Scala in Practice

Data classes

```
class User {
  private String name;
  private int age;
  . . .
  public User(String name, int age){
    this.name = name;
    this.age = age;
  }
  // more constructors
  public String getName(){
    return name;
  }
  public String setName(String name){
    name = this.name;
  }
  //more getters & setters
}
```

Case class structure

```
case class SomeName(. . .) {
  . . .
}
```

~~new~~ SomeName(...)

Auto-generation

- Auto-generates methods:
  - *apply*
  - *accessors* for all constructor parameters
  - *copy*
  - *equals* & *hashCode*
  - *toString*
  - *unapply*

Apply method

```
scala> case class Person(name: String, age: Int)
defined class Person

scala> val alice = Person("Alice", 35)
alice: Person = Person(Alice, 35)
```

Accessors methods

```
scala> alice.name
res0: String = Alice

scala> alice.name = "Bob"
<console>:10: error: reassignment to val
       alice.name = "Bob"
```

toString method

```
scala> val alice = Person("Alice", 35)
alice: Person = Person(Alice, 35)

scala > class Person(name: String, age: Int)
scala> val alice = new Person("Alice", 35)
alice: Person = Person@3830f918
```

Copy method

```
scala > case class Message(sender: String, body: String)

scala > msg1: Message = Message("Alice", "Hi")

scala > val msg2 = msg1.copy()

scala > val msg2 = msg1.copy(sender = "Bob")
```

Copy method

```
scala > case class Sender(name: String, id: String)
scala > case class Message(sender: Sender, body: String)

scala > val msg1 = Message(Sender("Alice", "al"), "Hi")
scala > val msg2 = msg1.copy(body = "Hello")

scala >  msg2.sender.name
res0: String = Alice
```

Shallow-copy

```
scala > case class A(s: String, l: scala.collection.mutable.ArrayBuffer[Int])

scala > val a1 = A("something", ArrayBuffer(1, 2, 3, 4))
scala > val a2 = a1.copy()

scala > a1.l.remove(0)
res0: Int = 1

scala > a1
res1: A = A(something, ArrayBuffer(2, 3, 4))

scala > a2
res2: A = A(something, ArrayBuffer(2, 3, 4))
```

equals & hashCode methods

```
scala> val msg1  = Message("Hi")
msg1: Message = Message(Hi)

scala> val msg2  = Message("Hi")
msg2: Message = Message(Hi)

scala> msg1 == msg2
res5: Boolean = true
```

Generate bytecode

case class Person(name: String, age: Int)

Generated bytecode

```
Compiled from "Person.scala"
public class Person extends java.lang.Object implements scala.ScalaObject,scala.Product,scala.Serializable{
    public static final scala.Function1 tupled();
    public static final scala.Function1 curry();
    public static final scala.Function1 curried();
    public scala.collection.Iterator productIterator();
    public scala.collection.Iterator productElements();
    public java.lang.String name();
    public int age();
    public Person copy(java.lang.String, int);
    public int copy$default$2();
    public java.lang.String copy$default$1();
    public int hashCode();
    public java.lang.String toString();
    public boolean equals(java.lang.Object);
    public java.lang.String productPrefix();
    public int productArity();
    public java.lang.Object productElement(int);
    public boolean canEqual(java.lang.Object);
    public Person(java.lang.String, int);
}
```

Generated bytecode

```
Compiled from "Person.scala"
public final class Person$ extends scala.runtime.AbstractFunction2
implements scala.ScalaObject,scala.Serializable{
    public static final Person$ MODULE$;
    public static {};
    public final java.lang.String toString();
    public scala.Option unapply(Person);
    public Person apply(java.lang.String, int);
    public java.lang.Object readResolve();
    public java.lang.Object apply(java.lang.Object, java.lang.Object);
}
```

Example

```
abstract class Publication
case class Novel ( title : String , author : String ) extends Publication
case class Anthology ( title : String ) extends Publication

val a = Anthology ( " Great Poems " )
val b = Novel ( " The Castle " , " F . Kafka " )

scala > val books : List [ Publication ] = List (a , b )
books : List [ Publication ] = List ( Anthology ( Great Poems ) ,
Novel ( The Castle , F . Kafka ))
```

Limitation: auxiliary constructors

```
case class Point(x: Int, y: Int)

object Point {
  def apply(): Point = Point(0, 0)
}
```

Limitation: Inheritance

case class Publication()

case class Novel (title : String , author : String) extends Publication

<console>:13: error: case class Novel has case ancestor Publication, but case-to-case inheritance is prohibited. To overcome this limitation, use extractors to pattern match on non-leaf nodes.

case class Novel (title : String , author : String) extends Publication

Inheritance-equality problem

scala > case class Point(x: Int, y: Int)

scala > case class ColoredPoint(x: Int, y: Int, c: Color) extends Point(x, y)

TRUE: Point(0, 0) == ColoredPoint(0, 0, RED)

FALSE: ColoredPoint(0, 0, RED) == Point(0, 0)

Sealed Classes

```
sealed abstract class Publication

case class Novel ( title : String , author : String )
extends Publication

class Anthology ( title : String ) extends Publication
```

Case object

```
case object SomeName {
  . . .
}
```

Case object

- auto-generated methods: e*quals* & *toString*
- *serializable*
- *great for enums*

Case object example

abstract class Direction {. . .}

case object Forward extends Direction {...}

case object Backward extends Direction {...}

case object Left extends Direction {...}

Pattern matching

```
expression match {
  case pattern1 => expression1
  case pattern2 => expression2
  …
}
```

Constant patterns

```
scala > val month : Int = 5
month : Int = 5

scala > val monthString = month match {
  case 1 => " January "
  case 2 => " February "
  case 3 => " March "
  case 4 => " April "
  case 5 => " May "
}
monthString : String = May
```

Constant patterns

```
scala > val month : Int = 5
month : Int = 5

scala > month match {
  case 1 | 3 | 5 | 7 | 9 => println("odd")
  case 2 | 4 | 6 | 8 | 10 => println("even")
}
odd
```

Constant patterns

```
def isTrue(a: Any) = a match {
    case 0 | "" => false
    case something => true
}

scala > isTrue(45)
true
```

Patterns with ifs

```
i match {
    case a if 0 to 9 contains a => println("0-9 range")
    case b if 10 to 19 contains b => println("10-19 range")
    case c if 20 to 29 contains c => println("20-29 range")
    case _ => println(">= 30 range")
}

scala > val i = 21
20-29 range
```

Type patterns

```
def readTextFile(filename: String): List[String] = {
    try {
        Source.fromFile(filename).getLines.toList
    } catch {
        case e: Exception => Nil
    }
}
```

Type patterns

```
def readTextFile(filename: String): List[String] = {
    try {
        Source.fromFile(filename).getLines.toList
    } catch {
        case e: IOException => Logger.error(s"io: $e"); Nil
        case e: FileNotFoundException => Logger.error(e); Nil
    }
}
```

Sequence patterns

```
def seqToString(s: Seq[Int]) = {
  s match {
    case Nil => "empty list"
    case List (0, _, _) => "a three-element list with 0 as the first element"
    case List (1, _*) => "a list beginning with 1, having any number of elements"
    case Vector (1, _*) => "a vector beginning with 1 and having any number …"
  }
}
```

Sequence patterns

```
def seqToString(s: Seq[Int]) = {
  s match {
    case Nil => "empty list"
    case head::tail => "list with at least one elem"
    case List (0, _, _) => "a three-element list with 0 as the first element"
    case List (1, _*) => "a list beginning with 1, having any number of elements"
    case Vector (1, _*) => "a vector beginning with 1 and having any number ..."
  }
}
```

Case classes again

```
abstract class Publication

case class Novel ( title : String , author : String ) extends Publication

case class Anthology ( title : String ) extends Publication

val a = Anthology ( " Great Poems " )
val b = Novel ( " The Castle " , " F . Kafka " )
val books : List [ Publication ] = List (a , b )
```

Auto-generation

- Auto-generates methods:
  - *apply*
  - *accessors* for all constructor parameters
  - *copy*
  - *equals* & *hashCode*
  - *toString*
  - *unapply*

Constructor patterns

```
for ( book <- books ) {
  val description = book match {
    case Anthology ( title ) => title
    case Novel ( title , author ) =>  title + ", " + author
  }
  println ( description )
}
```

Constructor patterns

```
for ( book <- books ) {
  val description = book match {
    case Novel ( title , " F . Kafka " ) => s"Kafka: $title"
    case Novel ( title , author ) => title + " by " + author
    case other => "some other book"
  }
  println ( description )
}
```

Constructor patterns

```
for ( book <- books ) {
  val description = book match {
    case Novel ( title , " F . Kafka " ) => s"Kafka: $title"
    case Novel ( title , author ) => title + " by " + author
    case default => "some other book"
  }
  println ( description )
}
```

Wildcard patterns

```
for ( book <- books ) {
  val description = book match {
    case Novel ( title , _ ) => title
    case Anthology ( title ) => title
    case _ => " unknown publication type "
  }
  println ( description )
}
```

Pattern binder

```
for ( book <- books ) {
  val description = book match {
    case n @ Novel ( _ , _ ) => s"${n.author}: ${n.title}"
    case Anthology ( title ) => title
    case _ => " unknown publication type "
  }
  println ( description )
}
```

Pattern binder

```
for ( book <- books ) {
  val description = book match {
    case n @ Novel ( _ , _ ) if n.title.length > 10 => s"${n.author}: $ {n.title}"
    case Anthology ( title ) => title
    case _ => " unknown publication type "
  }
  println ( description )
}
```

Uniwersytet Wrocławski

Pattern binder

```
for ( book <- books ) {
  val description = book match {
    case n @ Novel ( _ , _ ) if n.title.length > 10 => s"${n.author}: $ {n.title}"
    case Anthology ( anth @ title ) => anth
    case _ => " unknown publication type "
  }
  println ( description )
}
```

Sealed Classes again

sealed abstract class Publication

case class Novel (title : String , author : String) extends Publication

case class Anthology (title : String)

extends Publication

Patterns in vals

```
scala > val a = Anthology ( " Great Poems " )
scala > val b = Novel ( " The Castle " , " F . Kafka " )

scala > val Novel(title, _) = b
title: String = " The Castle"

scala> val Novel(title, _) = a
<console>:15: error: constructor cannot be instantiated to expected type;
 found    : Novel
 required: Anthology
       val Novel(title, _) = a
```

Tuples

```
scala > val x  = ("Alice" , 25)
x : ( String , Int ) = (Alice , 25)

scala > val ( name , surname, age) = ("Alice" , "Kowalska", 25)
name: String = Alice
surname: String = Kowalska
age: Int = 25
```

Tuples

```
scala > val myTuple = ("Alice" , "Kowalska", 25)

scala > myTuple._2
res3: String = Kowalska

scala> myTuple._4
<console>:13: error: value _4 is not a member of (String, String, Int)
       myTuple._4
```

Tuples in pattern matching

```
def tuplePrint ( expr : Any ) =
  expr match {
    case (a , _ , b ) => println ( " matched " + a + b )
    case _ => ()
  }

scala > tuplePrint (( " a " , 55 , " tuple " ))
matched a tuple
```

Matching types

```
def generalSize ( x : Any ) = x match {
  case s : String => s.length
  case m : List[_] => m.length
  case a: Array[_] => a.length
  case _ => -1
}
```