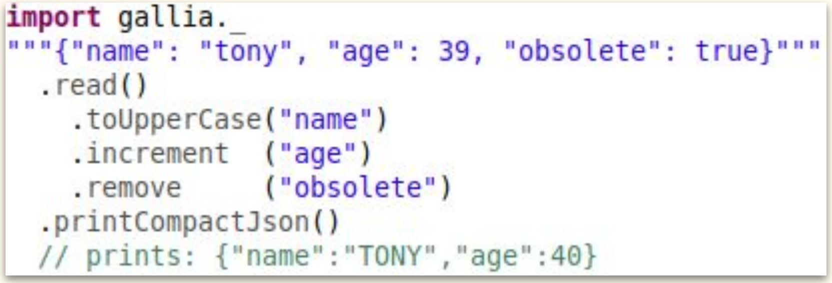
Migrating Gallia to Scala 3

—

the good, the bad, and the very good.

What is Gallia?



```
import gallia._
"""{"name": "tony", "age": 39, "obsolete": true}"""
  .read()
    .toUpperCase("name")
    .increment  ("age")
    .remove     ("obsolete")
  .printCompactJson()
  // prints: {"name":"TONY","age":40}
```

- **Github page:** github.com/galliaproject/gallia-core

- **Towards Data Science (1):** "*Gallia: A Library for Data Transformation*" (towardsdatascience.com/gallia-a-library-for-data-transformation-3fafaaa2d8b9)

- **Towards Data Science (2):** "*Data Transformations in Scala with Gallia: Version 0.4.0 Is Out*" (towardsdatascience.com/data-transformations-in-scala-with-gallia-version-0-4-0-is-out-f0b8df3e48f3)

- **Scala Days 2023 Seattle:** "*Gallia: Practical Data Transformation in Scala*" (youtube.com/watch?v=hl4GiFNCUv8)

What makes Gallia interesting? (in terms of migration)

- It's a library

- It's data-centric (so... → **types**)

- It integrates with Apache **Spark** (optionally)

- It relies on **reflection** in many places

- Had to delve head first in Scala 3's **metaprogramming** features

  → not an expert at it!

Multiple attempts

- **First attempt:** Spring 2021

  → chickened out

- **Second attempt:** Spring 2023

  → chickened out

- **Third attempt:** Summer 2023

  → success-ish!

Scary first error messages

"*value* ***runtime*** *is not a member of* ***reflect***"

"Incompatible combinations of ***tabs and spaces*** *in indentation prefixes"*

Not a full "migration"

- Still supporting Scala **2.12** (thank you, AWS EMR)

- Intellij **IDEA** support for Scala 3 buggy?

- Still relying on **enumeratum** for enums, not Scala 3's

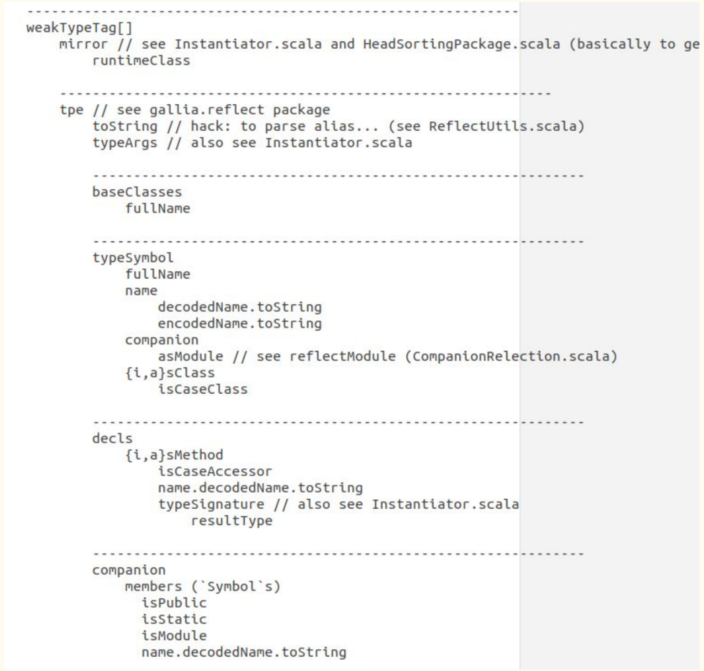
- Will have to rewrite **all** macros to go to/from case classes

  → see dedicated gallia-macros module

So where to start migration?

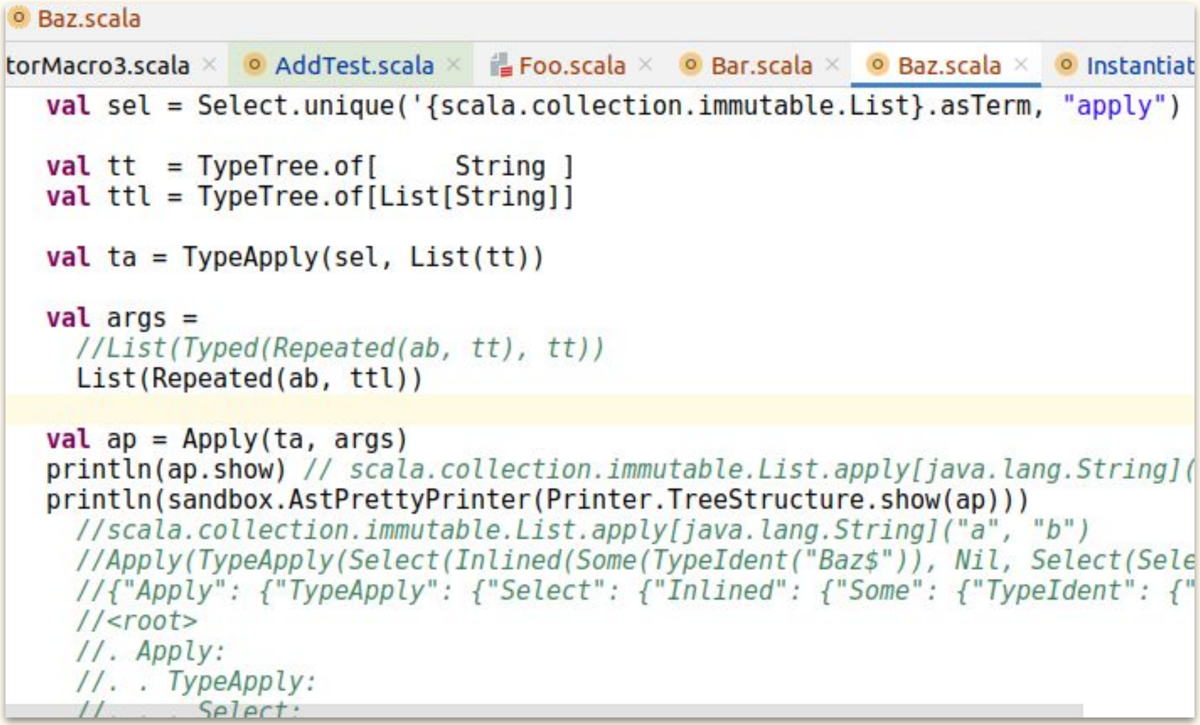
- Early changes are from last summer
- Hazy on some details...

The lay of the (reflected) land



```
------------------------------------------------------------
weakTypeTag[]
    mirror // see Instantiator.scala and HeadSortingPackage.scala (basically to ge
        runtimeClass

    ------------------------------------------------------------
    tpe // see gallia.reflect package
        toString // hack: to parse alias... (see ReflectUtils.scala)
        typeArgs // also see Instantiator.scala

        ------------------------------------------------------------
        baseClasses
            fullName

        ------------------------------------------------------------
        typeSymbol
            fullName
            name
                decodedName.toString
                encodedName.toString
            companion
                asModule // see reflectModule (CompanionRelection.scala)
            {i,a}sClass
                isCaseClass

        ------------------------------------------------------------
        decls
            {i,a}sMethod
                isCaseAccessor
                name.decodedName.toString
                typeSignature // also see Instantiator.scala
                    resultType

        ------------------------------------------------------------
        companion
            members (`Symbol`s)
              isPublic
              isStatic
              isModule
              name.decodedName.toString
```

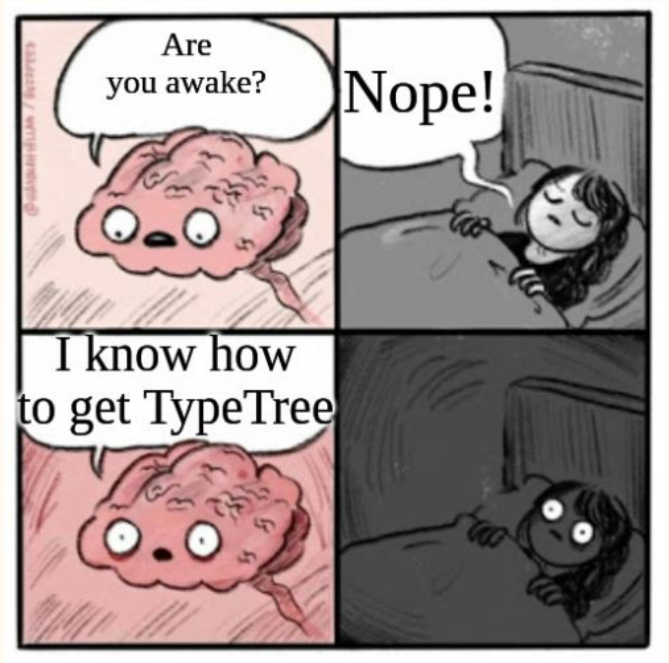
Poking around in **scala.quoted** land



Baz.scala

torMacro3.scala × | AddTest.scala × | Foo.scala × | Bar.scala × | Baz.scala × | Instantiat

```scala
  val sel = Select.unique('{scala.collection.immutable.List}.asTerm, "apply")

  val tt  = TypeTree.of[     String ]
  val ttl = TypeTree.of[List[String]]

  val ta = TypeApply(sel, List(tt))

  val args =
    //List(Typed(Repeated(ab, tt), tt))
    List(Repeated(ab, ttl))

  val ap = Apply(ta, args)
  println(ap.show) // scala.collection.immutable.List.apply[java.lang.String](
  println(sandbox.AstPrettyPrinter(Printer.TreeStructure.show(ap)))
    //scala.collection.immutable.List.apply[java.lang.String]("a", "b")
    //Apply(TypeApply(Select(Inlined(Some(TypeIdent("Baz$")), Nil, Select(Sele
    //{"Apply": {"TypeApply": {"Select": {"Inlined": {"Some": {"TypeIdent": {"
    //<root>
    //. Apply:
    //. . TypeApply:
    //. . . Select:
```



Errors encountered

"Good" errors

- *"'=' expected, but '{' found"* —> add "**: Unit =**"
- *"method X in Y must be called with **()** argument"* —> add ()
- *"**parentheses** are required around the parameter of a lambda"* —> add ()
- *"value X is not a **member of Object**"* (anonymous classes)
    - over reliance on my part on **def foo = new { def bar = ...**

—> create **class**

- `target.actualOpt.map(`**`_Rename`**`)` —> .map(_Rename**.apply**)

"Good" errors (Cont')

- "*result type of **implicit** definition needs to be given **explicitly***"

- "*value X needs result type because its right-hand side attempts **implicit** search*"

- "***Unbound** placeholder parameter; incorrect use of _*" (for self-types)

...

⇒ be more explicit about your implicits! eg `implicit val ctk: ClassTag[K] = ctag[K]`

"Bad" errors

Indentation errors?!

- "*Incompatible combinations of tabs and spaces in* ***indentation*** *prefixes*

- "*The start of this line does not match any of the previous* ***indentation*** *widths*"

- "*Un****indent*** *expected, but eof found*"

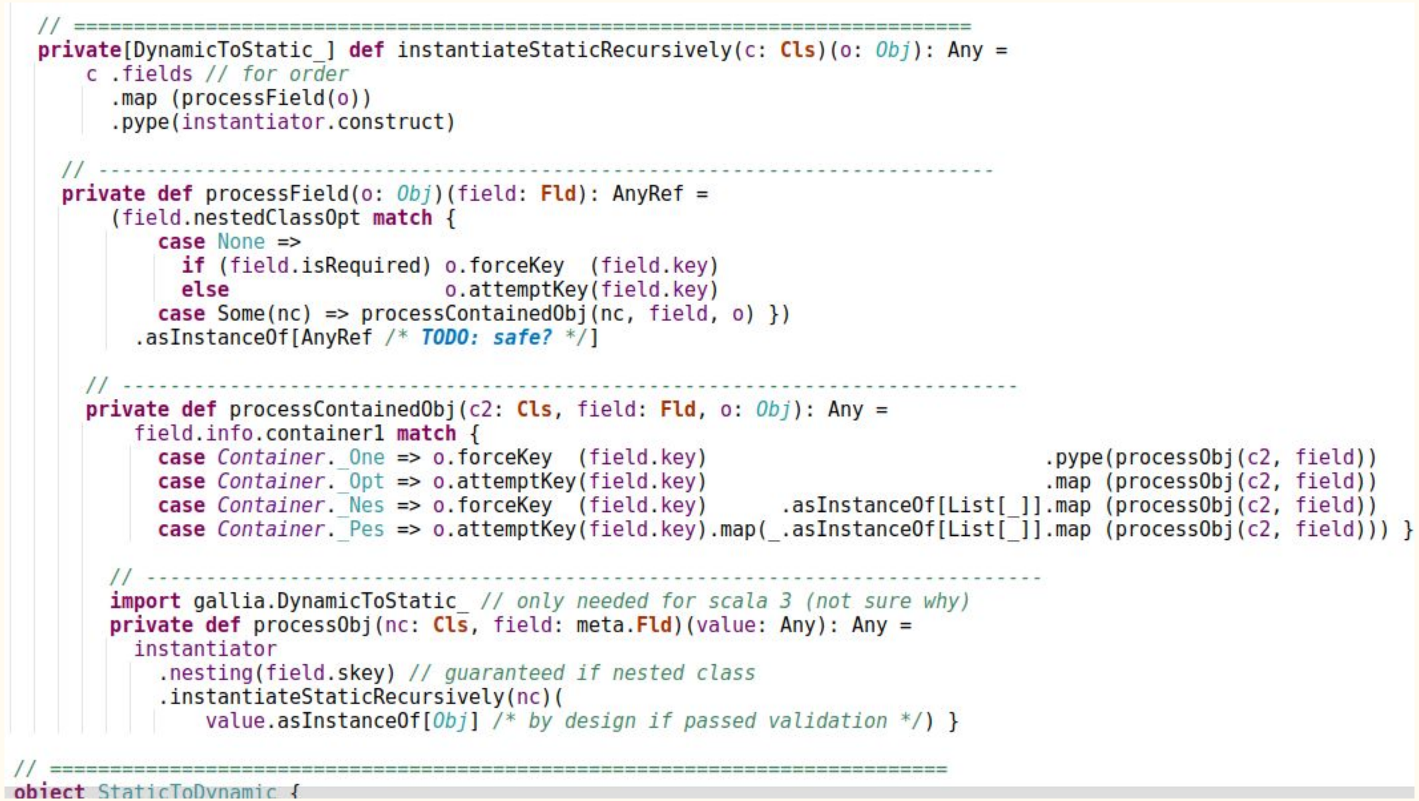
- "*An identifier expected, but* ***indent*** *found*"

- "*Line is* ***indented*** *too far to the right, or a `{` or `:` is missing*"

==

- "*Illegal start of statement: no* ***modifiers*** *allowed here*" → <u>private</u> def..

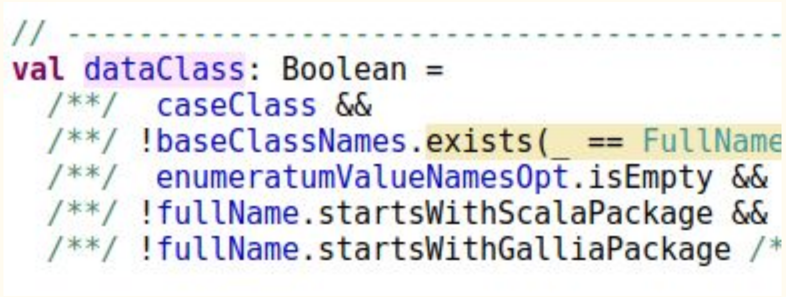
Idiosyncratic indentation+semantics



```
  // ============================================================================
  private[DynamicToStatic_] def instantiateStaticRecursively(c: Cls)(o: Obj): Any =
      c .fields // for order
        .map (processField(o))
        .pype(instantiator.construct)

    // ---------------------------------------------------------------------------
    private def processField(o: Obj)(field: Fld): AnyRef =
        (field.nestedClassOpt match {
            case None =>
              if (field.isRequired) o.forceKey  (field.key)
              else                  o.attemptKey(field.key)
            case Some(nc) => processContainedObj(nc, field, o) })
          .asInstanceOf[AnyRef /* TODO: safe? */]

      // ---------------------------------------------------------------------------
      private def processContainedObj(c2: Cls, field: Fld, o: Obj): Any =
          field.info.container1 match {
            case Container._One => o.forceKey  (field.key)                            .pype(processObj(c2, field))
            case Container._Opt => o.attemptKey(field.key)                            .map (processObj(c2, field))
            case Container._Nes => o.forceKey  (field.key)      .asInstanceOf[List[_]].map (processObj(c2, field))
            case Container._Pes => o.attemptKey(field.key).map(_.asInstanceOf[List[_]].map (processObj(c2, field))) }

        // ---------------------------------------------------------------------------
        import gallia.DynamicToStatic_ // only needed for scala 3 (not sure why)
        private def processObj(nc: Cls, field: meta.Fld)(value: Any): Any =
          instantiator
            .nesting(field.skey) // guaranteed if nested class
            .instantiateStaticRecursively(nc)(
                value.asInstanceOf[Obj] /* by design if passed validation */) }

// ============================================================================
object StaticToDynamic {
```

Significant Indentation issue (concrete example)

"***type mismatch***" error:



```
// -------------------------------------
val dataClass: Boolean =
  /**/  caseClass &&
  /**/ !baseClassNames.exists(_ == FullName
  /**/  enumeratumValueNamesOpt.isEmpty &&
  /**/ !fullName.startsWithScalaPackage &&
  /**/ !fullName.startsWithGalliaPackage /*
```

A new hope: **-no-indent** flag

But newcomers will still encounter all the error messages listed earlier...

"Odd" errors

- *"scalac: Error: class dotty.tools.dotc.core.Symbols**$NoSymbol$** cannot be cast to class dotty.tools.dotc.core.Symbols$ClassSymbol"*

- "*type mismatch*" error with "*Found = **X** and Required = **X***"

- Required with Scala 3 but not Scala 2:

  - "`Predef.`" prefix needed for `assert` with utest (e.g. see SquashingTest)

  - `import gallia.DynamicToStatic._` (for recursive call in extension method)

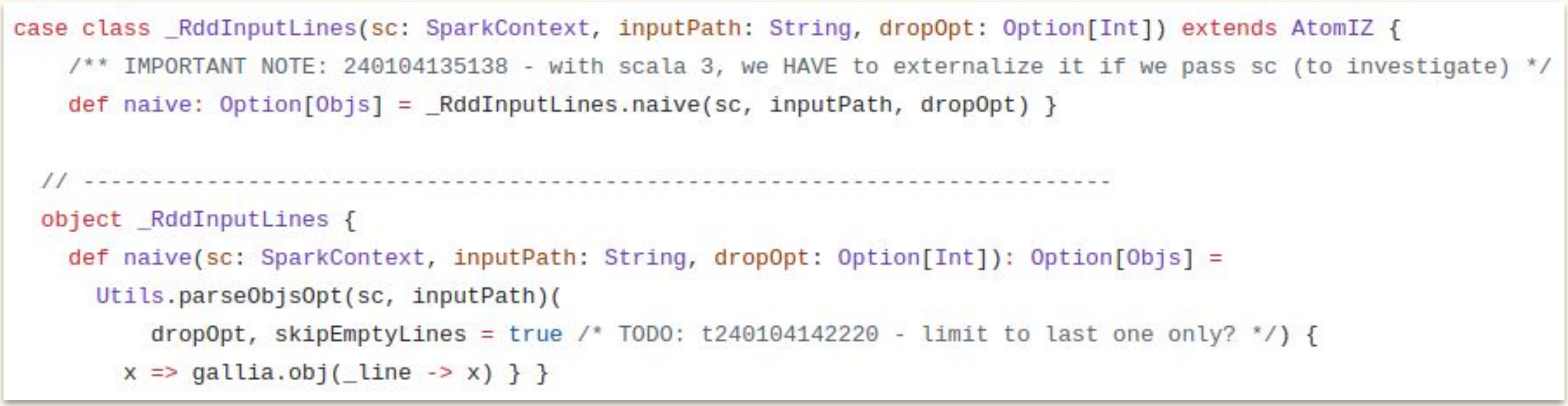
- Extra casting necessary, eg in WTT (dependent types?):

```
given _string: WTT[String] = WttBuiltIns._String.asInstanceOf[WTT[String]]
```

Tricky problems

- the dreaded "`org.apache.spark.SparkException: Task not serializable`"

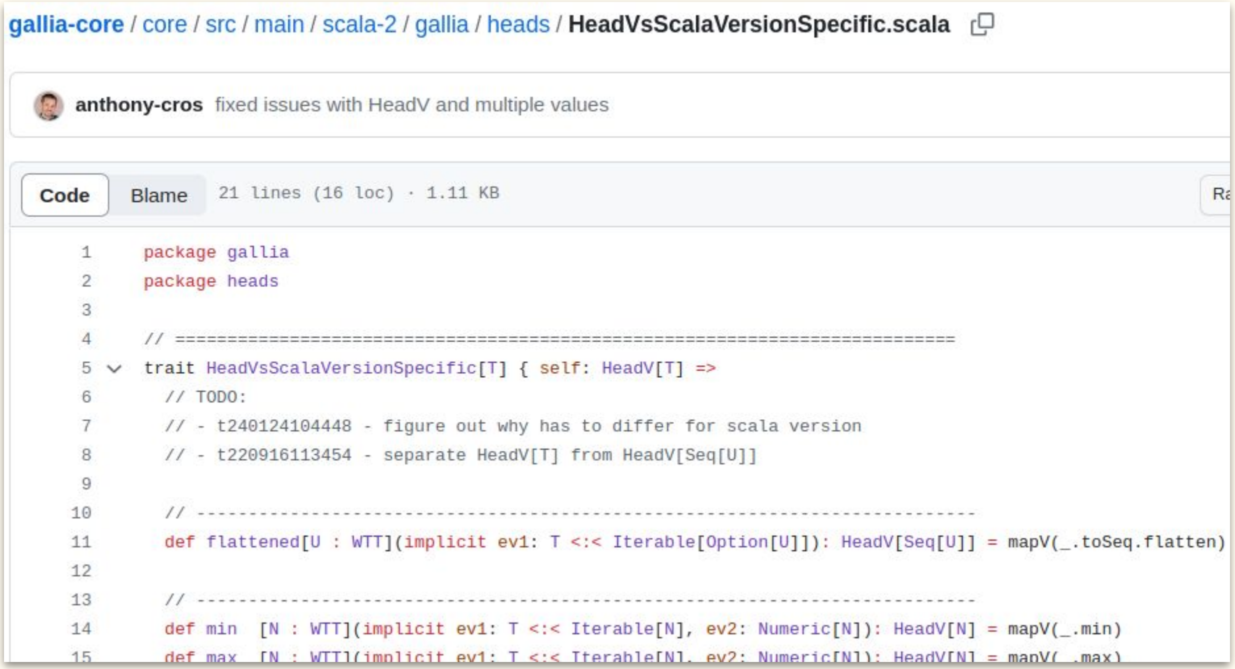
→ Serialization issues with Spark: see _RddInputLines



```
case class _RddInputLines(sc: SparkContext, inputPath: String, dropOpt: Option[Int]) extends AtomIZ {
    /** IMPORTANT NOTE: 240104135138 - with scala 3, we HAVE to externalize it if we pass sc (to investigate) */
    def naive: Option[Objs] = _RddInputLines.naive(sc, inputPath, dropOpt) }

  // ---------------------------------------------------------------------------

  object _RddInputLines {
    def naive(sc: SparkContext, inputPath: String, dropOpt: Option[Int]): Option[Objs] =
      Utils.parseObjsOpt(sc, inputPath)(
          dropOpt, skipEmptyLines = true /* TODO: t240104142220 - limit to last one only? */) {
        x => gallia.obj(_line -> x) } }
```

Tricky problems (Cont')

WTT+implicit evidences (mystery) - *HeadVsScalaVersionSpecific.scala*: 2.13 vs 3.3.1



gallia-core / core / src / main / scala-2 / gallia / heads / **HeadVsScalaVersionSpecific.scala**

anthony-cros fixed issues with HeadV and multiple values

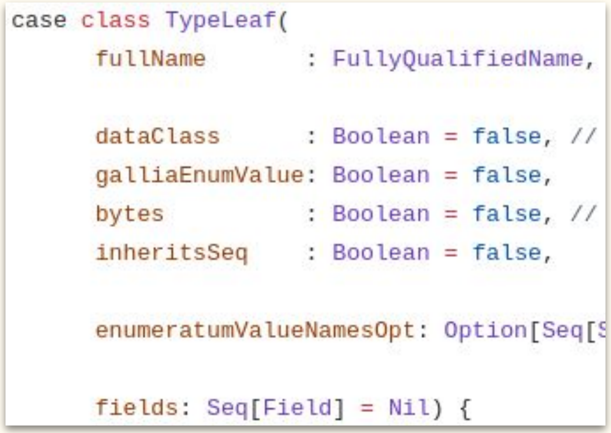
Code Blame 21 lines (16 loc) · 1.11 KB Ra

```scala
package gallia
package heads

// ============================================================================
trait HeadVsScalaVersionSpecific[T] { self: HeadV[T] =>
  // TODO:
  // - t240124104448 - figure out why has to differ for scala version
  // - t220916113454 - separate HeadV[T] from HeadV[Seq[U]]

  // ---------------------------------------------------------------------------
  def flattened[U : WTT](implicit ev1: T <:< Iterable[Option[U]]): HeadV[Seq[U]] = mapV(_.toSeq.flatten)

  // ---------------------------------------------------------------------------
  def min  [N : WTT](implicit ev1: T <:< Iterable[N], ev2: Numeric[N]): HeadV[N] = mapV(_.min)
  def max  [N : WTT](implicit ev1: T <:< Iterable[N], ev2: Numeric[N]): HeadV[N] = mapV(_.max)
```

Runtime type information

- No drop-in replacement for **(Weak)TypeTag** in Scala 3
- **izumi-reflect**: Interesting but... minimal + different API anyway

  → might as well learn to use the new **macros** system
- What I needed:
  - Ability to **instantiate** case classes
  - TypeNode / TypeLeaf pair for type info in Gallia



```
case class TypeLeaf(
      fullName        : FullyQualifiedName,

      dataClass       : Boolean = false, //
      galliaEnumValue: Boolean = false,
      bytes           : Boolean = false, //
      inheritsSeq     : Boolean = false,

      enumeratumValueNamesOpt: Option[Seq[S

      fields: Seq[Field] = Nil) {
```

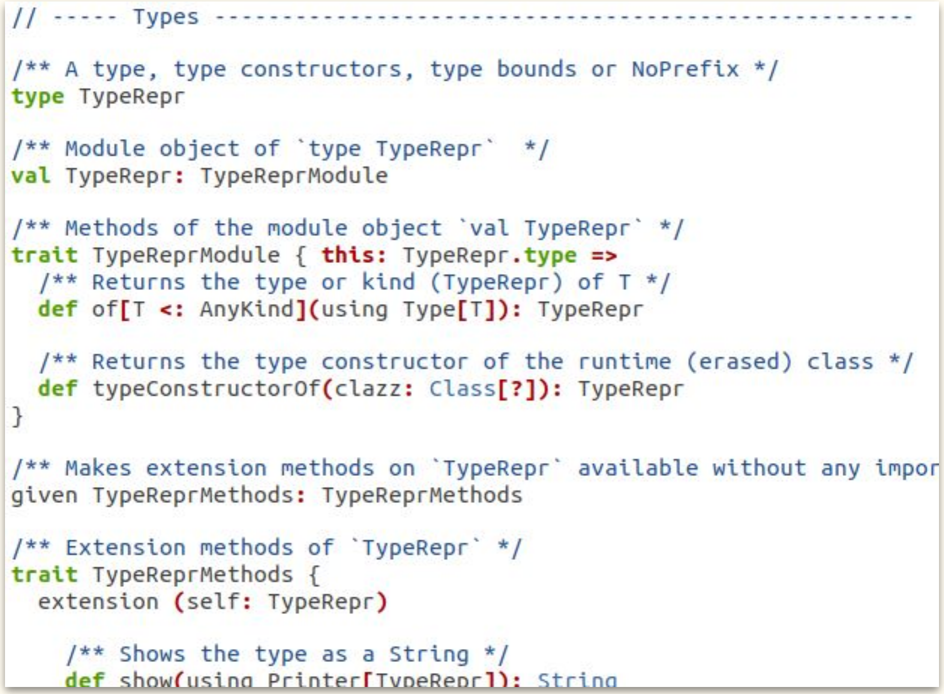
Macros in scala 3.x

- Harder to get started with than 2.x's
- Documentation can be confusing,

  → especially coming from 2.x macros
- No quasiquotes?
- Gotta love `Quotes.scala`

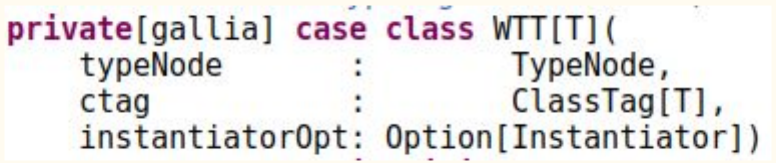
  ======================

  - "Feels" much sturdier
  - Same compilation unit :)



```
// ----- Types -------------------------------------------------------

/** A type, type constructors, type bounds or NoPrefix */
type TypeRepr

/** Module object of `type TypeRepr`  */
val TypeRepr: TypeReprModule

/** Methods of the module object `val TypeRepr` */
trait TypeReprModule { this: TypeRepr.type =>
  /** Returns the type or kind (TypeRepr) of T */
  def of[T <: AnyKind](using Type[T]): TypeRepr

  /** Returns the type constructor of the runtime (erased) class */
  def typeConstructorOf(clazz: Class[?]): TypeRepr
}

/** Makes extension methods on `TypeRepr` available without any impor
given TypeReprMethods: TypeReprMethods

/** Extension methods of `TypeRepr` */
trait TypeReprMethods {
  extension (self: TypeRepr)

    /** Shows the type as a String */
    def show(using Printer[TypeRepr]): String
```

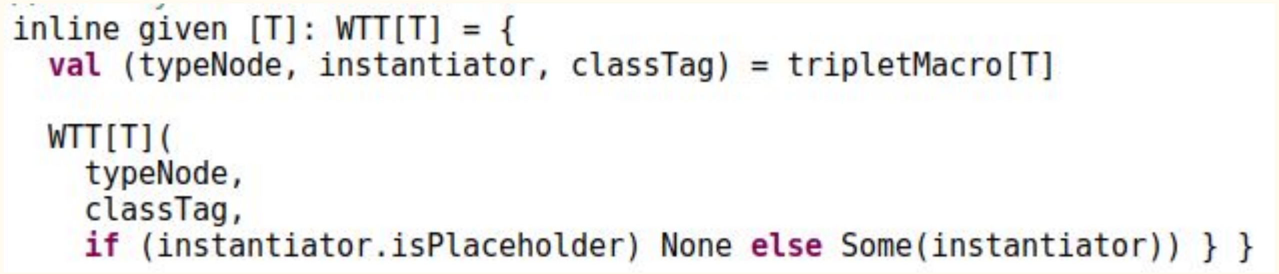
New WTT construct



```
private[gallia] case class WTT[T](
    typeNode        :         TypeNode,
    ctag            :         ClassTag[T],
    instantiatorOpt: Option[Instantiator])
```

- "WTT" was my alias **WeakTypeTag** with Scala 2

  → now has a life of its own in Scala 3.x



```
inline given [T]: WTT[T] = {
  val (typeNode, instantiator, classTag) = tripletMacro[T]

  WTT[T](
    typeNode,
    classTag,
    if (instantiator.isPlaceholder) None else Some(instantiator)) } }
```

Lessons

- Migrate to **2.13** if not the case yet
- Clean up/**refactor** your code (I refactored gallia-reflect first)
- Use **-no-indent** flag, at least at first
- Might want to play with compiler options **-rewrite** and **-Xsource:3**
- Address the **simple errors first** (adding ()s, adding explicit types, ...)
- If still maintaining 2.x: abuse SBT's /**scala-{2,3}/** folder convention at first

How to try Gallia with Scala 3

Published binaries for Scala 3.3.1 on Monday!

```
mkdir /tmp/scalaio24 && cd /tmp/scalaio24

echo -e 'scalaVersion := "3.3.1"\nlibraryDependencies +=
  "io.github.galliaproject" %% "gallia-core" % "0.6.1"' \
    > build.sbt

sbt console
-------------------------------------------------------
scala> import gallia.*
scala> """{"name": "anthony", "age": 39}"""
          .read() // thanks to import
            .increment("age")
          .printJson() // or e.g. .printRow()
{"name": "anthony", "age": 40}
```

Conclusion

- Mostly a **positive experience** for me in the end!

  → that really wasn't a given, so **kudos** to the Scala team :)

- As successful a new major version as can be

  → minus **significant indentation** part (*-no-indent* flag notwithstanding)

  → SBT's **/scala-{2,3}/** folder convention was a boon

  → 2.13 ⇔ 3.x **interoperability** was a brilliant idea

- Migrating **your codebase** may not be too painful an experience,
  at least if you're not relying on something like **reflect.runtime** or **2.x macros**

- But your **mileage may vary** based on what your codebase is about (lots of **deps**?)

Future Direction

- Will use Scala 3 enums instead of *enumeratum*
- Will finish porting macros from gallia-macros (read/write for case classes)
- Will eventually drop support for 2.12 (circa 2047 or 2048)
- Will generalize my WTT abstraction and make it a standalone library
- OSWO optimization!

Future Direction (OSWO optimization)

→ stands for "**O**n **S**teroid **W**ith **O**verhead"

(https://github.com/galliaproject/gallia-docs/blob/master/**oswo.md**)

Nutshell:

0. **Optimize** data run DAG
1. Generate **source code** for case classes+transformations
2. Runtime-**compiles** it (hence overhead)
3. **Run** it all (typically on Spark)

See prototype (OswoPrototype.scala)

G·A·L·L·I·A
PROJECT

Thank you!

Scalaio

—

Questions?