Warm up: Type Classes

Type classes are one of the most characteristic features of Haskell. They were introduced as a way of function overloading. It works by introducing a new type class, containing a method for the function you're overloading. You then write instances for the types for which you want the function to be available.

Defining Type Classes

A new type class contains the type signatures for all the methods of this class. Syntactically, this works as follows:

```
class TypeClass TypeVars where
    method1 :: SomeType
    ...
    methodN :: SomeType
```

For example, consider the `Show` class:

```
class Show a where
    show :: a -> String
```

As a second example, consider the `Eq` class. Here we define the `==` operator, which in this definition is written in the prefix notation `(==)`.

```
class Eq a where
    (==) :: a -> a -> Bool
```

Instances for this class then contain all the method implementations, for this specific type. Syntactically, this looks as follows:

```
instance TypeClass Type where
    method1 = SomeImplementation
    ...
    methodN = SomeImplementation
```

Consider again the `Show` class as an example:

```
instance Show Bool where
    show True  = "True"
    show False = "False"
```

Expressions

For this exercise, consider the following small ADT, representing expressions:

```
data MyBool = MyTrue
            | MyFalse

data Exp = Const MyBool
         | And Exp Exp
         | Or Exp Exp
```

1. Write an instance for the `Eq` class for `MyBool` and `Exp`. This class contains the `(==)` method, which checks whether the two arguments are equal. You should not evaluate the expressions for this exercise, and simply check whether they are literally equal. Don't write more than 4 cases for the implementation of the `(==)` function.

2. Write an instance for the `Show` class for `MyBool` and `Exp`. This class contains the `show` method, which converts the argument to a `String`. Ignore the use of parentheses for this exercise.

3. Define the `Evaluatable` class, which contains a method `eval`. This method evaluates its argument to a `Bool`.

4. Finally write instances for this class for both the `MyBool` and `Exp` type.

```
Main> MyTrue == MyTrue
True
Main> MyTrue == MyFalse
False
Main> (And (Const MyTrue) (Const MyFalse)) == (Const MyFalse)
False

Main> show MyTrue
"True"
Main> show (And (Const MyTrue) (Const MyFalse))
"True && False"
Main> show (Or (And (Const MyTrue) (Const MyFalse)) (Const MyTrue))
"True && False || True"

Main> eval MyFalse
False
Main> eval (Const MyTrue)
True
Main> eval (And (Or (Const MyFalse) (Const MyTrue)) (Const MyTrue))
True
```