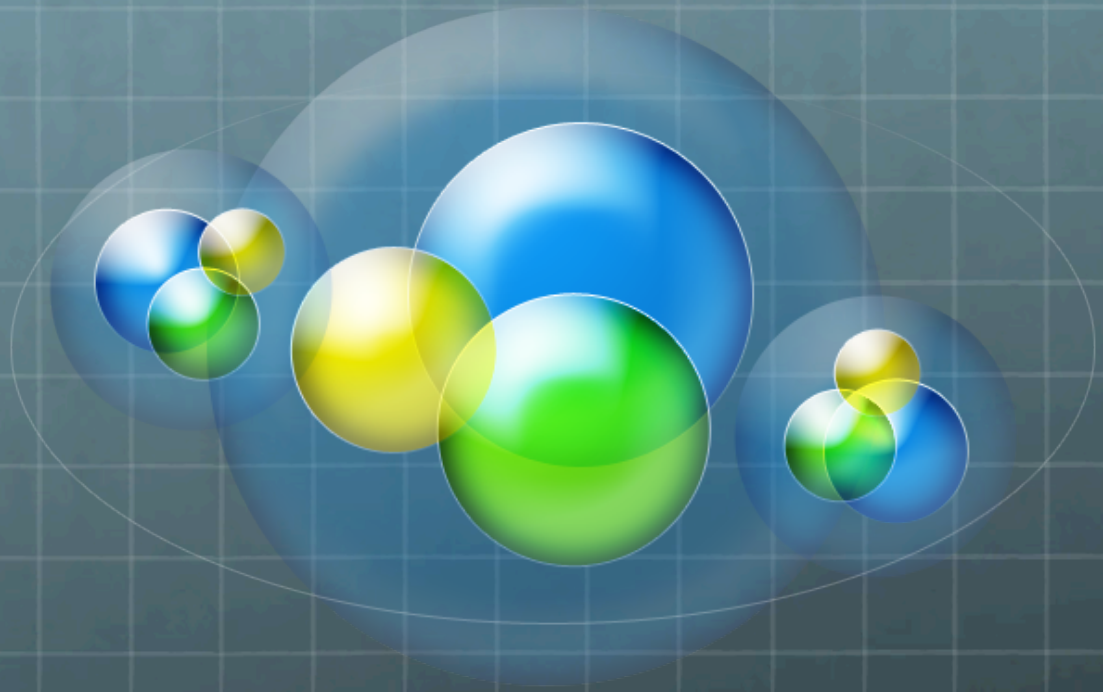

Composable Equality

in Java

What's wrong with java.lang.Object.equals(Object)?

- Objects of different types should not be equatable
- Weak type – you always cast
- Inheritance
- Not all types are equatable
- Object reference equality is a useless default
- A default is useless
- If you forget to override equals, it's still equatable

What's wrong with java.lang.Object.equals(Object)?

- "If you override equals, you must also override hashCode"
  - Seriously, WTF????
- For A<B>, A is usually equatable only if B is
- Optional<String> is comparable, but Optional<UnfathomableEnigma> is not
  - How to represent this in the Java type system?

Eq<T>

```
interface Eq<T> {
  boolean eq(T a, T b);
}
```

To the code!

https://github.com/techtangents/composableEq

So...

- java.lang.Object.equals(Object) is broken
- Not all types are equatable
- A type T is equatable if Eq<T> is inhabited
- Given an Eq<T>,
  you can construct an Eq<Optional<T>>
- Eq<T>s can be composed with &&, ||
- Eqs for an object can be constructed given:
  - Eqs for the field types
  - Getters for the field values

Haskell does it better

- Haskell does this better using type classes

```
data Mayhap a = None | Some a
instance Eq a => Eq (Mayhap a) where
  None      == None      = True
  None      == (Some _) = False
  (Some _) == None      = False
  (Some a) == (Some b) = a == b
```

- ... or simply

```
data Mayhap a = None | Some a deriving Eq
```

Scala does it better

- Scalaz Equals trait