Bengal: Dotty Cats

Walter Chang

Agenda

- Scala 3 is coming
- What Bengal is not
- Algebraic Data Type
- Top Level Definitions
- Implied Instances
- Implied Imports
- Inferable Parameters
- Extension Methods
- Type Lambdas
- Opaque Type Aliases
- Auto Parameter Tupling
- Typeclass Derivation

Scala 3 is coming

- Feature freeze and developer preview by first half of 2019
- Officially it should come out by early 2020
- Martin has been working on it for the past 5 years
- Dotty releases (developer preview preview) every 6 weeks
- Good integration with VS Code via LSP
- Uses the same standard library as Scala 2
- Most of the Scala 2 libraries should be binary compatible (no macros though)
- Code rewriting tools will be available to ease migration pain

What Bengal is not

- Not a cat
- Not a cats / scalaz clone
- Not complete by any stretch of imagination

Algebraic Data Type

```
enum Maybe[+A] {
 case Just(x: A)
 case Empty
}
```

Top Level Definitions

```
package bengal.examples

type Path = String

val defaultPath = "/tmp"

var theCauseOfThatElusiveBug = 0

def add(x: Int, y: Int): Int = x + y
```

Implied Instances

```
implied [A] given Monoid[A] for Monoid[Option[A]] {
 def empty = None
 def (x: Option[A]) combine (y: Option[A]) = (x, y) match {
   case (Some(x), Some(y)) => Some(x |+| y)
   case _ => None
 }
}
implied [A, B] given (m1: Monoid[A], m2: Monoid[B]) for Monoid[(A, B)] {
 def empty = (m1.empty, m2.empty)
 def (x: (A, B)) combine (y: (A, B)) = (x._1 |+| y._1, x._2 |+| y._2)
}
```

Implied Imports

```
import implied bengal.instances.all._
```

Inferable Parameters

```
def f given (u: Universe) (x: u.T) given Context = ...
implied global for Universe { type T = String ... }
implied ctx for Context { ... }


f("abc")
(f given global)("abc")
f("abc") given ctx
(f given global)("abc") given ctx
```

Extension Methods

```
trait Monoid[A] {
 def empty: A
 def (x: A) combine (y: A): A
 def (x: A) |+| (y: A): A = x combine y
}
```

Type Lambdas

```
implied [I] for Monad[[X] => I => X] // Monad[I => ?] {
 def pure[A](x: A) = _ => x
 def (f: I => A => B) ap [A, B] (x: I => A) =
   y => f(y)(x(y))
 def (x: I => A) map [A, B] (f: A => B) = x andThen f
 def (x: I => A) flatMap [A, B] (f: A => I => B) =
   y => f(x(y))(y)
}
```

Opaque Type Aliases

```
opaque type SumInt = Int

object SumInt {
 def apply(x: Int): SumInt = x
 def value(x: SumInt): Int = x
}
```

Auto Parameter Tupling

```
implied [A: Monoid] for Monoid[List[A]] {
 def empty = Nil
 def (x: List[A]) combine (y: List[A]) =
   x.zip(y).map(_ |+| _)
}
```

Typeclass Derivation

```
enum Maybe[+A] derives Eql {
 case Just(x: A)
 case Empty
}
```

That’s All Folks