Types algébriques de données: énumérations

Jean-Luc Falcone

09 mars 2023

Enums

Problème possible

```
def move( length: Int, dir: String ) = ???

move( 2, "N" )
move( -1, "E" )
move( 0, "W" )
```

Problème possible

```
def move( length: Int, dir: String ) = ???

move( 2, "N" )
move( -1, "E" )
move( 0, "W" )

move( 2, "NE" )
move( -1, "UP"
move( 0, "TO THE MOON" )
```

Solution: enums

```
enum Dir {
  case N
  case S
  case W
  case E
}
```

ou bien (syntaxe allégée):

```
enum Dir {
  case N, S, W, E
}
```

Utilisation

```
enum Dir {
  case N, S, W, E
}

def move( length: Int, dir: Dir ) = ???

move( 2, Dir.N )
move( -1, Dir.E )
move( 0, Dir.W )
```

Avec imports

```
import Dir.*
move( 2, N )
move( -1, E )
move( 0, W )
```

Pattern matching

```
case class Point( x: Double, y: Double ) {

  def move( length: Int, dir: Dir ) = dir match {
    case Dir.N => copy( y=y+length)
    case Dir.S => copy( y=y-length)
    case Dir.E => copy( x=x+length)
    case Dir.W => copy( x=x-length)
  }

}
```

Pattern matching: exhaustif (1)

```
enum Choice {
  case Yes,No,Maybe
}

def toBool( c: Choice ) = c match {
  case Choice.Yes => true
  case Choice.No => false
}
```

Pattern matching: exhaustif (2)

```
def toBool( c: Choice ) = c match {
   case Choice.Yes => true
   case Choice.No => false
}

c match {
^
match may not be exhaustive.
It would fail on pattern case: Maybe
```

Pattern matching: exhaustif (3)

```
def foo( c1: Choice, c2: Choice ) = (c1,c2) match {
   case (Choice.Yes, Choice.No) => "YN"
   case (_,Choice.Yes) => "*Y"
}

(c1,c2) match {
^^^^^^^

match may not be exhaustive.
It would fail on pattern case:
    (No, No), (Maybe, No), (_, Maybe)
```

Autres fonctionnalités

```
enum Choice {
  case Yes,No,Maybe
}

No.ordinal //=> 1
Choice.valueOf("Maybe") //=> Maybe
Choice.fromOrdinal(0) //=> Yes
Choice.values //=> Array(Yes,No,Maybe)
```

More enums

Champs

```
enum Color( r: Byte, g: Byte, b: Byte) {
  case Red   extends Color(255,0,0)
  case Green extends Color(0,255,0)
  case Blue  extends Color(0,0,255)
  case Black extends Color(0,0,0 )
  case White extends Color(255,255,255)
}

Color.Red.r == 255
Color.Green.b == 0
```

Constructeurs

```
enum Color {
  case RGB( r: Byte, g: Byte, b: Byte )
  case Gray( level: Byte )
}

val c1 = Color.RGB( 12, 231, 43 )
val c2 = Color.Gray( 127 )
```

Pattern matching

```
enum Color {
  case RGB( r: Byte, g: Byte, b: Byte )
  case Gray( level: Byte )
}

def isGray( col: Color ) = col match {
  case Color.Gray(_) => true
  case Color.RGB(x,y,z) =>
    x == y && y == z
}
```

Méthodes

```
enum Color {
  case RGB( r: Byte, g: Byte, b: Byte )
  case Gray( level: Byte )

  def toGray: Color = this match {
    case Gray(_) => this
    case RGB(r,g,b) => Gray((r+g+b)/3)
  }
}

val c1 = Color.RGB(128, 64, 32)
val c2 = c1.toGray
```