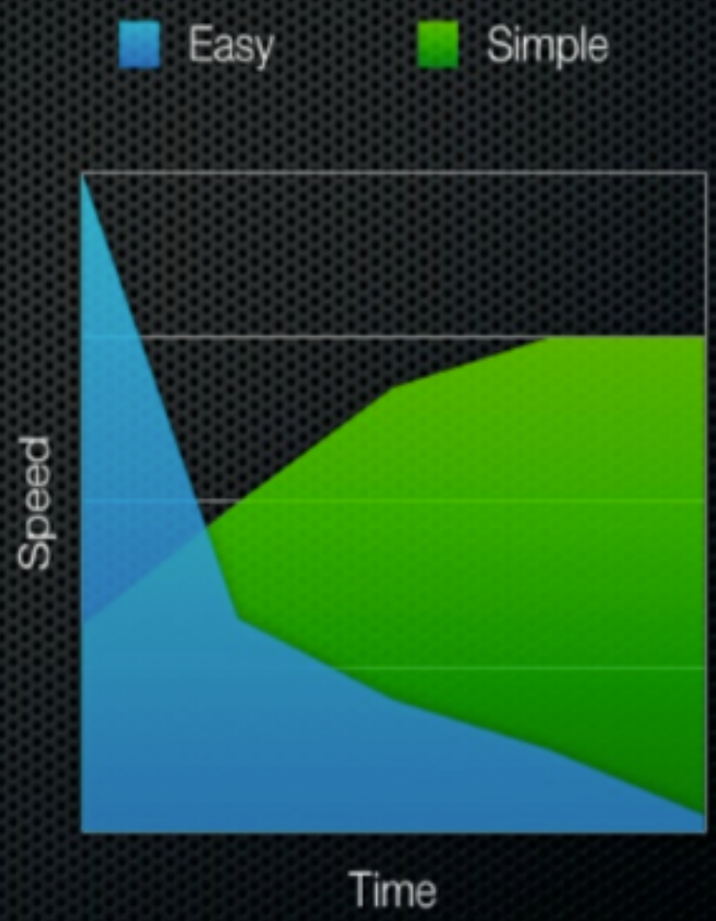
GYAKORLATI SCALA 9. HÉT

Scala DSL
és más egyszerűség növelők
(Simple Made Easy)
(Rich Hickey)

Development Speed

- Emphasizing ease gives early speed
- Ignoring complexity will slow you down over the long haul
- On throwaway or trivial projects, nothing much matters



https://www.infoq.com/presentations/Simple-Made-Easy

https://github.com/matthiasn/talk-transcripts/blob/master/Hickey_Rich/SimpleMadeEasy.md

https://www.slideshare.net/evandrix/simple-made-easy

Abstraction for Simplicity

- Abstract
  - drawn away
- vs Abstraction as complexity *hiding*
- Who, What, When, Where, Why and How
- I don't know, I don't want to know

Pragmatic Scala

```scala
import scala.language.implicitConversions
import java.time.LocalDate
import java.time.format.{DateTimeFormatter, FormatStyle}


object DayDslApp extends App {

  printDate(2 days ago)

  printDate(5 days from_now)


  sealed trait TimeDirection

  object ago extends TimeDirection
  object from_now extends TimeDirection

  implicit class DaysDsl(n: Int) {

    def days(d: TimeDirection): LocalDate = d match {
      case `ago` => today.minusDays(n)
      case `from_now` => today.plusDays(n)
    }

    private def today = LocalDate.now
  }

  def printDate(date: LocalDate): Unit =
    println(
      DateTimeFormatter.ofLocalizedDate(FormatStyle.SHORT).format(date)
    )
}
```

```
val dist: Length = Kilometers(160)
val time: Time = 1.hours + 40.minutes
val velocity: Velocity = dist / time

println(s"If I arrive to Budapest $time from now " +
  s"my average speed is ${velocity in KilometersPerHour}")

val consumption = 7.litres / 100.kilometers
println(s"If the consumption of my car is 7 litres / 100” +
s”kilometers I will need ${consumption * dist in Litres} fuel")

val forty = KilometersPerHour(40)
val acc: Acceleration = forty / Seconds(5)

println(s"If I drive with $forty speed ${Seconds(5)} after " +
  s"I got a green light my acceleration is $acc")

val gravity = 32.feet / second.squared
println(s"Gravity is $gravity”)
```

GITHUB.COM/SCALA-RULES/EXAMPLES

```
Given (always)
Calculate
    DefaultPaidHealthCost is 0.euro and
    DefaultMinimumOwedTaxes is 0.euro
  ,

Given (always)
Calculate
  BaseIncomeTax is BaseIncome * FlatTaxRate and
   BaseHealthCosts is first(TotalPaidHealthCost, DefaultPaidHealthCost) and
   HealthCostEligibleForReimbursement is BaseHealthCosts * HealthCostRsmentPercent and
. . .


Given (LegallyOwedTaxes < TotalPrepaidTaxes)
Calculate
  TaxReturnAmount is TotalPrepaidTaxes - LegallyOwedTaxes
,

Given (LegallyOwedTaxes >= TotalPrepaidTaxes)
Calculate
  TaxDueAmount is LegallyOwedTaxes - TotalPrepaidTaxes
```

State is Never Simple

- Complects value and time
- It *is* easy, in the at-hand and familiar senses
- Interweaves everything that touches it, directly or indirectly
  - Not mitigated by modules, encapsulation
- Note - this has nothing to do with asynchrony

What

- Operations
- Form abstractions from related sets of functions
  - *Small* sets
- Represent with polymorphism constructs
- Specify inputs, outputs, semantics
  - Use only values and other abstractions
- Don’t complect with:
  - How

Who

- Entities implementing abstractions
- Build from subcomponents direct-injection style
  - Pursue many subcomponents
    - e.g. policy
- Don't complect with:
  - component details
  - other entities

How

- Implementing logic
- Connect to abstractions and entities via polymorphism constructs
- Prefer abstractions that don't dictate *how*
  - Declarative tools
- Don't complect with:
  - anything

When, Where

- Strenuously avoid complecting these with anything in the design
- Can seep in via directly connected objects
  - Use queues

Why

- The policy and rules of the application
- Often strewn everywhere
  - in conditionals
  - complected with control flow etc
- Explore rules and declarative logic systems

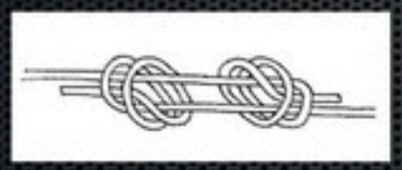
Information *is* Simple

- Don't ruin it
- By hiding it behind a micro-language
  - i.e. a class with information-specific methods
  - thwarts generic data composition
  - ties logic to representation du jour
- Represent data as data


Simplicity is not an objective in art, but one achieves simplicity despite one's self by entering into the real sense of things
Constantin Brancusi

Simplifying

- Identifying individual threads/roles/dimensions
- Following through the user story/code
- Disentangling

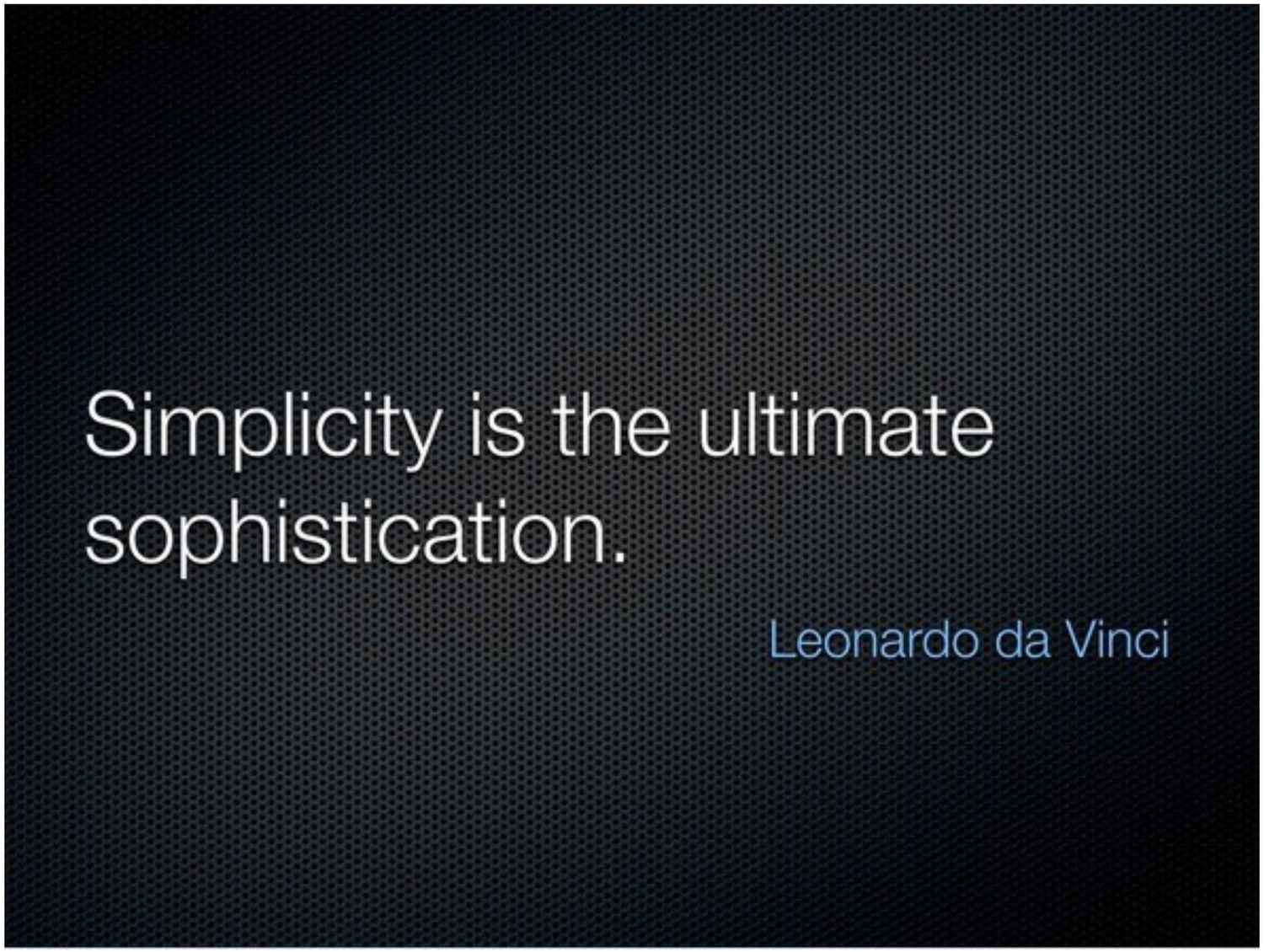


Simplicity is a Choice

- Requires vigilance, sensibilities and care
- Your sensibilities equating simplicity with ease and familiarity are wrong
  - Develop sensibilities around entanglement
- Your 'reliability' tools (testing, refactoring, type systems) don't care
  - and are quite peripheral to producing good software

Simplicity Made Easy

- Choose simple constructs over complexity-generating constructs
  - It's the artifacts, not the authoring
- Create abstractions with simplicity as a basis
- Simplify the problem space before you start
- Simplicity often means making more things, not fewer
- Reap the benefits!


Simplicity is the ultimate sophistication.
Leonardo da Vinci