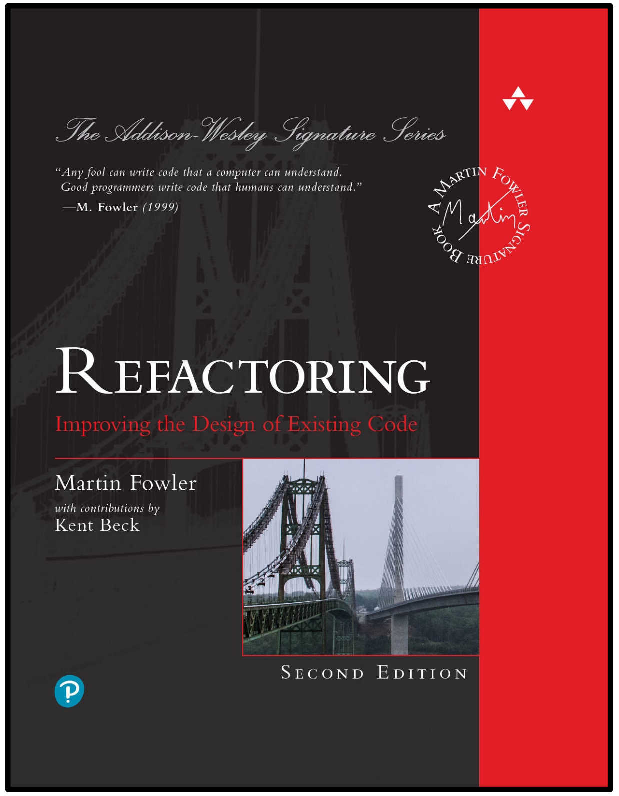
Scala and Java Side by Side
The Result of Martin Fowler's 1st Refactoring Example

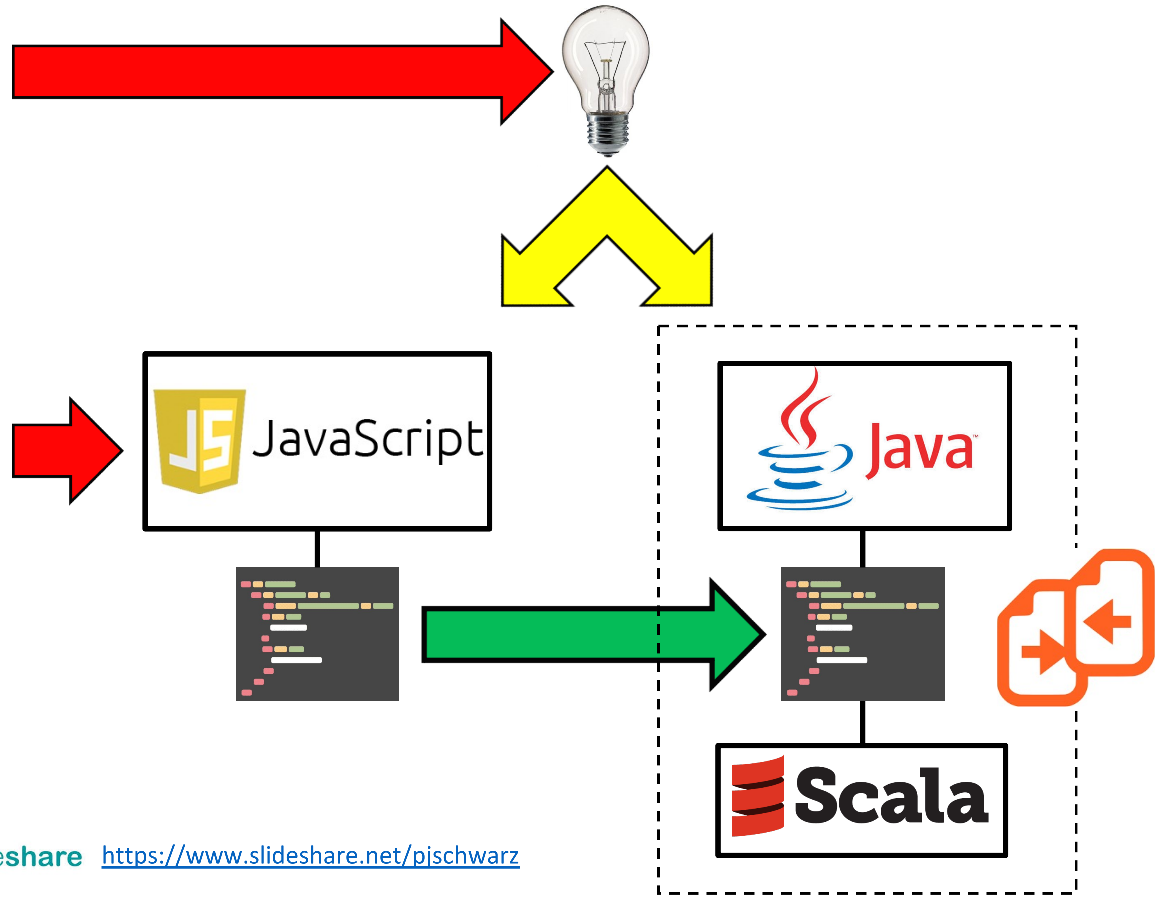
Java's **records**, **sealed interfaces** and **text blocks** are catching up with **Scala**'s **case classes**, **sealed traits** and **multiline strings**

Judge for yourself in this quick IDE-based **visual comparison**

of the **Scala** and **Java** translations of **Martin Fowler**'s refactored **Javascript** code

Martin Fowler
@martinfowler


The Addison-Wesley Signature Series
"Any fool can write code that a computer can understand. Good programmers write code that humans can understand."
—M. Fowler (1999)
A Martin Fowler Signature Book
REFACTORING
Improving the Design of Existing Code
Martin Fowler
with contributions by
Kent Beck
Second Edition



slides by @philip_schwarz slideshare https://www.slideshare.net/pjschwarz

@philip_schwarz

Java is in a constant process of catching up with some of the features found in other languages.

With this visual comparison of the Java and Scala translations of the refactored Javascript code from the 1st refactoring example of Martin Fowler's famous book, you can get some rapid and concrete evidence of some of the effects that Java's evolution is having on programming in that language.

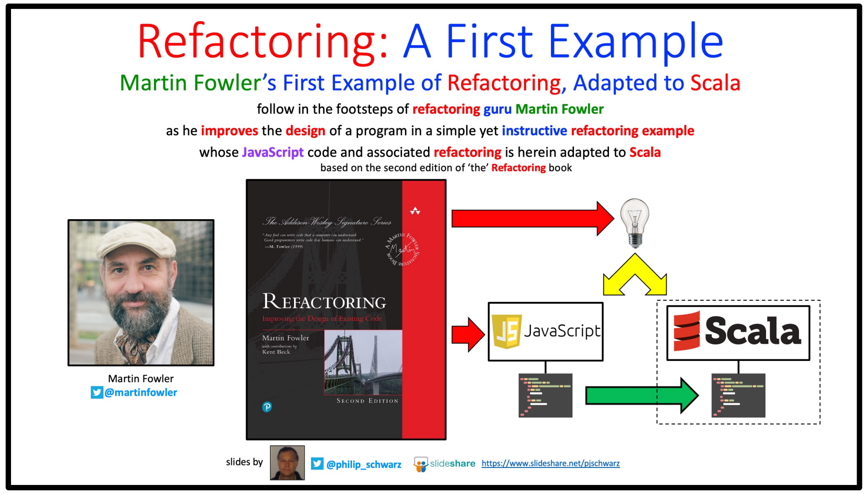
The code provides you with a simple example of the following Java features incorporated into long-term support (LTS) version JDK 17 (the previous LTS version being JDK 11):

- Text blocks (JDK 15)
- Records (JDK 16)
- Sealed interfaces (JDK 17)

If you want to know how the Java and Scala translations of the Javascript code came about, see the following pair of slide decks and repositories

Scala

Java



Refactoring: A First Example

Martin Fowler's First Example of Refactoring, Adapted to Scala

follow in the footsteps of refactoring guru Martin Fowler

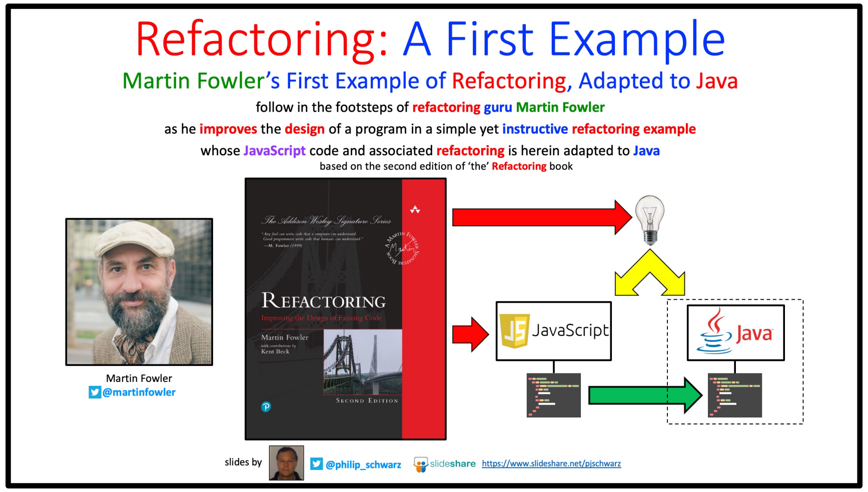
as he improves the design of a program in a simple yet instructive refactoring example

whose JavaScript code and associated refactoring is herein adapted to Scala

based on the second edition of 'the' Refactoring book

Martin Fowler
@martinfowler

slides by @philip_schwarz slideshare https://www.slideshare.net/pjschwarz



Refactoring: A First Example

Martin Fowler's First Example of Refactoring, Adapted to Java

follow in the footsteps of refactoring guru Martin Fowler

as he improves the design of a program in a simple yet instructive refactoring example

whose JavaScript code and associated refactoring is herein adapted to Java

based on the second edition of 'the' Refactoring book

Martin Fowler
@martinfowler

slides by @philip_schwarz slideshare https://www.slideshare.net/pjschwarz

https://www.slideshare.net/pjschwarz/refactoring-a-first-example-martin-fowlers-first-example-of-refactoring-adapted-to-(java|scala)

https://github.com/philipschwarz/refactoring-a-first-example-(java|scala)

```scala
@main def main(): Unit =

  assert(
    statement(invoices(0), plays)
    ==
    """|Statement for BigCo
       |  Hamlet: $650.00 (55 seats)
       |  As You Like It: $580.00 (35 seats)
       |  Othello: $500.00 (40 seats)
       |Amount owed is $1,730.00
       |You earned 47 credits
       |""".stripMargin
  )

  assert(
    htmlStatement(invoices(0), plays)
    ==
    """|<h1>Statement for BigCo</h1>
       |<table>
       |<tr><th>play</th><th>seats</th><th>cost</th></tr>
       |<tr><td>Hamlet</td><td>55</td><td>$650.00</td></tr>
       |<tr><td>As You Like It</td><td>35</td><td>$580.00</td></tr>
       |<tr><td>Othello</td><td>40</td><td>$500.00</td></tr>
       |</table>
       |<p>Amount owed is <em>$1,730.00</em></p>
       |<p>You earned <em>47</em> credits</p>
       |""".stripMargin
  )
```

Scala

```java
public static void main(String[] args) {

   if (!Statement.statement(invoices.get(0), plays).equals(
     """
     Statement for BigCo
       Hamlet: $650.00 (55 seats)
       As You Like It: $580.00 (35 seats)
       Othello: $500.00 (40 seats)
     Amount owed is $1,730.00
     You earned 47 credits
     """
   )) throw new AssertionError();

   if (!Statement.htmlStatement(invoices.get(0), plays).equals(
     """
     <h1>Statement for BigCo</h1>
     <table>
     <tr><th>play</th><th>seats</th><th>cost</th></tr>
     <tr><td>Hamlet</td><td>55</td><td>$650.00</td></tr>
     <tr><td>As You Like It</td><td>35</td><td>$580.00</td></tr>
     <tr><td>Othello</td><td>40</td><td>$500.00</td></tr>
     </table>
     <p>Amount owed is <em>$1,730.00</em></p>
     <p>You earned <em>47</em> credits</p>
     """
   )) throw new AssertionError();

 }
```

Java

```scala
val invoices: List[Invoice] = List(
  Invoice( customer = "BigCo",
           performances = List(Performance(playID = "hamlet",  audience = 55),
                               Performance(playID = "as-like", audience = 35 ,
                               Performance(playID = "othello", audience = 40)))
)

val plays = Map (
  "hamlet"  -> Play(name = "Hamlet", `type` = "tragedy"),
  "as-like" -> Play(name = "As You Like It", `type` = "comedy"),
  "othello" -> Play(name = "Othello", `type` = "tragedy")
)
```

Scala

```java
static final List<Invoice> invoices =
  List.of(
    new Invoice(
      "BigCo",
      List.of(new Performance( "hamlet", 55),
              new Performance("as-like", 35),
              new Performance("othello", 40))));

static final Map<String,Play> plays = Map.of(
  "hamlet" , new Play("Hamlet", "tragedy"),
  "as-like", new Play("As You Like It", "comedy"),
  "othello", new Play("Othello", "tragedy"));
```

Java

```
Statement for BigCo
  Hamlet: $650.00 (55 seats)
  As You Like It: $580.00 (35 seats)
  Othello: $500.00 (40 seats)
Amount owed is $1,730.00
You earned 47 credits
```

```scala
case class Play(
  name: String,
  `type`: String
)

case class Performance(
  playID: String,
  audience: Int
)

case class EnrichedPerformance(
  playID: String,
  play: Play,
  audience: Int,
  amount: Int,
  volumeCredits: Int
)

case class Invoice(
  customer: String,
  performances: List[Performance]
)

case class StatementData(
  customer: String,
  performances: List[EnrichedPerformance],
  totalAmount: Int,
  totalVolumeCredits: Int
)
```

```
Statement for BigCo
  Hamlet: $650.00 (55 seats)
  As You Like It: $580.00 (35 seats)
  Othello: $500.00 (40 seats)
Amount owed is $1,730.00
You earned 47 credits
```

```java
record Play(
  String name,
  String type
) { }

record Performance(
  String playID,
  int audience
) { }

record EnrichedPerformance(
  String playID,
  Play play,
  int audience,
  int amount,
  int volumeCredits
) { }

record Invoice(
  String customer,
  List<Performance> performances
) { }

record StatementData(
  String customer,
  List<EnrichedPerformance> performances,
  int totalAmount,
  int totalVolumeCredits,
) { }
```

```scala
def renderPlainText(data: StatementData): String =
  s"Statement for ${data.customer}\n" + (
    for
      perf <- data.performances
    yield s"  ${perf.play.name}: ${usd(perf.amount/100)} (${perf.audience} seats)\n"
  ).mkString +
  s"""|Amount owed is ${usd(data.totalAmount/100)}
      |You earned ${data.totalVolumeCredits} credits
      |""".stripMargin

def renderHtml(data: StatementData): String =
  s"""|<h1>Statement for ${data.customer}</h1>
      |<table>
      |<tr><th>play</th><th>seats</th><th>cost</th></tr>
      |""".stripMargin + (
    for
      perf <- data.performances
    yield  s"<tr><td>${perf.play.name}</td><td>${perf.audience}</td>" +
           s"<td>${usd(perf.amount/100)}</td></tr>\n"
  ).mkString +
  s"""|</table>
      |<p>Amount owed is <em>${usd(data.totalAmount/100)}</em></p>
      |<p>You earned <em>${data.totalVolumeCredits}</em> credits</p>
      |""".stripMargin
```

```
Statement for BigCo
  Hamlet: $650.00 (55 seats)
  As You Like It: $580.00 (35 seats)
  Othello: $500.00 (40 seats)
Amount owed is $1,730.00
You earned 47 credits
```

```
<h1>Statement for BigCo</h1>
<table>
<tr><th>play</th><th>seats</th><th>cost</th></tr>
<tr><td>Hamlet</td><td>55</td><td>$650.00</td></tr>
<tr><td>As You Like It</td><td>35</td><td>$580.00</td></tr>
<tr><td>Othello</td><td>40</td><td>$500.00</td></tr>
</table>
<p>Amount owed is <em>$1,730.00</em></p>
<p>You earned <em>47</em> credits</p>
```

Scala

```java
static String renderPlainText(StatementData data) {
 return
  "Statement for %s\n".formatted(data.customer()) +
   data.performances()
     .stream()
     .map(p ->
       "  %s: %s (%d seats)\n”
       .formatted(p.play().name(), usd(p.amount()/100), p.audience())
     ).collect(Collectors.joining()) +
   """
   Amount owed is %s
   You earned %d credits
   """.formatted(usd(data.totalAmount()/100), data.totalVolumeCredits());
}

static String renderHtml(StatementData data) {
 return
  """
  <h1>Statement for %s</h1>
  <table>
  <tr><th>play</th><th>seats</th><th>cost</th></tr>
  """.formatted(data.customer()) +
   data
    .performances()
    .stream()
    .map(p -> "<tr><td>%s</td><td>%d</td><td>%s</td></tr>\n"
         .formatted(p.play().name(),p.audience(),usd(p.amount()/100))
    ).collect(Collectors.joining()) +
  """
  </table>
  <p>Amount owed is <em>%s</em></p>
  <p>You earned <em>%d</em> credits</p>
  """.formatted(usd(data.totalAmount()/100), data.totalVolumeCredits());
}
```

Java

```scala
def statement(invoice: Invoice, plays: Map[String, Play]): String =
  renderPlainText(createStatementData(invoice,plays))

def htmlStatement(invoice: Invoice, plays: Map[String, Play]): String =
  renderHtml(createStatementData(invoice,plays))

def usd(aNumber: Int): String =
  val formatter = NumberFormat.getCurrencyInstance(Locale.US)
  formatter.setCurrency(Currency.getInstance(Locale.US))
  formatter.format(aNumber)


…
```

Scala

```java
public class Statement {

  static String statement(Invoice invoice, Map<String, Play> plays) {
    return renderPlainText(CreateStatementData.createStatementData(invoice,plays));
  }

  static String htmlStatement(Invoice invoice, Map<String, Play> plays) {
    return renderHtml(CreateStatementData.createStatementData(invoice, plays));
  }

  static String usd(int aNumber) {
    final var formatter = NumberFormat.getCurrencyInstance(Locale.US);
    formatter.setCurrency(Currency.getInstance(Locale.US));
    return formatter.format(aNumber);
  }

…
```

Java

```
Statement for BigCo
  Hamlet: $650.00 (55 seats)
  As You Like It: $580.00 (35 seats)
  Othello: $500.00 (40 seats)
Amount owed is $1,730.00
You earned 47 credits
```

```html
<h1>Statement for BigCo</h1>
<table>
<tr><th>play</th><th>seats</th><th>cost</th></tr>
<tr><td>Hamlet</td><td>55</td><td>$650.00</td></tr>
<tr><td>As You Like It</td><td>35</td><td>$580.00</td></tr>
<tr><td>Othello</td><td>40</td><td>$500.00</td></tr>
</table>
<p>Amount owed is <em>$1,730.00</em></p>
<p>You earned <em>47</em> credits</p>
```

```scala
def createStatementData(invoice: Invoice, plays: Map[String,Play]): StatementData =

  def playFor(aPerformance: Performance): Play =
    plays(aPerformance.playID)

  def totalVolumeCredits(performances: List[EnrichedPerformance]): Int =
    performances.map(_.volumeCredits).sum

  def totalAmount(performances: List[EnrichedPerformance]): Int =
    performances.map(_.amount).sum

  …
```

Scala

```java
public class CreateStatementData {

  static StatementData createStatementData(Invoice invoice, Map<String, Play> plays) {

    Function<Performance, Play> playFor =
      aPerformance -> plays.get(aPerformance.playID());

    Function<List<EnrichedPerformance>, Integer> totalVolumeCredits = (performances) ->
      performances.stream().mapToInt(EnrichedPerformance::volumeCredits).sum();

    Function<List<EnrichedPerformance>, Integer> totalAmount = (performances) ->
      performances.stream().mapToInt(EnrichedPerformance::amount).sum();

    …
```

Java

```
def enrichPerformance(aPerformance: Performance): EnrichedPerformance =
  val calculator =
    PerformanceCalculator(aPerformance,playFor(aPerformance))
  EnrichedPerformance(
    aPerformance.playID,
    calculator.play,
    aPerformance.audience,
    calculator.amount,
    calculator.volumeCredits)

val enrichedPerformances =
  invoice.performances.map(enrichPerformance)
StatementData(
  invoice.customer,
  enrichedPerformances,
  totalAmount(enrichedPerformances),
  totalVolumeCredits(enrichedPerformances))
```

Scala

```
    Function<Performance, EnrichedPerformance> enrichPerformance = aPerformance -> {
      final var calculator =
        PerformanceCalculator.instance(aPerformance,playFor.apply(aPerformance));
      return new EnrichedPerformance(
        aPerformance.playID(),
        calculator.play(),
        aPerformance.audience(),
        calculator.amount(),
        calculator.volumeCredits());
    };

    final var enrichedPerformances =
      invoice.performances().stream().map(enrichPerformance).collect(toList());
    return new StatementData(
      invoice.customer(),
      enrichedPerformances,
      totalAmount.apply(enrichedPerformances),
      totalVolumeCredits.apply(enrichedPerformances));
  }

}
```

Java

```scala
sealed trait PerformanceCalculator:

  def performance: Performance
  def play: Play
  def amount: Int
  def volumeCredits: Int = math.max(performance.audience - 30, 0)

object PerformanceCalculator:
  def apply(aPerformance: Performance, aPlay: Play): PerformanceCalculator =
    aPlay.`type` match
      case "tragedy" => TragedyCalculator(aPerformance, aPlay)
      case "comedy" => ComedyCalculator(aPerformance, aPlay)
      case other => throw IllegalArgumentException(s"unknown type ${aPlay.`type`}")
…
```

```java
sealed interface PerformanceCalculator {

  Performance performance();
  Play play();
  int amount();
  default int volumeCredits() { return Math.max(performance().audience() - 30, 0); }

  static PerformanceCalculator instance(Performance aPerformance, Play aPlay) {
    return switch (aPlay.type()) {
      case "tragedy" -> new TragedyCalculator(aPerformance, aPlay);
      case "comedy" -> new ComedyCalculator(aPerformance, aPlay);
      default -> throw new IllegalArgumentException(String.format("unknown type '%s'", aPlay.type()));
    };
  }
}
…
```

```scala
case class TragedyCalculator(performance: Performance, play: Play) extends PerformanceCalculator:

  def amount: Int =
    val basicAmount = 40_000
    val largeAudiencePremiumAmount =
      if performance.audience <= 30 then 0 else 1_000 * (performance.audience - 30)
    basicAmount + largeAudiencePremiumAmount
…
```

Scala

```java
record TragedyCalculator(Performance performance, Play play) implements PerformanceCalculator {

  @Override public int amount() {
    final var basicAmount = 40_000;
    final var largeAudiencePremiumAmount =
      performance.audience() <= 30 ? 0 : 1_000 * (performance.audience() - 30);
    return basicAmount + largeAudiencePremiumAmount;
  }

}
…
```

Java

```scala
case class ComedyCalculator(performance: Performance, play: Play) extends PerformanceCalculator:

  def amount: Int =
    val basicAmount = 30_000
    val largeAudiencePremiumAmount =
      if performance.audience <= 20 then 0 else 10_000 + 500 * (performance.audience - 20)
    val audienceSizeAmount = 300 * performance.audience
    basicAmount + largeAudiencePremiumAmount + audienceSizeAmount

  override def volumeCredits: Int =
    super.volumeCredits + math.floor(performance.audience / 5).toInt
```

Scala

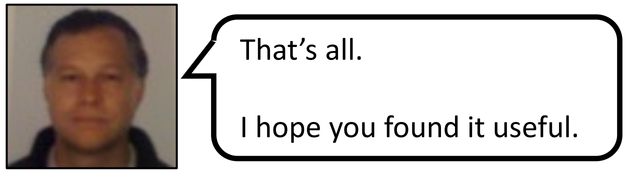
```java
record ComedyCalculator(Performance performance, Play play) implements PerformanceCalculator {

  @Override public int amount() {
    final var basicAmount = 30_000;
    final var largeAudiencePremiumAmount =
      performance.audience() <= 20 ? 0 : 10_000 + 500 * (performance.audience() - 20);
    final var audienceSizeAmount = 300 * performance.audience();
    return basicAmount + largeAudiencePremiumAmount + audienceSizeAmount;

  @Override public int volumeCredits() {
    return PerformanceCalculator.super.volumeCredits() + (int) Math.floor(performance().audience() / 5);
  }

}
```

Java


That’s all.

I hope you found it useful.