Table of Contents

Type Classes

Benson Joeris
benson@bjoeris.com
http://bjoeris.com

pluralsight
hardcore developer training

Overview

- Type Class Instances
- Derived Type Classes
- Creating Type Classes
- Subclasses

Table of Contents

Type Class Instances

```
elem _ [] = False
elem x (y : ys)
  | x == y     = True
  | otherwise = elem x ys
```

Type Class Instances

```
elem :: Eq a => a -> [a] -> Bool
elem _ [] = False
elem x (y : ys)
  | x == y    = True
  | otherwise = elem x ys
```

Type Class Instances

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

```
data RGB = RGB Int Int Int

colors = [RGB 255 0 0, RGB 0 255 0, RGB 0 0 255]
green = RGB 0 255 0
greenInColors = elem green colors
```

Type Class Instances

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

```
data RGB = RGB Int Int Int
```

```
instance Eq RGB where
  (RGB r1 g1 b1) == (RGB r2 g2 b2) =
    (r1 == r2) && (g1 == g2) && (b1 == b2)
```

GHCi> `elem green colors`

Result: `True`

Type Class Instances

```
data RGB = RGB Int Int Int
```

```
instance Show RGB where
  show (RGB r g b) =
    "RGB " ++ (show r) ++ " " ++
    (show g) ++ " " ++ (show b)
```

```
GHCi> show (RGB 255 0 255)
Result: "RGB 255 0 255"
```

Table of Contents

Type Class Instances for Parameterized Types

```
data Maybe' a = Nothing' | Just' a
```

```
instance Eq (Maybe' a) where
  Nothing' == Nothing' = True
  Nothing' == (Just' _) = False
  (Just' _) == Nothing' = False
  (Just' x) == (Just' y) = x == y
```

Type Class Instances for Parameterized Types

```
data Maybe' a = Nothing' | Just' a
```

```
instance (Eq a) => Eq (Maybe' a) where
  Nothing' == Nothing' = True
  Nothing' == (Just' _) = False
  (Just' _) == Nothing' = False
  (Just' x) == (Just' y) = x == y
```

Table of Contents

Deriving Type Class Instances

```
data RGB = RGB Int Int Int
instance Eq RGB where
  (RGB r1 g1 b1) == (RGB r2 g2 b2) =
    (r1 == r2) && (g1 == g2) && (b1 == b2)
```

```
data Person = Person String Int Int
instance Eq Person where
  (Person name1 age1 height1) ==
    (Person name2 age2 height2) =
      (name1 == name2) && (age1 == age2) &&
      (height1 == height2)
```

Deriving Type Class Instances

```
data RGB = RGB Int Int Int
  deriving Eq
```

Deriving Type Class Instances

- `Eq`
  - Deriving –components-wise equality
- `Ord`
  - `(<), (>), (<=), (>=)`
  - Deriving –component-wise comparison
- `Show`
  - `show`
  - Deriving –``{Constructor-name} {argument-1} {argument2} …"
- `Read`
  - `read`
  - Deriving –parse output of default show

Deriving Type Class Instances

```
data Foo = Foo (Int -> Int)
  deriving Eq
```

Table of Contents

Defining Type Classes

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

Defining Type Classes

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

  x /= y = not (x == y)
  x == y = not (x /= y)
```

```
instance Eq RGB where
  (RGB r1 g1 b1) == (RGB r2 g2 b2) =
    (r1 == r2) && (g1 == g2) && (b1 == b2)
```

Defining Type Classes

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

  x /= y = not (x == y)
  x == y = not (x /= y)
```

```
instance Eq RGB where
  (RGB r1 g1 b1) /= (RGB r2 g2 b2) =
    (r1 /= r2) || (g1 /= g2) || (b1 /= b2)
```

Defining Type Classes

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

  x /= y = not (x == y)
  x == y = not (x /= y)
```

```
instance Eq RGB where
  (RGB r1 g1 b1) == (RGB r2 g2 b2) =
    (r1 == r2) && (g1 == g2) && (b1 == b2)
  (RGB r1 g1 b1) /= (RGB r2 g2 b2) =
    (r1 /= r2) || (g1 /= g2) || (b1 /= b2)
```

Defining Type Classes

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

  x /= y = not (x == y)
  x == y = not (x /= y)
```

- Minimum complete definition: `(==)` or `(/=)`

Defining Type Classes

```
data Point2 = Point2 Double Double
```

```
data Point3 = Point3 Double Double Double
```

```
distance2 :: Point2 -> Point2 -> Double
distance2 (Point2 x1 y1) (Point2 x2 y2) =
  sqrt (dx * dx + dy * dy)
  where dx = x1 - x2
        dy = y1 - y2
```

```
distance3 :: Point3 -> Point3 -> Double
distance3 (Point3 x1 y1 z1) (Point3 x2 y2 z2) =
  sqrt (dx * dx + dy * dy + dz * dz)
  where dx = x1 - x2
        dy = y1 - y2
        dz = z1 - z2
```

Defining Type Classes

```
pathLength2 :: [Point2] -> Double
pathLength2 [] = 0
pathLength2 (_ : []) = 0
pathLength2 (p0 : p1 : ps) =
  distance2 p0 p1 + pathLength2 (p1 : ps)
```

```
pathLength3 :: [Point3] -> Double
pathLength3 [] = 0
pathLength3 (_ : []) = 0
pathLength3 (p0 : p1 : ps) =
  distance3 p0 p1 + pathLength3 (p1 : ps)
```

Defining Type Classes

```
class Measurable a where
  distance :: a -> a -> Double
```

```
instance Measurable Point2 where
  distance = distance2
```

```
instance Measurable Point3 where
  distance (Point3 x1 y1 z1) (Point3 x2 y2 z2) =
    sqrt (dx * dx + dy * dy + dz * dz)
    where dx = x1 - x2
          dy = y1 - y2
          dz = z1 - z2
```

```
instance Measurable Double where
  distance x y = abs (x - y)
```

Defining Type Classes

```
class Measurable a where
  distance :: a -> a -> Double
```

```
pathLength :: Measurable a => [a] -> Double
pathLength [] = 0
pathLength (_ : []) = 0
pathLength (p0 : p1 : ps) =
  distance p0 p1 + pathLength (p1 : ps)
```

Table of Contents

Subclasses of Type Classes

- `Ord`
  - `(<), (>), (<=), (>=)`

```
class (Eq a) => Ord a where
  (<)     :: a -> a -> Bool
  (>)     :: a -> a -> Bool
  (<=)    :: a -> a -> Bool
  (>=)    :: a -> a -> Bool
  compare :: a -> a -> Ordering
  max     :: a -> a -> a
  min     :: a -> a -> a
```

```
data Ordering = LT | EQ | GT
```

- Minimum complete definition: `compare` or `(<=)`

Subclasses of Type Classes

```
data Point2 = Point2 Double Double
data Point3 = Point3 Double Double Double
```

```
class Measurable a where
  distance :: a -> a -> Double
```

```
class (Measurable a, Show a) => Directions a where
  getDirections :: a -> a -> String
```

Subclasses of Type Classes

```
data Point2 = Point2 Double Double
data Point3 = Point3 Double Double Double
```

```
class Measurable a where
  distance :: a -> a -> Double
```

```
class (Measurable a, Show a) => Directions a where
  getDirections :: a -> a -> String
  getDirections p1 p2 =
    "Go from " ++ (show p1) ++
    " towards " ++ (show p2) ++
    " and stop after " ++ (show (distance p1 p2))
```

Subclasses of Type Classes

```
data Point2 = Point2 Double Double
data Point3 = Point3 Double Double Double
```

```
class (Measurable a, Show a) => Directions a where
  getDirections :: a -> a -> String
  getDirections p1 p2 =
    "Go from " ++ (show p1) ++
    " towards " ++ (show p2) ++
    " and stop after " ++ (show (distance p1 p2))
```

```
instance Directions Point3 where
  getDirections p1 p2 =
    "Fly from " ++ (show p1) ++
    " towards " ++ (show p2) ++
    " and stop after " ++ (show (distance p1 p2))
```

Subclasses of Type Classes

```
data Point2 = Point2 Double Double
  deriving Show
data Point3 = Point3 Double Double Double
  deriving Show
```

```
class (Measurable a, Show a) => Directions a where
  getDirections :: a -> a -> String
  ...
```

```
instance Directions Point3 where
  getDirections p1 p2 =
    "Fly from " ++ (show p1) ++
    " towards " ++ (show p2) ++
    " and stop after " ++ (show (distance p1 p2))
```

Subclasses of Type Classes

```
data Point2 = Point2 Double Double
  deriving Show
data Point3 = Point3 Double Double Double
  deriving Show
```

```
class (Measurable a, Show a) => Directions a where
  getDirections :: a -> a -> String
  ...
```

```
instance Directions Point2 where
```

Table of Contents

Summary

- Instances
- Deriving
- Defining type classes
- Subclasses