Code Review Gems

Richard Dallaway, @d6y

underscore

Agenda

The External Review

Three Code Examples

Problem → Solution → Key Points

Limits of Review

External Code Review

...can take many forms
...can be DIY

Focus on Two Questions

Are we producing good Scala?

...making the most of what Scala offers?

Is this code maintainable?

Challenges

Code	Maintenance
Learning the language	Explaining the problem
Framework experience	Explaining the thinking
Exploring the ecosystem	Readable tests

— Example 1 —

Option is not always the right option

problem.scala — gem

problem.scala | solution.scala

```scala
package example1

class Settings() {
  val dbUser: Option[String] = read("db.user")
}

object Problem extends App {

  def doTheWork(conf: Settings): Unit = {



    // Actual work to be done would go here
  }

  doTheWork(new Settings())
}
```

Line 22, Column 2 | 0 misspelled words | Spaces: 2 | Scala

problem.scala — gem

problem.scala | solution.scala

```scala
package example1

class Settings() {
  val dbUser: Option[String] = read("db.user")
}

object Problem extends App {

  def doTheWork(conf: Settings): Unit = {

    val user = try {
      conf.dbUser.get
    } catch {
      case nse: NoSuchElementException =>
        throw new IllegalArgumentException("Missing user setting")
    }

    // Actual work to be done would go here
  }

  doTheWork(new Settings())
}
```

Line 22, Column 2 | 0 misspelled words | Spaces: 2 | Scala

problem.scala — gem

problem.scala | solution.scala

```scala
package example1

class Settings() {
  val dbUser: Option[String] = read("db.user")
}

object Problem extends App {

  def doTheWork(conf: Settings): Unit = {

    val user = try {
      conf.dbUser.get
    } catch {
      case nse: NoSuchElementException =>
        throw new IllegalArgumentException("Missing user setting")
    }

    // Actual work to be done would go here
  }

  doTheWork(new Settings())
}
```

⇒ Unit

Line 22, Column 2 | 0 misspelled words | Spaces: 2 | Scala

problem_refeactor.scala — gems

problem_refeactor.scala

```scala
package example1

class Settings() {
  val dbUser: Option[String] = read("db.user")
}

object Problem extends App {

  def doTheWork(dbUser: String): Unit = {



    // Actual work to be done would go here



}

 doTheWork( ? )
}
```

Moving the problem

Line 22, Column 2 | 0 misspelled words | Spaces: 2 | Scala

solution.scala — gem

problem.scala | solution.scala

```scala
package example1

object Solution extends App {

  import com.typesafe.config.{Config,ConfigFactory}

  def doWork(conf: Settings): Unit = {
    println(s"Using ${conf.dbUser}")
  }

  doWork(settings)
}
```

Line 23, Column 2 | 4 misspelled words | Spaces: 2 | Scala

solution.scala — gem

problem.scala | solution.scala

```scala
package example1

object Solution extends App {

  import com.typesafe.config.{Config,ConfigFactory}

  // Fail fast by using strict vals to lookup configuration values:
  class Settings(conf: Config) {
    val dbUser: String = conf.getString("db.user")
  }

  val settings = new Settings(ConfigFactory.load())

  // Without a configuration value...
  // com.typesafe.config.ConfigException$Missing:
  //   No configuration setting found for key 'db'

  def doWork(conf: Settings): Unit = {
    println(s"Using ${conf.dbUser}")
  }

  doWork(settings)
}
```

Line 23, Column 2 | 4 misspelled words | Spaces: 2 | Scala

Observations

Make time to discover what’s already out there

…and adapt if it doesn’t quite fit.

Resources

GitHub, Inc. github.com/lauris/awe

README.md

Awesome Scala

A community driven list of useful Scala libraries, frameworks and software. This is not a catalog of all the libraries, just a starting point for your explorations. Inspired by awesome-python. Other amazingly awesome lists can be found in the awesome-awesomeness list.

- Awesome Scala
  - Database
  - Web Frameworks
  - i18n
  - Authentication
  - Testing
  - JSON Manipulation
  - Serialization
  - Science and Data Analysis
  - Big Data
  - Functional Reactive Programming
  - Modularization and Dependency Injection

Observations

Make time to discover what's already out there

…and adapt if it doesn't quite fit.

Unit is the end of the line

Good Scala?

Something the team arrives at

Based on the broader community

Changes over time

— Example 2 —

Stringly typed

problem.scala — gems

problem.scala

```scala
package example2

object PaymentStatus {
  val NOT_STARTED = "n/a"
  val PENDING     = "pending"
  val COMPLETE    = "complete"
  val REJECTED    = "rejected"
}

object Problem extends App {



}
```

Line 21, Column 2 | 0 misspelled words | Spaces: 2 | Scala

problem.scala — gems

problem.scala

```scala
package example2

object PaymentStatus {
  val NOT_STARTED = "n/a"
  val PENDING     = "pending"
  val COMPLETE    = "complete"
  val REJECTED    = "rejected"
}

object Problem extends App {

  def isFinished(status: String): Boolean =
    status == PaymentStatus.COMPLETE ||
    status == PaymentStatus.REJECTED



}
```

Line 21, Column 2 | 0 misspelled words | Spaces: 2 | Scala

problem.scala — gems

problem.scala

```scala
package example2

object PaymentStatus {
  val NOT_STARTED = "n/a"
  val PENDING     = "pending"
  val COMPLETE    = "complete"
  val REJECTED    = "rejected"
}

object Problem extends App {

  def isFinished(status: String): Boolean =
    status == PaymentStatus.COMPLETE ||
    status == PaymentStatus.REJECTED

  // Run-time match error:
  "uh-oh" match {
    case PaymentStatus.NOT_STARTED | PaymentStatus.PENDING  => false
    case PaymentStatus.COMPLETE    | PaymentStatus.REJECTED => true
  }
}
```

Line 21, Column 2 | 0 misspelled words | Spaces: 2 | Scala

“Enums fill a different niche: essentially as efficient as integer constants but safer and more convenient to define and to use.”

– Martin

solution_enum.scala — gems

problem.scala | solution_enum.scala

```scala
package example2

object StatusEnum extends Enumeration {
  type Status    = Value

  val NotStarted = Value("n/a")
  val Pending    = Value("pending")
  val Complete   = Value("complete")
  val Rejected   = Value("rejected")
}

object EnumSolution extends App {



}
```

Line 24, Column 2 | 0 misspelled words | Spaces: 2 | Scala

solution_enum.scala — gems

problem.scala | solution_enum.scala

```scala
package example2

object StatusEnum extends Enumeration {
  type Status    = Value

  val NotStarted = Value("n/a")
  val Pending    = Value("pending")
  val Complete   = Value("complete")
  val Rejected   = Value("rejected")
}

object EnumSolution extends App {

  import StatusEnum._

  def isFinished(status: Status): Boolean =
    status == Complete || status == Rejected

  def isInProgress(status: Status): Boolean =
    status match {
      case Pending              => true
      case Complete | Rejected => false
    }
}
```

Line 24, Column 2 | 0 misspelled words | Spaces: 2 | Scala

Enumerations

😃	Automatic integer identifier
😃	Automatic string name
😃	Values are ordered
😃	Automatic iterator
😃	Light (not one class per value)
☹️	Same type after erasure
☹️	No non-exhaustive match warning

solution_case.scala — gems

problem.scala | solution_enum.scala | solution_case.scala

```scala
package example2

object Status {

  sealed abstract class Value(val name: String)

  case object NotStarted extends Value("n/a")
  case object Pending    extends Value("pending")
  case object Complete   extends Value("complete")
  case object Rejected   extends Value("rejected")

  val values = Seq(NotStarted, Pending, Complete, Rejected)
}

object Solution extends App {


}
```

Line 31, Column 2 | 0 misspelled words | Spaces: 2 | Scala

solution_case.scala — gems

problem.scala | solution_enum.scala | solution_case.scala

```scala
package example2

object Status {

  sealed abstract class Value(val name: String)

  case object NotStarted extends Value("n/a")
  case object Pending    extends Value("pending")
  case object Complete   extends Value("complete")
  case object Rejected   extends Value("rejected")

  val values = Seq(NotStarted, Pending, Complete, Rejected)
}

object Solution extends App {

  import Status._

  def isFinished(status: Status.Value): Boolean =
    status == Complete || status == Rejected

  // Compile error:  match may not be exhaustive
  // It would fail on the following input: NotStarted

  def isInProgress(status: Status.Value): Boolean =
    status match {
      case Pending              => true
      case Complete | Rejected => false
    }

}
```

Line 31, Column 2 | 0 misspelled words | Spaces: 2 | Scala

But I have `VARCHAR`
not types...

solution_db.scala — gems

problem.scala | solution_enum.scala | solution_case.scala | solution_db.scala

```scala
package example2

object DbSolution extends App {

  import Status._




  case class Payment(status: Value, id: Long=0L)

  class Payments(tag: Tag) extends Table[Payment](tag, "payments") {
    def id     = column[Long]("id", O.PrimaryKey, O.AutoInc)
    def status = column[Value]("status")
    def *      = (status, id) <> (Payment.tupled, Payment.unapply)
  }

  lazy val payments = TableQuery[Payments]

  def pending(implicit s: Session) =
    payments.filter(_.status === (Pending : Value)).list
}
```

Line 25, Column 2 | 0 misspelled words | Spaces: 2 | Scala

solution_db.scala — gems

problem.scala | solution_enum.scala | solution_case.scala | solution_db.scala

```scala
package example2

object DbSolution extends App {

  import Status._

  implicit val statusColumnType =
    MappedColumnType.base[Value, String](
      _.name,
      s => Status.values.find(_.name == s) getOrElse NotStarted
    )

  case class Payment(status: Value, id: Long=0L)

  class Payments(tag: Tag) extends Table[Payment](tag, "payments") {
    def id     = column[Long]("id", O.PrimaryKey, O.AutoInc)
    def status = column[Value]("status")
    def *      = (status, id) <> (Payment.tupled, Payment.unapply)
  }

  lazy val payments = TableQuery[Payments]

  def pending(implicit s: Session) =
    payments.filter(_.status === (Pending : Value)).list
}
```

Line 25, Column 2 | 0 misspelled words | Spaces: 2 | Scala

Observations

Let the type system help you

Get data into types as soon as you can

“Scala Enumerations”
https://underscoreconsulting.com/blog/posts/2014/09/03/enumerations.html

— Example 3 —

Tests as docs

problem.scala — gems

problem.scala

```scala
package example3

import org.specs2.mutable._

class ProblemSpec extends Specification {

  "The processor" should {
    "detect a name change" in {



    }
  }
}
```

Line 1, Column 1 | 0 misspelled words | Spaces: 2 | Scala

problem.scala — gems

problem.scala

```scala
package example3

import org.specs2.mutable._

class ProblemSpec extends Specification {

  "The processor" should {
    "detect a name change" in {

      // If this was in the database...
      val applicant =
        Applicant(number="0000000002", surname="Smith", forenames=None)

      // ...and we received this change...
      val input = CsvRow(
        year      = "2014",
        code      = Code.Important,
        personId  = "000000001",
        number    = "0000000002",
        choice    = Choice.Third,
        surname   = Some("Smith"),
        forenames = Some("Alice")
        )

      // ...the action must be a name change:
      Processor.diff(applicant, input) must beSome(NameChange())

    }
  }
}
```

Line 1, Column 1 | 0 misspelled words | Spaces: 2 | Scala

solution.scala — gems

problem.scala | solution.scala

```scala
package example3

import org.specs2.mutable._

class SolutionSpec extends Specification {

  "The processor" should {
    "detect a name change" in {

      // If this was in the database...
      val applicant = TestApplicant() withNoForenames

      // ...and we received this change...
      val input = TestCsvRow() matching(applicant) withForenames "Alice"

      // ...the action must be a name change:
      Processor.diff(applicant.create, input.create) must beSome(NameChange())

    }
  }
}
```

Line 1, Column 1 | 0 misspelled words | Spaces: 2 | Scala

Builder with fixed data...

fixed_builders.scala — gems

problem.scala | solution.scala | fixed_builders.scala

```scala
package example3

case class TestApplicant(
  number:    String="002",
  surname:   String="Smith",
  forenames: Option[String]=Some("Colin")) {

  def withNoForenames: TestApplicant = copy(forenames=None)

  def create: Applicant = Applicant(number, surname, forenames)
}



}
```

Line 25, Column 2 | 0 misspelled words | Spaces: 2 | Scala

fixed_builders.scala — gems

problem.scala | solution.scala | fixed_builders.scala

```scala
package example3

case class TestApplicant(
  number:    String="002",
  surname:   String="Smith",
  forenames: Option[String]=Some("Colin")) {

  def withNoForenames: TestApplicant = copy(forenames=None)

  def create: Applicant = Applicant(number, surname, forenames)
}

case class TestCsvRow(
  number:    String="000",
  forenames: Option[String]=Some("Diana")) {

  def matching(a: TestApplicant): TestCsvRow = copy(number=a.number)
  def withForenames(fn: String): TestCsvRow = copy(forenames=Some(fn))

  def create: CsvRow =
    CsvRow("2014", Code.Important, "001",
      number,
      Choice.Third, Some("Smith"),
      forenames)
}
```

Line 25, Column 2 | 0 misspelled words | Spaces: 2 | Scala

Alternative with random test data...

random_builders.scala — gems

problem.scala | solution.scala | fixed_builders.scala | random_builders.scala

```scala
package example3

import org.scalacheck._, Arbitrary._, Gen._

object Builders {

  lazy val applicantGenerator: Gen[Applicant] =
    for {
      number    <- tenRandomDigits
      surname   <- arbitrary[String]
      forenames <- arbitrary[Option[String]]
    } yield Applicant(number, surname, forenames)

  val tenRandomDigits =
    listOfN(10, Gen.numChar).map(_.mkString)




  // Same pattern for random CSV Row
  // ...
}
```

Line 30, Column 1 | 0 misspelled words | Spaces: 2 | Scala

random_builders.scala — gems

problem.scala | solution.scala | fixed_builders.scala | random_builders.scala

```scala
package example3

import org.scalacheck._, Arbitrary._, Gen._

object Builders {

  lazy val applicantGenerator: Gen[Applicant] =
    for {
      number    <- tenRandomDigits
      surname   <- arbitrary[String]
      forenames <- arbitrary[Option[String]]
    } yield Applicant(number, surname, forenames)

  val tenRandomDigits =
    listOfN(10, Gen.numChar).map(_.mkString)

// Same pattern for random CSV Row
// ...
}
```

```
Applicant(8698360245,,None)

Applicant(0327349935,ᵊ祜쨔,None)

Applicant(5652792232,㉤襍🇼템ᵊ稙勺泳侊晉i儺怠♲蜦▩|||躪憁稃
珺苶✋鄒🇼뤋椙巖휆①🇯↓맫ᵢ¢🇯璒🇼鋼🇼🇼ⓞ·퓧🇼ᵘ🇼倜ᘓ藜뢸🇼铜,
Some(吾枃ᵊc膟顔))
```

Line 30, Column 1 | 0 misspelled words | Spaces: 2 | Scala

random_builders.scala — gems

problem.scala | solution.scala | fixed_builders.scala | random_builders.scala

```scala
package example3

import org.scalacheck._, Arbitrary._, Gen._

object Builders {

  lazy val applicantGenerator: Gen[Applicant] =
    for {
      number    <- tenRandomDigits
      surname   <- arbitrary[String]
      forenames <- arbitrary[Option[String]]
    } yield Applicant(number, surname, forenames)

  val tenRandomDigits =
    listOfN(10, Gen.numChar).map(_.mkString)




  // Same pattern for random CSV Row
  // ...
}
```

Line 30, Column 1 | 0 misspelled words | Spaces: 2 | Scala

random_builders.scala — gems

problem.scala | solution.scala | fixed_builders.scala | random_builders.scala

```scala
package example3

import org.scalacheck._, Arbitrary._, Gen._

object Builders {

  lazy val applicantGenerator: Gen[Applicant] =
    for {
      number    <- tenRandomDigits
      surname   <- arbitrary[String]
      forenames <- arbitrary[Option[String]]
    } yield Applicant(number, surname, forenames)

  val tenRandomDigits =
    listOfN(10, Gen.numChar).map(_.mkString)

  implicit class ApplicantBuider(app: Applicant) {
    def withNoForenames: Applicant = app.copy(forenames=None)
  }

  implicit val applicant: Arbitrary[Applicant] = Arbitrary(applicantGenerator)

  // Same pattern for random CSV Row
  // ...
}
```

Line 30, Column 1 | 0 misspelled words | Spaces: 2 | Scala

solution_random.scala — gems

problem.scala | solution.scala | fixed_builders.scala | random_builders.scala | solution_random.scala

```scala
package example3

import org.specs2.mutable._
import org.specs2.ScalaCheck

class RandomSolutionSpec extends Specification with ScalaCheck {

  import Builders._

  "The processor" should {
    "detect a name change" in prop {
      (applicant: Applicant, input: CsvRow) =>

      Processor.diff(
        applicant withNoForenames,
        input matching(applicant) withForenames "Alice") must beSome(NameChange())
    }
  }
}
```

Line 19, Column 2 | 0 misspelled words | Spaces: 2 | Scala

Observations

You can still use what you know

Make use of an expressive syntax

...and neat libraries

Don’t settle for crappy tests

Challenges

Code	Maintenance
Learning the language	Explaining the problem
Framework experience	Explaining the thinking
Exploring the ecosystem	Readable tests

“You cannot inspect quality into a product.”

–Harold S. Dodge

Summary

Learn about the wider Scala ecosystem

Don’t leave it until “the review”

Thanks!

Richard Dallaway, @d6y

https://github.com/d6y/gems

underscore