# Objectives

## Type Classes

#### Dr. Mattox Beckman

Illinois Institute of Technology Department of Computer Science

- Describe the concept of *polymorphism*.
- Show how to declare instances of a type class.
- Understand the Eq, Ord, Show, and Read type classes.



Dr. Mattox Beckman (IIT) Type Classes Motivation

1 / 11

Dr. Mattox Beckman (IIT)

Type Classes Motivation

Type Classes

◆□▶ ◆□▶ ◆■▶ ◆■▶ ■ 900

Polymorphism

# Overloading

- We often want to use the *same operation* on things of *different type*.
- How can we do that?

Dr. Mattox Beckman (IIT)

- Overloading C++ like languages
- Inheritance Object oriented languages
- Parameterized Types Hindley Milner typed languages (Haskell, SML, etc.); C++ (templates), Java (generics)

Type Classes

• Type Classes — Haskell

```
int inc(int i) {
   return i + 1;
double inc(double i) {
   return i + 1.0;
```

Dr. Mattox Beckman (IIT)

#### Inheritance

```
public class Shape {
   public int loc_x,loc_y;
public class Square extends Shape {
   public int width,height;
```

```
public class List<E> {
   public E data;
   public List<E> next;
data Cons a = Cons a (Cons a)
            | Nil
```

Parametric Polymorphism

◆□▶◆□▶◆臺▶◆臺▶ 臺 釣魚@ Type Classes

Dr. Mattox Beckman (IIT)

Type Classes

Type Classes

Type Classes

Dr. Mattox Beckman (IIT)

Type Classes

## The Eq Type Class

## class Eq a where (==), (/=) :: a -> a -> Bool -- Minimal complete definition: (==) or (/=)x /= y not (x == y) = not (x /= y)x == y

## Using Eq

```
data Foo = Foo Int
x = Foo 10
y = Foo 10
```

• If you try to compare these...

```
*Main> x == y
<interactive>:1:3:
   No instance for (Eq Foo)
     arising from a use of '=='
   Possible fix: add an instance declaration for (Eq Foo)
   In the expression: x == y
   In an equation for 'it': it = x == y
```

Dr. Mattox Beckman (IIT)

◆□▶◆□▶◆■▶◆■▶ ■ 900 Type Classes

Type Classes Type Classes

Use an Instance

```
tl;dc
```

```
instance Eq Foo where

(==) (Foo i) (Foo j) = i == j
```

• Now if you try to compare these...

```
*Main> let x = Foo 10

*Main> let y = Foo 10

*Main> x == y
True
```

- Too long! Didn't Code!
- Let Haskell do the work.

```
data Foo = Foo Int
  deriving Eq
```

```
    (コトイラトイミト (基) ) 至 かので
    (コトイラトイミト (基) ) 至 かので
    (ロトイラトイミト (基) ) 至 がので
    (ロトイラト (基) ) 至 がので
    (ロトイラトイミト (基) ) 至 がので
    (ロトイラト (基) ) 至 がので
    (ロトイライン (基) ) 至 がので
    (ロトイラト (基) ) 至 がので
    (ロトイラトイミト (基) ) 至 がので
    (ロトイラト (基) ) 至 がので
    (ロトイライン (基) ) 至 がので
    (ロトイラトイン (基) ) 至 がので
    (ロトイラト (基) ) 至 がので
    (ロトイラン (基) ) 至 がので
    <
```

#### The Ord Typeclass

max, min

Dr. Mattox Beckman (IIT)

#### 

:: a -> a -> a

# The Show Typeclass

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

instance Show Foo where

data Foo = Foo Int
-- one way...
    deriving (Show,Eq)

-- other way...
instance Show Foo where
    show (Foo i) = "Foo " ++ show i
```

 $\max x y = \text{if } x \le y \text{ then } y \text{ else } x$  $\min x y = \text{if } x \le y \text{ then } x \text{ else } y$ 

Type Classes

Type Classes

◆□ > ◆□ > ◆豆 > ◆豆 > ・豆 \* りへで

Type Classes

# The Read Typeclass

```
{-# LANGUAGE ViewPatterns #-}
import Data.List

instance Read Foo where
   read (stripPrefix "Foo " -> Just i) = Foo (read i)

• Sample run...

*Main> let x = "Foo 10"

*Main> read it :: Foo
Foo 10
```



Dr. Mattox Beckman (IIT)

Type Classes

13 / 11