Level Styles

Introduction

- *What do we mean by meta-programming?*
  - Writing code that writes code
- *What's the motivation for this?*
  - You can do most things in Scala with three simple patterns: ADTs, pattern matching, and type classes
  - We can make code shorter and easier to read by making good use of reusable high-level patterns (monads, monoids, etc)
  - However, sometimes we end up with boilerplate and we want to get rid of it
  - If we've got a lot of "mechanical-looking" code, we might look at meta-programming to get rid of it:
    - Lucy the dog
- *We'll look at three types of meta-programming*
  - The types are:
    - Macros
    - Shapeless
    - Code generation
  - We could also throw compiler plugins and reflection in here
    - In Scala, there's a lot of overlap between these and macros
    - Reflection is unfashionable in Scala because it happens at runtime
  - When we look at each approach, we're interested in:
    - What is it good or bad at?
    - When should we use or not use it?
    - What are some cool examples?
    - What are some simple examples (to give you an in)?
  - It's going to be fairly high-level and example-based, with a few GIFs for good measure

The Punchline

- *Macros are good at syntaxy stuff, shapeless is good at typey stuff*
- *Neither approach is good without a solid non-meta-programming base*
  - Think about your library design to enable your developers work without meta-programming
  - Sprinkle meta-programming techniques as shortcuts for boilerplatey

Macros

- *Four types of macros:*
  - def macros
  - string interpolator macros
  - macro annotations
  - implicit macros
- *Def macros (blackbox):*
  - What are they?
    - Macros that look like method calls
  - What are they good for?
    - Looking at the syntax in the surrounding context
    - Expanding to an expression
    - Error messages
    - Really fast!
  - What are they bad at / what do you need to look out for?
    - Hygiene
    - Separate compolation
    - Inspecting sealed types prior to SI-7046 (mostly fixed by Miles in Typelevel Scala 2.12.0 and Lightbent Scala 2.12.1)
    - Introducing identifiers (see macro annotations)
  - What are some examples?
    - Monocle *GenLens[A](_.accessor)*
      https://github.com/julien-truffaut/Monocle/blob/master/macro/shared/src/main/scala/monocle/macros/internal/Macro.scala#L13-L54
    - macwire *wire[A]*
      https://github.com/adamw/macwire/blob/master/macros/src/main/scala/com/softwaremill/macwire/MacwireMacros.scala#L10-L31
    - enumeratum *findValues*
      https://github.com/lloydmeta/enumeratum/blob/master/macros/src/main/scala/enumeratum/EnumMacros.scala#L13-L19
    - checklist *validator.field(_.accessor)*
      https://github.com/davegurnell/checklist/blob/develop/src/main/scala/checklist/RuleMacros.scala#L10-L17
  - Design point:
    - Good library code works without macros
      - Checklist is monads, applicative functors, Ior, and a custom data type, Rule
      - The macros just avoid one tiny bit of syntax
    - Macros can't do runtime stuff (obviously)
      - But you typically *need* runtime stuff (e.g. to read input from the user)
      - See macros as setting up a board game:
        - "I want to have fun playing a board game"

- Macros represent setting up the board and the pieces, ready to play, before your friends get there
- Runtime represents actually playing with your friends
- The game doesn't work without rolling dice and cheating and abusing your friends

- *String interpolator macros:*
  - What are they?
    - Same as def macros
  - What are they good for?
    - Parsing completely non-Scala-type DSLs into Scala
  - What are they bad at?
    - Same as def macros
  - What are some examples?
    - unindent
      https://github.com/davegurnell/unindent/blob/develop/src/main/scala/unindent/package.scala#L19-L29
    - compose
      https://github.com/underscoreio/compose/blob/develop/core/src/main/scala/compose/tablature/TablatureSyntax.scala#L14-L30
    - Good blog post on type-safe email address macros
      http://blog.xrrocha.net/2014/08/type-safe-strings-with-scala-macros.html
    - TODO: THERE'S GOT TO BE SOME BETTER EXAMPLES OF EMBEDDED LANGUAGES: CSS? HTML? PROLOG?
  - Design point:
    - Design a dumb ADT to represent what you want (e.g. Score)
    - Write a macro that:
      - parses a string (using parser combinators, parboiled, or some existing library)
      - EITHER just says "ok that's valid" and leaves the string to be parsed again at runtime
      - OR expands into a larger syntax expression that removes the need to parse the string at runtime
      - (if you have a simple target expression language, like in compose, you can build an instance of it in the macro and then traverse it to turn it back into syntax)
        https://github.com/underscoreio/compose/blob/develop/core/src/main/scala/compose/tablature/TablatureSyntax.scala#L41-L48
- *Macro annotations:*
  - What are they?
    - Expanded before type-checking
    - Operate on a definition and its context
    - Can expand into multiple definitions
  - What are they good for?
    - Introducing new definitions
  - What are they bad at?
    - Anything to do with types
    - Being picked up by IDEs (unless you implement a custom plugin for IntelliJ)
      https://blog.jetbrains.com/scala/2015/10/14/intellij-api-to-build-scala-macros-support/
  - What are some examples?
    - Monocle @Lenses (and show the example of @Prisms)
      https://github.com/julien-truffaut/Monocle/blob/master/macro/shared/src/main/scala/monocle/macros/Lenses.scala
      https://github.com/cb372/Monocle/blob/3290ca6e01c57c08e0a0dfc755efbf774551cb7/macro/shared/src/main/scala/monocle/macros/Prisms.scala#L30-L34
    - Smartypants @smart (and its genesis from @smarts, and the dumb string-naming-matching version of @smarts that I'm probably going to end up with)
      https://github.com/davegurnell/smartypants/blob/develop/src/main/scala/smartypants/Macros.scala
      https://github.com/davegurnell/smartypants/blob/feature/smarts/src/main/scala/smartypants/Macros.scala#L9-L46
  - Design points:
    - Macro annotations are super flexible
    - They allow us to introduce
- *Implicit macros:*
  - What are they?
    - Macros marked with the *implicit* keyword
  - What are they good for?
    - Default instances of type classes
  - What are they bad at?
    - Nothing more than the above
  - What are some examples?
    - CSV example from "Macros for the Rest of Us"
      https://github.com/underscoreio/essential-macros/blob/master/csv/lib/src/main/scala/CsvMacros.scala#L8-L24
    - Shapeless... !!!

Shapeless

- *What is it?*
  - "Generic programming"
    - Expressing data types in a generic way
    - Breaking them down into common components
    - Solving problems in a generic space
  - So it's type oriented, as opposed to syntax oriented
    - Built on top of implicit resolution and dependent types
    - Small parts implemented using macros, enabling boilerplate-free derivation
- *What is it good for?*

- Main use case: type class derivation
- Also type calculation

What is it bad for?

Compilation times, mitigated partially by:

- Inductive implicit resolution
- cachedImplicit

Two primary use cases:

- Type calculation ("lemma" pattern)
- Type class derivation

Examples:

- Spray JSON Shapeless
  https://github.com/fommil/spray-json-shapeless/blob/master/src/main/scala/fommil/sjs/RecordFormats.scala
- Pureconfig
  https://github.com/pureconfig/pureconfig/blob/master/core/src/main/scala/pureconfig/DerivedConverters.scala
- Cartographer map indices
  https://bitbucket.org/untyped/cartographer/src/5cb3dffdfcd585b88a43b3bbf43b98c926d3a283/api/api/src/main/scala/cartographer/map/urs/UrsIndex.scala?fileviewer=file-view-default#UrsIndex.scala-41:112

Design points:

- Again, we're only using shapeless as a thin layer
- The underlying mechanism is a type class
- We can use the type class independently of shapeless
- We can even mix the type class and shapeless

Code generation

Why talk about this?

There are clear down-sides to macros and shapeless:

- Macros can be difficult to get right in some cases

What is it?

Two forms:

- generating Scala code from another language
- generating code in other languages from Scala