Advanced Pattern Matching

Advanced Pattern Matching

We can define our own patterns with unapply

```scala
class Person(val name: String, val age: Int)

object Person {
  def unapply(person: Person): Option[(String, Int)] =
    Some((person.name, person.age))
}
```

the instance being decomposed

results as an Option (simple value or tuple)

```scala
person match {
  case Person(name, _) => println(s"Hi, I'm $name.")
}
```

the compiler searches for the appropriate unapply

Patterns are independent of classes we decompose

- e.g. pattern can belong to Person, but we deconstruct a String

Advanced Pattern Matching

Infix patterns

```
numbers match {
  case head :: Nil => println("single element" + head)
  // equivalent:
  case ::(head, Nil) => println("single element" + head)
}
```

Match sequences

```
object MyList {
  def unapplySeq[A](list: MyList[A]): Option[Seq[A]] = // ...
}

myList match {
  case MyList(1,2, _*) => // ...
}
```

Rare PM: boolean patterns, custom unapply return types

Scala rocks