Pattern

# Type Class

ve Scale

Jakub Janeček
AVAST Software

# Type Class

- pochází z jazyka Haskell

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

- ad-hoc polymorfismus
- type class definuje **koncept** (“rozhraní”)
  - množina požadavků na nějaký typ
    - např. metoda pro řazení hodnot typu T může být napsána genericky za předpokladu, že víme jak hodnoty typu T porovnávat
- … ale s možností koncepty **skládat** a **bez dědičnosti**!

# Syntax

- currying

```
def cplus(a: Int)(b: Int) = a + b
```

- implicitní parametry

```
def iplus(a: Int)(implicit b: Int) = a + b
```

- context bound

```
def m[T: Context](value: T) = …

def m[T](value: T)(implicit context: Context[T]) = …
```

# Příklad 1

```
def getCount[T](v: T)(implicit counter: Countable[T]): Int = {
  counter.count(v)
}

trait Countable[-U] {
   def count(value: U): Int
}
```

```
implicit object StringCountable extends Countable[String] {
   def count(value: String) = value.size
}
implicit object ListCountable extends Countable[List[_]] {
   def count(value: List[_]) = value.size
}
object ListUniqueCountable extends Countable[List[_]] {
   def count(value: List[_]) = value.toSet.size
}

> getCount(“hello”) // 5
> getCount(List(1,1,2,2,3,3)) // 6
> getCount(List(1,1,2,2,3,3))(ListUniqueCountable) // 3
```

# Příklad 2

```
def accept[T: Acceptable](value: T) = value
```

```
trait Acceptable[U]
```

```
implicit object IntAcceptable extends Acceptable[Int]
```

```
scala> accept(100) // OK
```

```
scala> accept(100.0) // ERROR
```

```
error: could not find implicit value for evidence parameter of type Acceptable[Double]
```

# Příklad 3 - skládání

```
def min[T](x: T, y: T)(implicit c: Comparable[T]) = {
  if (c.less(x, y)) x else y
}

trait Comparable[T] {
  def less(a: T, b: T): Boolean
}

implicit object IntComparable extends Comparable[Int] {
  def less(a: Int, b: Int) = a < b
}

class ListComparable[U](implicit c: Comparable[U])
        extends Comparable[List[R]] {
  def less(a: List[U], b: List[U]) = c.less(a.head, b.head)
}
implicit def constructListComparable[V: Comparable] = new ListComparable[V]

> min(20, 10) // 10
> min(List(20), List(10)) // List(10)
```

# Použito v praxi

- Metoda sum na kolekcích (Numeric)

```
def sum[B >: A](implicit num: Numeric[B]): B
```

- Metoda sorted na kolekcích (Ordering)

```
def sorted[B >: A](implicit ord: Ordering[B]): List[A]
```

- CanBuildFrom v kolekcích

- ScalaZ a obecně v FP

# Dotazy?