Programming Languages Midterm 4

Winter 2011
Problems 9 and 10

9.

(a).

```
data Msg = Msg { cur::String, deflt::String }
data MeanMsg = MeanMsg Msg String
data ReallyMeanMsg = ReallyMeanMsg Msg String
data NiceMsg = NiceMsg Msg String
-- (could label the fields for these data types also, to match the
Java labels)

class Message m where
    toString  :: m -> String
    construct :: m

instance Message MeanMsg where
    construct = MeanMsg (Msg " You slime" " Default Message") " You
slime"
    toString (MeanMsg _ m) = m

instance Message ReallyMeanMsg where
    construct = ReallyMeanMsg (Msg " You yellow slime" " Default
Message")
                " You yellow slime"
    toString (ReallyMeanMsg _ m) = m

instance Message NiceMsg where
    construct = NiceMsg (Msg " You done good" " Default Message")
                " You done good"
    toString (NiceMsg _ m) = m

main = do putStrLn "Hello World"
          let msg1 = toString (construct :: MeanMsg)
              msg2 = toString (construct :: ReallyMeanMsg)
              msg3 = toString (construct :: NiceMsg)
           in putStrLn ("Message: " ++ msg1 ++ msg2 ++ msg3)
```

(b).

I defined the abstract structure of the actual data in the classes as algebraic data types. For Java class methods, I defined a type class with type signatures for the methods. Then, data types can be instances of that class, with their own specific definitions for the class methods, just like Java.

If a subclass introduces a new method, then I would create a new type class that derives the superclass with the signature for the new method.

If you have a subclass in Java with a data field that overrides one in the superclass, I don't think you can really do that in Haskell, but there wouldn't be a need to because algebraic data types are different: they're not a hierarchy like Java classes, but like a web.

I really liked this exercise because it made me realize that structures in Haskell do a much better job of separating data (algebraic data types) and control (type classes and instances) than Java, where the object classes have data (instance variables) and control (methods) grouped closely together.

10.

(I didn't make constructors or the error() function, but I think semantically it's all there)

```
public class Expr {
    Expr e;
    public int eval() { return e.eval(); }
}

class Val extends Expr {
    private int val;
    public int eval() { return val; }
}

class BinOp extends Expr {
    public enum Op { Mul, Div }
    private Op op;
    private Expr e1;
    private Expr e2;
    public int eval() {
        if (op == Op.Mul) return (e1.eval()) * (e2.eval());
        else if (op == Op.Div) {
            int divisor = e2.eval();
            if (divisor==0) error("Division by zero.");
            else return (e1.eval()) / divisor;
        }
       else { error("Invalid expr"); }
   }
}
```