

Type Class: The Ultimate Ad Hoc

George Wilson

Data61/CSIRO

george.wilson@data61.csiro.au

August 3, 2017



Type classes are a language feature

- ▶ Haskell
- ▶ Eta
- ▶ Purescript
- ▶ Clean

Type classes are a language feature

- ▶ Haskell
- ▶ Eta
- ▶ Purescript
- ▶ Clean

or sometimes a design pattern

- ▶ Scala
- ▶ OCaml

Polymorphism

A type is *polymorphic* iff it can be applied at multiple types.

Polymorphism is good

- ▶ less duplication
- ▶ more reuse
- ▶ fewer possible implementations

Broadly speaking there are two major forms of polymorphism:

- ▶ *parametric* polymorphism
- ▶ *ad-hoc* polymorphism

A type is parametrically polymorphic iff it has at least one *type parameter* which can be instantiated to *any type*.

```
reverse :: [a] -> [a]
```

```
id :: a -> a
```

```
(.) :: (b -> c) -> (a -> b) -> (a -> c)
```


Ad-hoc polymorphism

A type which is ad-hocly polymorphic can be instantiated to different types, and may behave differently at each type

Type Classes

```
class Equal a where  
  eq :: a -> a -> Bool
```

```
class Equal a where  
  eq :: a -> a -> Bool
```

```
data Person = Person {  
  age :: Int  
, name :: String  
}
```

```
class Equal a where  
    eq :: a -> a -> Bool
```

```
data Person = Person {  
    age :: Int  
, name :: String  
}
```

```
instance Equal Person where  
    eq p1 p2 = eq (age p1) (age p2) && eq (name p1) (name p2)
```

```
elementOf :: Equal a => a -> [a] -> Bool  
elementOf a list = any (eq a) list
```

Instances can be constrained

```
instance Equal a => Equal (Maybe a) where
  eq Nothing Nothing = True
  eq (Just x) (Just y) = True
  eq (Just x) Nothing = False
  eq Nothing (Just y) = False
```

Instances can be constrained

```
instance Equal a => Equal (Maybe a) where
  eq Nothing Nothing = True
  eq (Just x) (Just y) = True
  eq (Just x) Nothing = False
  eq Nothing (Just y) = False
```

We can add type class instances for types we didn't write

Interfaces

```
interface Equal<A> {  
    public boolean eq(A other);  
}
```

```
interface Equal<A> {  
    public boolean eq(A other);  
}  
  
class Person implements Equal<Person> {  
    public int age;  
    public String name;  
  
    public boolean eq(Person other) {  
        return this.age == other.age && this.name.equals(other.name);  
    }  
}
```

```
static <A extends Equal<A>> boolean elementOf(A a, List<A> list) {  
    for (A element : list) {  
        if (a.eq(element)) return true;  
    }  
    return false;  
}
```


Implicit