Scala Interview Handbook — Batch 8

Generated: 2025-09-13 02:45:31Z (UTC)

Scala Theory & Cheatsheet

```
SCALA THEORY & CHEATSHEET (Quick Ref)
-------------------------------------
- Syntax: vals vs vars; methods; case classes; pattern matching.
- Collections: immutable List/Vector/Map/Set; mutable variants.
- Functional: map/flatMap/filter/fold; Options/Eithers; for-comprehensions.
- Performance: prefer immutable + structural sharing; use arrays for tight loops.
- Testing: ScalaTest AnyFunSuite; assertions; property-based (optional).
```

071. ParseCsvLine

Problem Overview & Strategy

ParseCsvLine — Detailed Explanation Approach: Idiomatic Scala solution with clear invariants and tested behavior. Correctness: Proven by invariant reasoning and unit tests. Complexity: See code comments.

Scala Solution

```
package problems


object Problem071ParseCsvLine {
  def parse(line:String): List[String] = {
    val res = scala.collection.mutable.ListBuffer[String]()
    val sb = new StringBuilder
    var i=0; var inQ=false
    while (i<line.length) {
      val c = line.charAt(i)
      if (c=='"') {
        if (inQ && i+1<line.length && line.charAt(i+1)=='"') { sb.append('"'); i+=1 }
        else inQ = !inQ
      } else if (c==',' && !inQ) {
        res += sb.toString; sb.clear()
      } else sb.append(c)
      i+=1
    }
    res += sb.toString
    res.toList
  }
}
```

ScalaTest

```
package tests

import org.scalatest.funsuite.AnyFunSuite
import problems.Problem071ParseCsvLine

class Problem071ParseCsvLineSpec extends AnyFunSuite {
  test("parse csv") { val r=Problem071ParseCsvLine.parse("a,"b,b",c"); assert(r==List("a","b,b","c")) }
}
```

072. UrlShortenerEncodeDecode

Problem Overview & Strategy

UrlShortenerEncodeDecode — Detailed Explanation Approach: Use precompiled regex patterns with sane anchors and character classes. Correctness: Patterns encode structural rules; not a full RFC check but covers common cases. Complexity: O(n) matching.

Scala Solution

```
package problems


object Problem072UrlShortenerEncodeDecode {
  private var id: Long = 0
  private val store = scala.collection.mutable.HashMap[Long,String]()
  private val chars = "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
  def encode(url:String): String = {
    id += 1; store(id)=url; base62(id)
  }
  def decode(code:String): String = store(base62inv(code))
  private def base62(n:Long): String = {
    var x=n; val sb=new StringBuilder; if (x==0) return "0"
    while (x>0) { sb.append(chars.charAt((x % 62).toInt)); x/=62 }
    sb.reverse.toString
  }
  private def base62inv(s:String): Long = s.foldLeft(0L)((acc,c)=> acc*62 + chars.indexOf(c))
}
```

ScalaTest

```
package tests

import org.scalatest.funsuite.AnyFunSuite
import problems.Problem072UrlShortenerEncodeDecode

class Problem072UrlShortenerEncodeDecodeSpec extends AnyFunSuite {
  test("shortener") { val c=Problem072UrlShortenerEncodeDecode.encode("http://x"); assert(Problem072UrlShortenerEncodeD
}
```

073. RateLimiterFixedWindow

Problem Overview & Strategy

RateLimiterFixedWindow — Detailed Explanation Approach: Maintain a window with counts/deque; expand and contract to satisfy constraints. Correctness: Each index enters/leaves window at most once, preserving minimality when contracted. Complexity: O(n) time, O(Σ) space.

Scala Solution

```
package problems


object Problem073RateLimiterFixedWindow {
  class Limiter(limit:Int, windowMillis:Long) {
    private var windowStart = 0L
    private var count = 0
    def allow(now:Long): Boolean = {
      if (now - windowStart >= windowMillis) { windowStart = now; count = 0 }
      if (count < limit) { count += 1; true } else false
    }
  }
}
```

ScalaTest

```
package tests

import org.scalatest.funsuite.AnyFunSuite
import problems.Problem073RateLimiterFixedWindow

class Problem073RateLimiterFixedWindowSpec extends AnyFunSuite {
  test("fixed window") { val l=new Problem073RateLimiterFixedWindow.Limiter(2,1000); assert(l.allow(0) && l.allow(1) &&
}
```

074. TokenBucketLimiter

Problem Overview & Strategy

TokenBucketLimiter — Detailed Explanation Approach: Window and token-bucket algorithms for rate limiting; exponential backoff for retries; blocking queues for producer/consumer. Correctness: Counters/tokens enforce limits; backoff reduces contention. Complexity: Amortized O(1) per event.

Scala Solution

```
package problems


object Problem074TokenBucketLimiter {
  class TokenBucket(ratePerSec:Double, capacity:Int) {
    private var tokens: Double = capacity
    private var last: Long = 0L
    def allow(nowMillis:Long): Boolean = {
      if (last==0L) last = nowMillis
      val delta = (nowMillis - last) / 1000.0
      tokens = math.min(capacity, tokens + delta * ratePerSec)
      last = nowMillis
      if (tokens >= 1.0) { tokens -= 1.0; true } else false
    }
  }
}
```

ScalaTest

```
package tests

import org.scalatest.funsuite.AnyFunSuite
import problems.Problem074TokenBucketLimiter

class Problem074TokenBucketLimiterSpec extends AnyFunSuite {
  test("token bucket") { val b=new Problem074TokenBucketLimiter.TokenBucket(1.0,1); assert(b.allow(0)); assert(!b.allow
}
```

075. DebounceFunction

Problem Overview & Strategy

DebounceFunction — Detailed Explanation Approach: Window and token-bucket algorithms for rate limiting; exponential backoff for retries; blocking queues for producer/consumer. Correctness: Counters/tokens enforce limits; backoff reduces contention. Complexity: Amortized O(1) per event.

Scala Solution

```
package problems


object Problem075DebounceFunction {
  class Debounce(waitMillis:Long) {
    private var last: Long = Long.MinValue
    def run(now:Long)(f: => Unit): Boolean = {
      if (now - last >= waitMillis) { last = now; f; true } else false
    }
  }
}
```

ScalaTest

```
package tests

import org.scalatest.funsuite.AnyFunSuite
import problems.Problem075DebounceFunction

class Problem075DebounceFunctionSpec extends AnyFunSuite {
  test("debounce") { val d=new Problem075DebounceFunction.Debounce(100); assert(d.run(0){()}, "first ok"); assert(!d.ru
}
```

076. ThrottleFunction

Problem Overview & Strategy

ThrottleFunction — Detailed Explanation Approach: Window and token-bucket algorithms for rate limiting; exponential backoff for retries; blocking queues for producer/consumer. Correctness: Counters/tokens enforce limits; backoff reduces contention. Complexity: Amortized O(1) per event.

Scala Solution

```
package problems


object Problem076ThrottleFunction {
  class Throttle(limit:Int, windowMillis:Long) {
    private var times = List[Long]()
    def allow(now:Long): Boolean = {
      times = (now :: times).filter(t => now - t < windowMillis)
      if (times.length <= limit) true else { times = times.tail; false }
    }
  }
}
```

ScalaTest

```
package tests

import org.scalatest.funsuite.AnyFunSuite
import problems.Problem076ThrottleFunction

class Problem076ThrottleFunctionSpec extends AnyFunSuite {
  test("throttle") { val t=new Problem076ThrottleFunction.Throttle(2,1000); assert(t.allow(0) && t.allow(10) && !t.allo
}
```

077. DecoratorTiming

Problem Overview & Strategy

DecoratorTiming — Detailed Explanation Approach: Idiomatic Scala solution with clear invariants and tested behavior. Correctness: Proven by invariant reasoning and unit tests. Complexity: See code comments.

Scala Solution

```
package problems


object Problem077DecoratorTiming {
  def time[T](f: => T): (T, Long) = {
    val s = System.nanoTime(); val r = f; (r, System.nanoTime()-s)
  }
}
```

ScalaTest

```
package tests

import org.scalatest.funsuite.AnyFunSuite
import problems.Problem077DecoratorTiming

class Problem077DecoratorTimingSpec extends AnyFunSuite {
  test("timing") { val (v,ns)=Problem077DecoratorTiming.time{ 1+1 }; assert(v==2 && ns>=0) }
}
```

078. ContextManagerExample

Problem Overview & Strategy

ContextManagerExample — Detailed Explanation Approach: Idiomatic Scala solution with clear invariants and tested behavior. Correctness: Proven by invariant reasoning and unit tests. Complexity: See code comments.

Scala Solution

```
package problems


object Problem078ContextManagerExample {
  def using[A <: AutoCloseable, B](r: A)(f: A => B): B =
    try f(r) finally if (r != null) r.close()
}
```

ScalaTest

```
package tests

import org.scalatest.funsuite.AnyFunSuite
import problems.Problem078ContextManagerExample

class Problem078ContextManagerExampleSpec extends AnyFunSuite {
  test("using") { class R extends AutoCloseable { var closed=false; def close()= { closed=true } }; val r=new R; Proble
}
```

079. PickleSerialize

Problem Overview & Strategy

PickleSerialize — Detailed Explanation Approach: Preorder with sentinels for trees; adjacency lists for graphs; custom JSON builder for maps. Correctness: Deterministic encoding/decoding yields isomorphic structures. Complexity: O(n) in nodes/keys.

Scala Solution

```
package problems
import java.io._
object Problem079PickleSerialize {
  @SerialVersionUID(1L) case class Person(name:String, age:Int) extends Serializable
  def roundtrip(p:Person): Person = {
    val baos = new ByteArrayOutputStream()
    val oos = new ObjectOutputStream(baos); oos.writeObject(p); oos.close()
    val b = baos.toByteArray
    val ois = new ObjectInputStream(new ByteArrayInputStream(b))
    ois.readObject().asInstanceOf[Person]
  }
}
```

ScalaTest

```
package tests

import org.scalatest.funsuite.AnyFunSuite
import problems.Problem079PickleSerialize

class Problem079PickleSerializeSpec extends AnyFunSuite {
  test("serialize/deserialize") { val p=Problem079PickleSerialize.Person("a",1); assert(Problem079PickleSerialize.round
}
```

080. JsonSerializeCustom

Problem Overview & Strategy

JsonSerializeCustom — Detailed Explanation Approach: Preorder with sentinels for trees; adjacency lists for graphs; custom JSON builder for maps. Correctness: Deterministic encoding/decoding yields isomorphic structures. Complexity: O(n) in nodes/keys.

Scala Solution

```
package problems
object Problem080JsonSerializeCustom {
  def toJson(m: Map[String,Any]): String = {
    def esc(s:String) = s.replace("\\","\\\\").replace("""","\\"")
    def render(v:Any): String = v match {
      case s:String => """ + esc(s) + """
      case b:Boolean => b.toString
      case n:Int => n.toString
      case n:Long => n.toString
      case d:Double => if (d.isNaN) ""NaN"" else d.toString
      case l:List[_] => l.map(render).mkString("[",",","]")
      case m:Map[_,_] => toJson(m.asInstanceOf[Map[String,Any]])
      case null => "null"
      case other => """ + esc(other.toString) + """
    }
    m.map{case(k,v)=> """ + esc(k) + "":" + render(v)}.mkString("{",",","}")
  }
}
```

ScalaTest

```
package tests

import org.scalatest.funsuite.AnyFunSuite
import problems.Problem080JsonSerializeCustom

class Problem080JsonSerializeCustomSpec extends AnyFunSuite {
  test("toJson") { val s=Problem080JsonSerializeCustom.toJson(Map("x"->1,"y"->"z")); assert(s.contains(""x":1") && s.co
}
```