Typeclasses everywhere

Wilber Chao 2018/12/20

TYPECLASSES

Let's take a look of the typeclasses code

TYPECLASSES

```
// A type class to provide textual representation
trait Show[A] {
  def show(f: A): String
}
```

TYPECLASSES

1. An abstraction interface
2. A type variable
3. An abstraction method

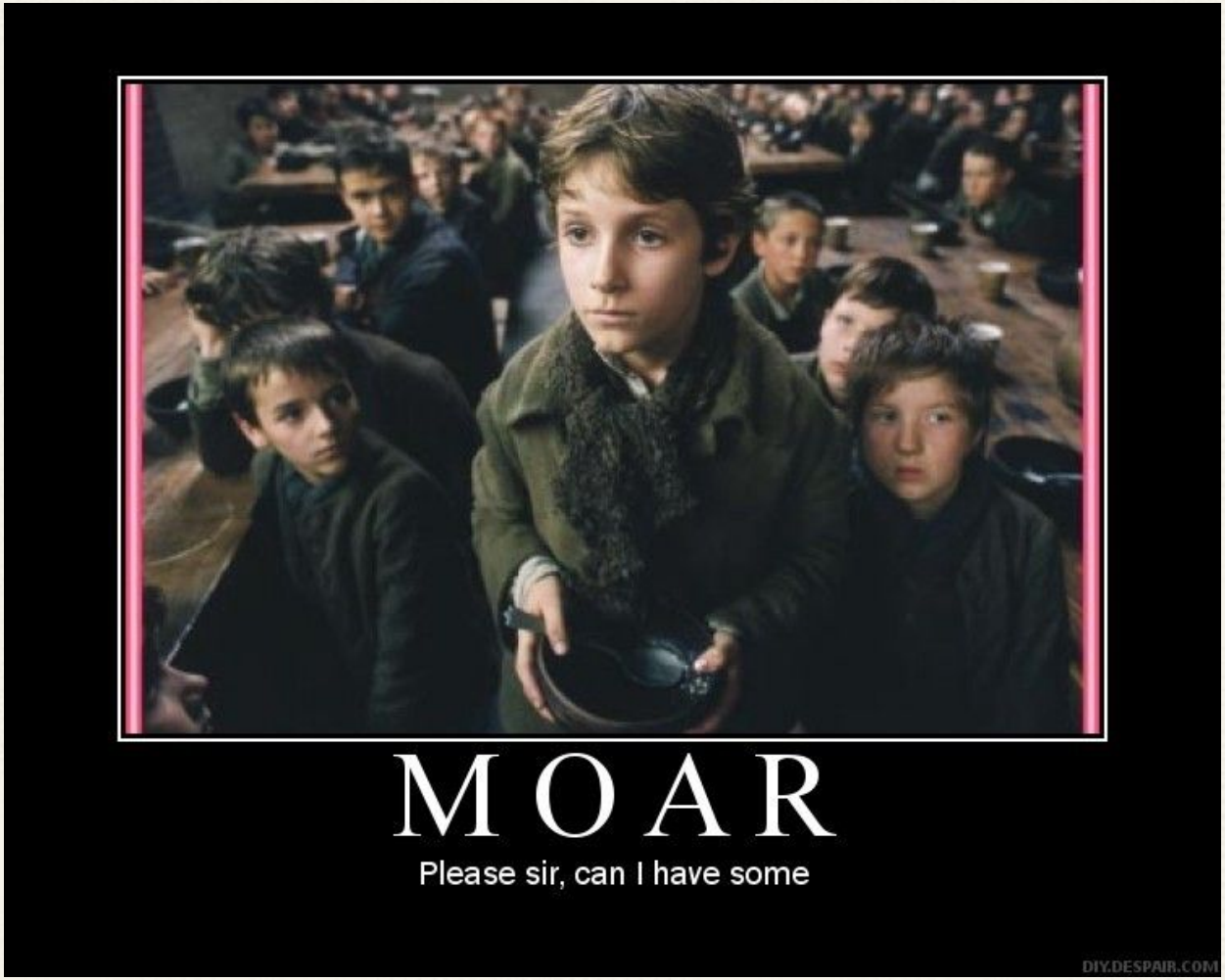
WHICH IS TYPECLASSES

- sealed abstract class Option[+A] extends Product with Serializable
- sealed abstract class Either[+A, +B] extends Product with Serializable
- trait FSM[S, D] extends Actor with Listeners with ActorLogging //Akka Finite State Machine
- trait DatabaseConfig[P <: BasicProfile] //Slick

WHAT IN YOUR CODE EVERYWHERE

How much is your test to code ratio

UNIT TEST



Unit test to code ratio: 1:1 line of code is about right. If you have 20K lines of code, you probably ought to have about 20K lines of tests.

(Uncle Bob)

TEST YOU CODE

```
import com.google.cloud.storage.Bucket

class MyService(bucket: Bucket) {
    def getData(n: String) = bucket.get(blob)
}
```

A unit test should test functionality in isolation. Side effects from other classes or the system should be eliminated for a unit test, if possible. (Mockito)

CODE WITH MOCK

```
trait BucketLike {
    def get(n: String): Blob
}
```

CODE WITH MOCK

```
import com.google.cloud.storage.Bucket
class RealBucket(b: Bucket) extends BucketLike {
    def get(n: String): Blob = b.get(n)
}

class TestBucket extends BucketLike {
    def get(n: String): Blob = ??? // test mock result
}
```

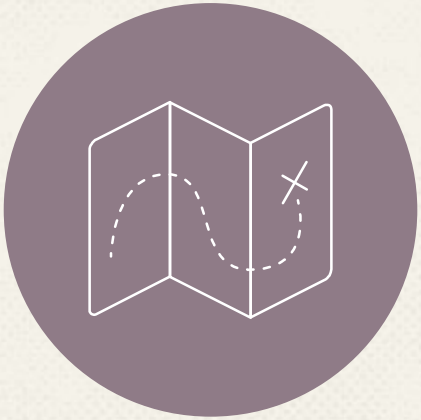
CODE WITH MOCK

```
import com.google.cloud.storage.Bucket

class MyService(bucket: BucketLike) {
    def getData(n: String) = bucket.get(blob)
}

val inTest = new MyService(new TestBucket())

val inProduction = new MyService(new RealBucket(???))
```



Haskell Typeclasses

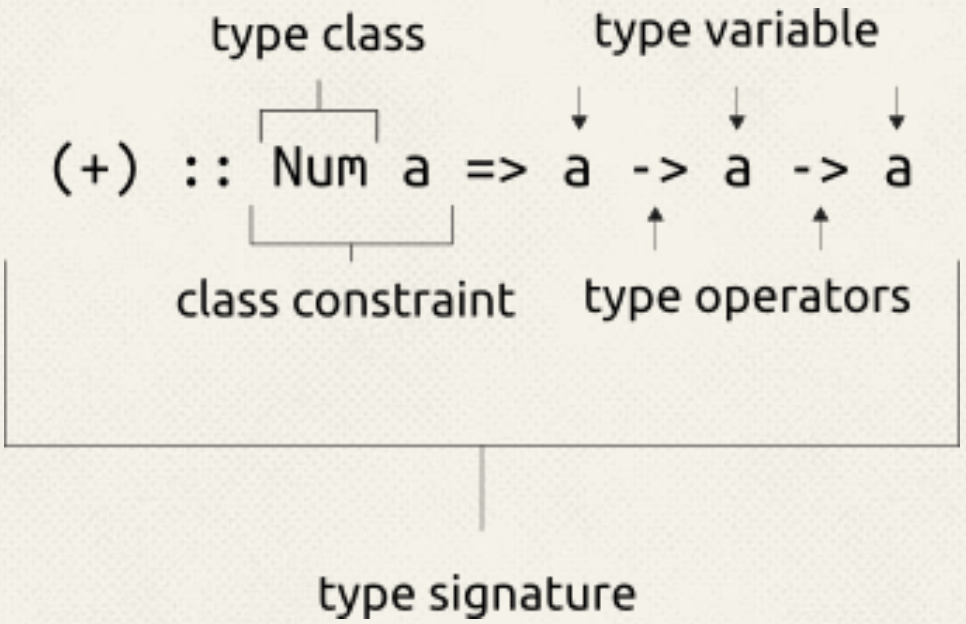
A paradigm shift from Haskell

IDEA OF TYPECLASSES

```
add :: (Num T) => T -> T -> T
```

```
def add[T](x: T)(y: T)(implicit n: Num[T]): T
```

TYPECLASSES CONSTRAINT



We have no idea of type variable, but we have the typeclasses to know what we can do

TYPECLASSES MOCK

```
trait BucketLike[T] {
  def get(b: T, n: String): Blob
}
Class List[A]
def length(l: List)
String => List[String]
```

TYPECLASSES MOCK

```
import com.google.cloud.storage.Bucket

object BucketLike {
  case class TestBucket()
  implicit val testBucket = new BucketLike[TestBucket] {
    override def get(b: TestBucket, n: String): Blob = ???
  }

  implicit val realBucket = new BucketLike[Bucket] {
    override def get(b: Bucket, n: String): Blob = b.get(n)
  }
}
```

TYPECLASSES MOCK

```
implicit class BucketSyntax[Bucket: BucketLike](b: Bucket) {
    def get(n: String) = implicitly[BucketLike[Bucket]].get(b, n)
}

class MyService[Bucket: BucketLike](b: Bucket){
  import BucketLike.BucketLikeSyntax

  def get(n: String) = b.get(n)
}
```

TYPECLASSES MOCK

```
//test suit
import BucketLike.testBucket
new MyService(TestBucket()).get(???)

//production main
import com.google.cloud.storage.Bucket
val b: Bucket = ???
new MyService(b).get(???)
```

Typeclasses polymorphism

- Typeclasses for test mock in unit test
- Provide instance via implicit scope for unique design
- Ad-hoc polymorphism
  - Each type variable has its own implementation
  - For each one type, you only can have one implement implicit instance - eliminating redundant, ambiguous, or confuse design

THANKS!

Any questions?

You can find me at:
cecol3500123@gmail.com