Lecture: Inside Regular Expressions

```
sealed trait Regex {

  def >>(other: Regex): Regex = (this, other) match {
    case (Zero, _) => Zero
    case (_, Zero) => Zero
    case (re1, One) => re1
    case (One, re2) => re2
    case (re1, re2) => Seq(re1, re2)
  }

  def |(other: Regex): Regex = (this, other) match {
    case (Zero, re2) => re2
    case (re1, Zero) => re1
    case (re1, re2) => if (re1 == re2) re1 else Alt(re1, re2)
  }
}

case class Character(ch: Char) extends Regex
case class Seq(re1: Regex, re2: Regex) extends Regex
case class Alt(re1: Regex, re2: Regex) extends Regex
case class Star(re: Regex) extends Regex
/** Accepts no strings. */
case object Zero extends Regex
/** Accepts the empty string. */
case object One extends Regex

object Macther {

  def empty(re: Regex): Regex = re match {
    case Character(_) => Zero
    case Alt(re1, re2) => empty(re1) | empty(re2)
    case Seq(re1, re2) => empty(re1) >> empty(re2)
    case Star(_) => One
    case One => One
    case Zero => Zero
  }

  def deriv(regex: Regex, ch: Char): Regex = regex match {
    case Character(ch_) => if (ch == ch_) One else Zero
    case One => Zero
    case Zero => Zero
    case Alt(re1, re2) => deriv(re1, ch) | deriv(re2, ch)
    case Seq(re1, re2) => (deriv(re1, ch) >> re2) | (empty(re1) >> deriv(re2, ch))
    case Star(r) => deriv(r, ch) >> Star(r)
  }

  def reMatch(regex: Regex, str: List[Char]): Boolean = str match {
    case Nil => empty(regex) == One
    case ch :: rest => reMatch(deriv(regex, ch), rest)
  }

}
```