assignments to memory.

8.
(a).
Encapsulation is simply a grouping of data and control. Information hiding seems to be an aspect or a feature of encapsulation that provides the added feature of hiding certain data and control from outside the package/module.

(b).
They both involve the grouping of data and control. Information hiding could be described as a feature of encapsulation. When data/control is encapsulated and hides some of its inner workings/data then it has information hiding.

(c).

```
module MyMod (a) where
a n = n+1
b n = n+2
```

MyMod only exports a, hiding b, and it encapsulates both of a and b.