

# Type Class: The Ultimate Ad Hoc

George Wilson

Data61/CSIRO

[george.wilson@data61.csiro.au](mailto:george.wilson@data61.csiro.au)

August 2, 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

Let's go beyond the basics of type classes

# Polymorphism

## Polymorphism is good

- ▶ greater reuse
- ▶ less repetition
- ▶ fewer names need inventing
- ▶ fewer possible implementations

Broadly speaking there are two major forms of polymorphism:

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

## Parametric polymorphism (sometimes called *generics*)

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

Parametrically polymorphic functions behave the same way no matter which type they are instantiated to.



## Parametric polymorphism (sometimes called *generics*)

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

Parametrically polymorphic functions behave the same way no matter which type they are instantiated to.

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

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

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

# Ad-hoc polymorphism

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

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

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

```
eqBool :: Bool -> Bool -> Bool
```

```
eqBool True True  = True
```

```
eqBool False False = True
```

```
eqBool False True  = False
```

```
eqBool True False  = False
```

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

```
eqBool :: Bool -> Bool -> Bool
```

```
eqBool True True    = True
```

```
eqBool False False  = True
```

```
eqBool False True   = False
```

```
eqBool True False   = False
```

```
eqString :: String -> String -> Bool
```

```
eqString [] []      = True
```

```
eqString (_: _) []  = False
```

```
eqString [] (_: _)  = False
```

```
eqString (c:cs) (d:ds) = eqChar c d && eqString cs ds
```

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

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

```
public 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);  
    }  
}
```

```
import java.util.List;
```

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

elementOf exhibits ad-hoc polymorphism



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