An Overview of Haskell Type Classes

Prof. Susan Older

7 October 2019

A Simple Example

Consider the following code for summing up a list of Integers:

```
sumUp :: [Integer] -> Integer
sumUp [] = 0
sumUp (x:xs) = x + sumUp xs
```

If, instead, we want to sum up a list of Floats:

```
sumUp :: [Float] -> Float
sumUp [] = 0
sumUp (x:xs) = x + sumUp xs
```

The code looks identical, so why not make it polymorphic?

```
sumUp :: [a] -> a
sumUp [] = 0
sumUp (x:xs) = x + sumUp xs
```

The problem: 0 and + are not defined for all types!

Type Classes: The Basic Idea

Type classes are "clubs for types":

- Like clubs, type classes have membership requirements:

  Any type that meets the requirements can join.

- Like clubs, type classes provide membership benefits.

For example, Haskell includes the Eq type class:

- To join, a type *t* must provide at least one of the following:

  ```
  (==) :: t -> t -> Bool
  (/=) :: t -> t -> Bool
  ```

  This is overloading: each type *t* has its own definition for == and /=.

- As a sample membership benefit, Eq types can make use of:

  ```
  elem :: Eq a => a -> [a] -> Bool
  ```

Some of Haskell's Built-In Type Classes

The Prelude introduces these classes (and others):

- Eq – equality
- Ord – types with ordering comparisons (less than, equal, greater than)
- Bounded – types with upper and lower bounds
- Enum – sequentially ordered types
- Num – numbers
- Integral – whole numbers
- Floating – floating-point numbers
- Show – types that can be displayed
- Read – types that can be read

Back to Our Example

Recall our previous two versions of sumUp:

```
sumUp :: [Integer] -> Integer
sumUp [] = 0
sumUp (x:xs) = x + sumUp xs
```

```
sumUp :: [Float] -> Float
sumUp [] = 0
sumUp (x:xs) = x + sumUp xs
```

They can be generalized as follows:

```
sumUp :: Num a => [a] -> a
sumUp [] = 0
sumUp (x:xs) = x + sumUp xs
```

For More Information

Look in any of these places:

- Chapter 13 of the textbook (Thompson's *Haskell: The Craft of Functional Programming*, Third Edition)
- This week's lab (tangentially)
- Ghci (e.g., `:info Num`)
- Hoogle (http://www.haskell.org/hoogle/)
  Search for the `Prelude` or for specific type classes

For this course, what do you need to know?

☞ What overloading is

☞ What type classes are, and how to coexist with them

✗ How to define your own classes

✗ How to add types to type classes (except for: `deriving`)

✗ Which type classes depend on other type classes