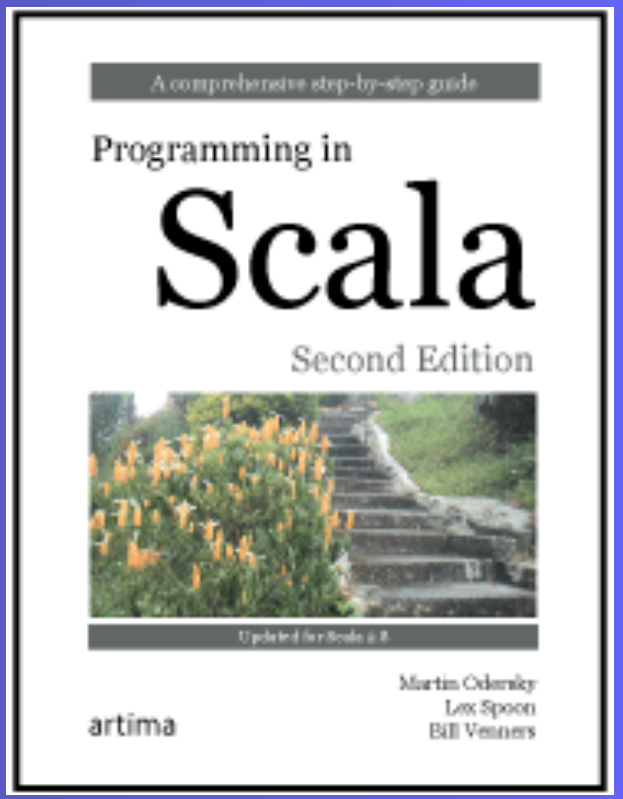
Stairway to Scala - Flight 7

Composition and inheritance

Bill Venners
Dick Wall

www.artima.com



Flight 5 goal

Learn Scala's way of doing composition and inheritance relationships between classes (extends, abstract classes, parameterless methods, overriding, parametric fields, invoking superclass constructors...)

Where we're going

```
elem(s: String): Element

val column1 = elem("hello") above elem("***")
val column2 = elem("***") above elem("world")
column1 beside column2

hello ***
 *** world
```

Abstract classes

```
abstract class Element {
  def contents: Array[String]
}
```

Defining parameterless methods

```
abstract class Element {
  def contents: Array[String]
  def height: Int = contents.length
  def width: Int = if (height == 0) 0 else contents(0).length
}
```

- Can call parameterless and empty-paren methods with or without parens
- Recommended: Use parens if side effects

Uniform access principle

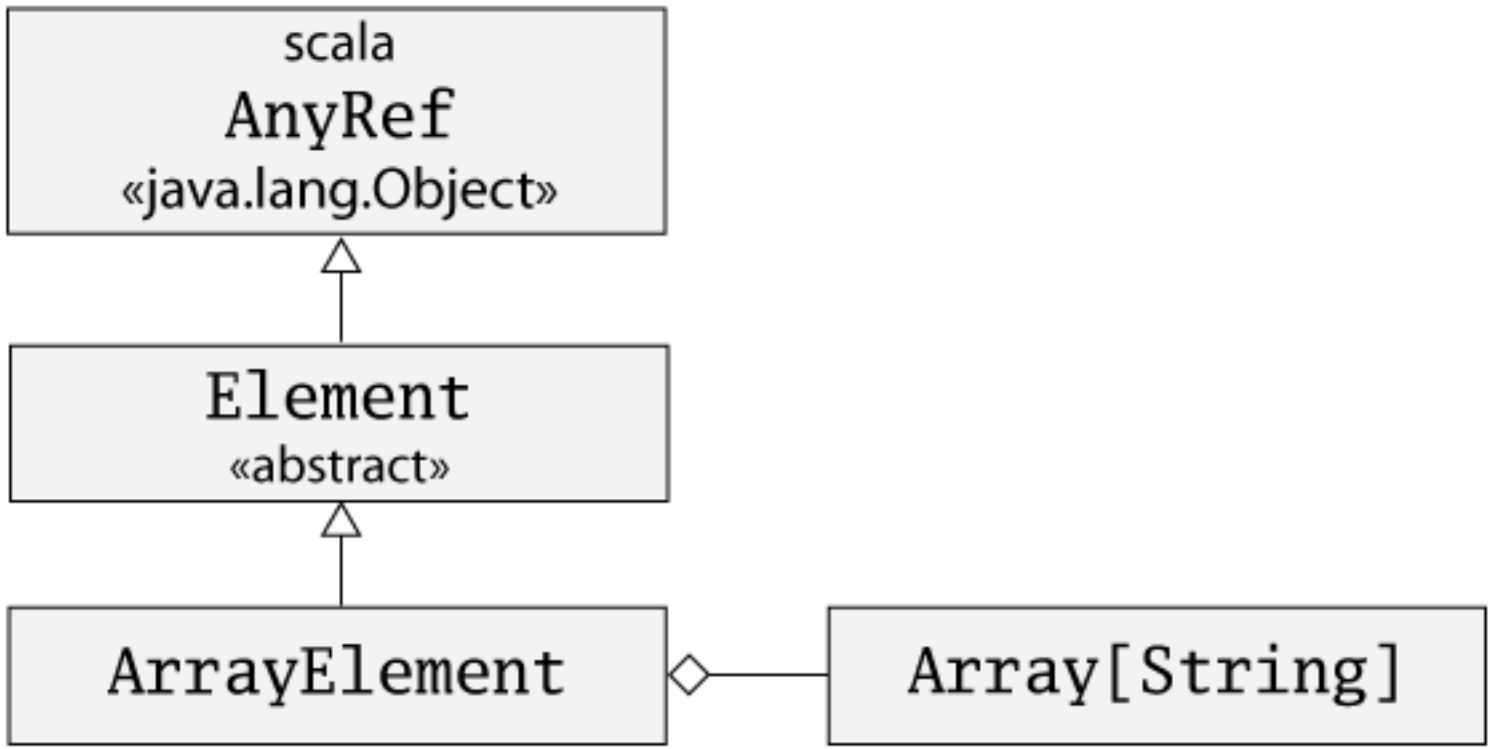
```
abstract class Element {
  def contents: Array[String]
  val height: Int = contents.length
  val width: Int = if (height == 0) 0 else contents(0).length
}
```

Extending classes

```
class ArrayElement(conts: Array[String]) extends Element {
  def contents: Array[String] = conts
}
```

- If you leave out extends, you extend scala.AnyRef (java.lang.Object)

ArrayElement class diagram



Using ArrayElement

```
scala> val ae = new ArrayElement(Array("hello", "world"))
ae: ArrayElement = ArrayElement@d94e60

scala> ae.width
res1: Int = 5

val e: Element = new ArrayElement(Array("hello"))
```

Overriding methods and fields

- Fields and methods belong to the same namespace in Scala (not so in Java)

```
class ArrayElement(conts: Array[String]) extends Element {
  val contents: Array[String] = conts
}
```

Defining parametric fields

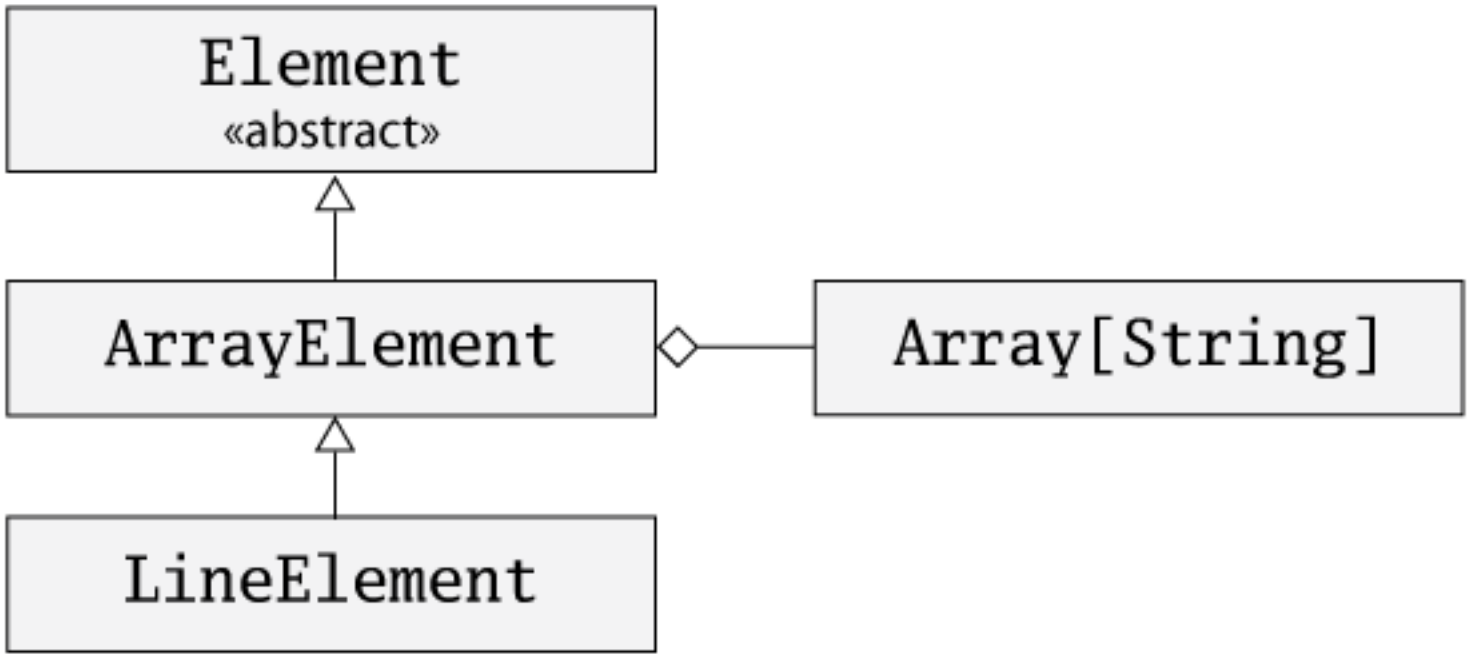
```
class ArrayElement(
  val contents: Array[String]
) extends Element
```

As if...

```
class ArrayElement(x123: Array[String]) extends Element {
  val contents: Array[String] = x123
}
```

Invoking superclass constructors

```
class LineElement(s: String) extends ArrayElement(Array(s)) {
  override def width = s.length
  override def height = 1
}
```



Using override modifiers

- helps prevent accidental overrides
- addresses, but doesn't solve, the "fragile base class problem"

Declaring final members

```
class ArrayElement extends Element {
  final override def demo() {
    println("ArrayElement's implementation invoked")
  }
}

final class ArrayElement extends Element {
  override def demo() {
    println("ArrayElement's implementation invoked")
  }
}
```

Added above, beside, and toString

```
abstract class Element {

  def contents: Array[String]

  def width: Int =
    if (height == 0) 0 else contents(0).length

  def height: Int = contents.length

  def above(that: Element): Element =
    new ArrayElement(this.contents ++ that.contents)

  def beside(that: Element): Element =
    new ArrayElement(
      for (
        (line1, line2) <- this.contents zip that.contents
      ) yield line1 + line2
    )

  override def toString = contents mkString "\n"
```

Factory methods in a companion object

```
object Element {

  def elem(contents: Array[String]): Element =
    new ArrayElement(contents)

  def elem(chr: Char, width: Int, height: Int): Element =
    new UniformElement(chr, width, height)

  def elem(line: String): Element =
    new LineElement(line)
}
```

Use the factory methods

```
import Element.elem

abstract class Element {

  def contents: Array[String]

  def width: Int = if (height == 0) 0 else contents(0).length

  def height: Int = contents.length

  def above(that: Element): Element = elem(this.contents ++ that.contents)

  def beside(that: Element): Element = elem(
    for ((line1, line2) <- this.contents zip that.contents) yield line1 + line2
  )

  override def toString = contents mkString "\n"
}
```

Hiding the implementation classes

```
object Element {

  private class ArrayElement(
    val contents: Array[String]
  ) extends Element

  private class LineElement(s: String) extends Element {
    val contents = Array(s)
    override def width = s.length
    override def height = 1
  }

  private class UniformElement(
    ch: Char,
    override val width: Int,
    override val height: Int
  ) extends Element {
    private val line = ch.toString * width
    def contents = Array.fill(height)(line)
```

Hiding the implementation classes

```
  def elem(contents:  Array[String]): Element =
    new ArrayElement(contents)

  def elem(chr: Char, width: Int, height: Int): Element =
    new UniformElement(chr, width, height)

  def elem(line: String): Element =
    new LineElement(line)
}
```

Exercises - Make a Train