10. 상속과 구성

부제: OOP in Scala

데이터인프라개발팀
조영일

목차

- 추상클래스
- 파라미터 없는 메소드 ⚠️
- 클래스 확장
  - 상속과 구성
- 메소드 및 필드 오버로드
- 파라미터 필드 ⚠️
- 슈퍼클래스 생성자 호출
- 다형성과 동적바인딩
- final
- 팩토리 객체와 팩토리 메소드 ⚠️

추상클래스

- Java의 abstract와 동일함

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

Element.scala

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

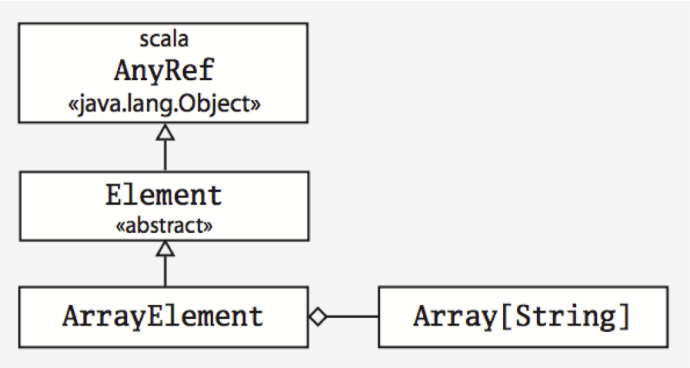
파라미터 없는 메소드 정의

- 단순 getter처럼 쓸 경우,
  - def로 정의하지 않고 val로 정의해도 됨

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

클래스 확장

- 상속
  - 서브클래스가 슈퍼클래스 중의 한 가지(is-a 관계)인 경우
- 구성(composition)
  - 다른 클래스를 참조/사용



ArrayElement.scala

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

메소드와 필드 오버라이드

- 파라미터 없는 메소드의 경우
  - def 대신에 val로 구현할 수 있음

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

파라미터 필드 정의

- 생성자로 넘기는 필드 초기값 처리를 좀 더 간단히
  - val을 이용하여 파라미터와 멤버 변수를 연결해줌
  - 생성자 body도 필요없어짐

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

슈퍼클래스의 생성자 호출

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

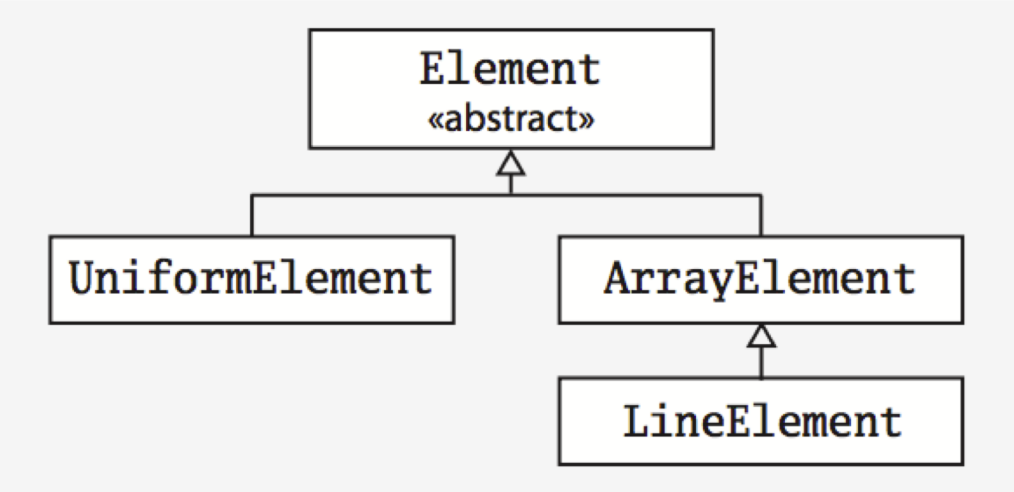
override 수식자 사용

- 부모 클래스의 구체적인 멤버를 오버라이드할 필요가 있을 때
- 추상 멤버에는 필요없음

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

다형성과 동적 바인딩

- Subtyping Polymorphism
  - ex) val Element = new ArrayElement(...)



다형성과 동적 바인딩

- 동적 바인딩
  - 다형성이 존재할 때 실제 타입을 결정하는 것은 동적으로

```
abstract class Element {
  def demo() {
    println("Element's implementation invoked")
  }
}

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

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

// UniformElement inherits Element's demo
class UniformElement extends Element
```

```
def invokeDemo(e: Element) {
  e.demo()
}

scala> invokeDemo(new ArrayElement)
ArrayElement's implementation invoked

scala> invokeDemo(new LineElement)
LineElement's implementation invoked

scala> invokeDemo(new UniformElement)
Element's implementation invoked
```

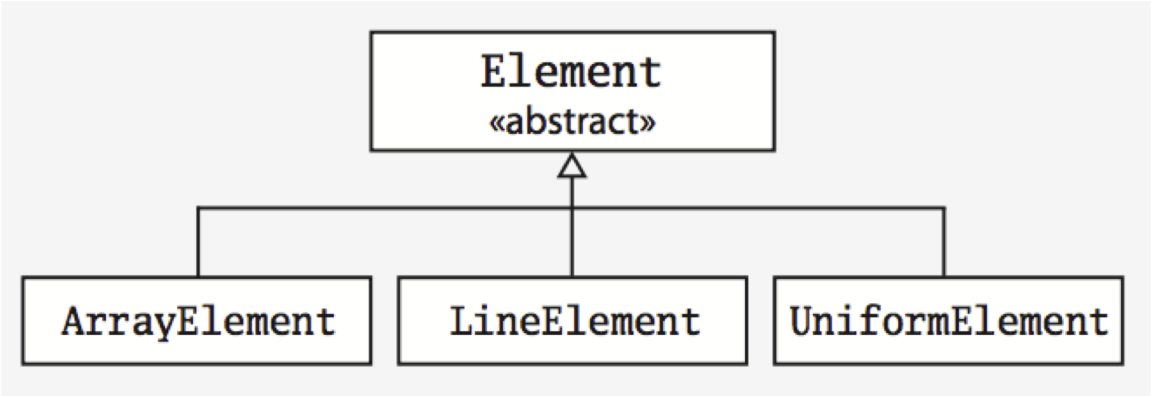
final 멤버 선언

```
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")
  }
}
```

상속과 구성 사용

- LineElement는 ArrayElement와 is-a 관계가 아님
- 그래서 Element를 직접 상속받는 것으로 변경



LineElement.scala

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

above, beside, toString 구현

- above: concatenate
- beside: side-by-side merge

```
def above(that: Element): Element =
  new ArrayElement(this.contents ++ that.contents)
def beside(that: Element): Element = {
  val contents = new Array[String](this.contents.length)
  for (i <- 0 until this.contents.length)
    contents(i) = this.contents(i) + that.contents(i)
  new ArrayElement(contents)
}
override def toString = contents mkString "\n"
```

```
new ArrayElement(
  for (
    (line1, line2) <- this.contents zip that.contents
  ) yield line1 + line2
)
```

팩토리 객체 정의

- Element 추상클래스
- ElementFactory 오브젝트
  - elem이라는 생성자
  - ArrayElement, LineElement, UniformElement는 private

```
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"
}
```

```
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)
  }

  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)
}
```

```scala
import ElementFactory.elem

object ElementTest {
  def main(args: Array[String]): Unit = {
    println(elem(Array("music")) above elem("pops!!!"))
    println(elem(Array("classics", "jazz")) beside
  elem(Array("rock", "punk", "metal")))
  }
}
```