Kotlin Cookbook

Jaideep Ganguly, Sc.D.

Contents

Preface

This book has been written to help Java developers migrate to Kotlin easily. A Java developer should be able to migrate to Kotlin and be reasonably proficient in a day. The IntelliJ IDE from JetBrains, creators of Kotlin, should be used to develop in Kotlin.

The Object Oriented Programming (OOP) paradigm mostly often results in obfuscating code using a plethora of classes, sub-classes and interfaces that add levels of abstraction but no value. The code follows a verbose, iterative, if-then style to encode business logic. For a piece of code with business logic of just 10 lines, it is not unusual to see developers write 100 lines in form of Interface, Abstract Classes, Concrete Classes, Factories, Abstract Factories, Builders and Managers. Making changes has become tedious, time consuming and error prone. Most of the assumptions around the need for extensibility is false and much of the code written for the sake of extensibility is never used.

Kotlin, a next generation language, helps eliminate verbose iterative and *if-then* style of coding and replace it with simple, terse and easily readable code using a **functional** style through the use of *collections*, *filters*, *lambdas and maps*. One can rely on *static objects* and use *extension functions* whenever required. However, Kotlin also allows you to write in the OOP style.

Kotlin inter-operates well with the Java based ecosystem in which we have heavy investments. It is easy to write high performant non-blocking, asynchronous, event driven code in half the size or even lesser than comparable Java code. It is easy to use its language constructs and write smaller code which removes null-checks, loops, branches and other boiler-plate code, leaving developers to focus on business logic than on repeated boilerplate code. Using Kotlin as a development language is a shift towards the right direction. An added benefit is that it will reduce our hardware costs. It learns from modern languages like C#, GO, Python and Ruby, and plugs in concepts in JVM ecosystem never seen before, and that too, in a highly intuitive and a smooth learning curve. Kotlin is the language of choice for Android development and is officially endorsed by Google.

CHAPTER 1

Basic Structure

1.1 Class & Function

Listing 1.1 – Class & Function.

```
package template

// A class can be stand alone, outside any function
fun mytest(x: Int): Int {
    var x: Int = 3
    x = 4
    println("The value of x = $x")
    return 1
}

/* modifiers are on class, not allowed on variables in functions
   public is used by default, which means that your declarations will be
   visible everywhere;
   internal - visible everywhere in the same module, eg.kotlin_template
   private - visible inside the file containing the declaration;
   protected - strictly accessible only by declaring class \& subclasses.
   Note: to use a visible top-level declaration from another package,
   you should still import it. */
internal class BasicStruct() {

    fun foo(x: Int, y: Int): Unit {

        // Type inference
        val z = 10
        // Explicit type declaration
        val zz: Int = 12

        // String interpolation
        println("sum of x and y is ${x + y}")

        // Smart casts
        if (z is Int)
            println("$z is Int")

        // Class name
        println(z::class.simpleName.toString())
        println(z::class.qualifiedName.toString())
    }

    fun foo(letter: Char) {
        println(letter)
    }

    fun foo(letter: Char, num: Int,
            area: Float = 10.0f, name: String = "Josh") {
        // -s makes it left flush, else right flush
        println("%c %2d %.2f %-8s".format(letter,num,area,name))
    }
```

1.2 Null Check

Listing 1.2 – Null Check.

```
    fun fooNull() {
        var a: String = "abc"
        var b: String? = null
        //a = null  // will not compile if uncommented; null check

        if (b?.length == null)
            println("b is null")  // safe call, evaluates to null; no NPE

//          try {
//              if (b!!.length == null)
//                  println("null")
//          }
//          catch (e: Exception) {
//                 e.printStackTrace()
//          }
    }

}
```

CHAPTER 2

Class, Inheritance, Interface

2.1 Class

Listing 2.1 – Class.

```kotlin
/* www.callicoder.com/kotlin-inheritance/
   In Kotlin, get(), set() are auto-generated, no need need to create them
   www.programiz.com/kotlin-programming/getters-setters#introduction */
package template

class A(val id: Int) {
    var name: String = ""  // Note: id not required to be instance variable
    var test: Float = 0.0f // id instance variable, will cause compile error

    constructor(id: Int, name: String): this(id) {
        this.name = name
    }

    init {
        test = 0.0f // test not passed in constructor, no need for function
    }
}
```

2.2 Inheritance

Listing 2.2 – Inheritance.

```kotlin
class Foo3(name: String, num: Int, city: String, state: String):
    Foo2(name,num,city) {
}
```

2.3 inheritance & Interface

Listing 2.3 – Inheritance & Interface.

```kotlin
open class Foo2: Foo, IFoo {
    var city: String = ""
    // A class has a primary constructor and multiple secondary constructors
    constructor(name: String, num: Int, city: String): super(name, num) {
        this.city = city
    }

    init { println("In Foo2") } // initialization code

    override fun sum(x: Int, y: Int): Int { // override must for extending
        return(x-y)
    }

    fun mult(x: Int, y: Int): Int {
        return(x*y)
    }

    override fun f1(): String {
        println("In f1")
        return("f1")
    }
}
```

2.4 Inheritance & Interface

Listing 2.4 – Inheritance & Interface.

```
open class Foo(name: String, age: Int) { // default final, open extendible
    var name: String = ""
    var age:  Int    = 0
    var msg:  String = ""
    var id:   Int = 0

    // constructor cannot contain code
    constructor(name: String, age: Int, msg: String):  this(name,age) {
        println("Secondary constructor of base class Foo: $name $age $msg")
        this.name = name
        this.age  = age
        this.msg  = msg
    }

    init {  // initilization code
        id += 1000
        println("In init of base class:id = " + this.id)
    }

    open fun sum (x: Int, y: Int): Int {
        return x + y + this.id
    }
}

interface IFoo {
    fun f1(): String
}
```

2.5 data class

Listing 2.5 – data class.

```
// Data Class
data class Person(var name:   String,
                  var age:    Int,
                  var gender: Char)
```

2.6 object

Listing 2.6 – object.

```
object ObjectExample {
    fun hello(): String {
        return "hello from singleton object"
    }
}
```

2.7 enum class

Listing 2.7 – enum class.

```
/* kotlinlang.org/docs/reference/enum-classes.html
   Each enum constant is an object. Enum constants are separated by commas.
   Each enum is an instance of the enum class, e.g., initialized as: */
enum class ColorEnum(val colorCode: Int) { //val c = ColorEnum.Red.colorCode
    Red(2),
    Blue(11212),
    Green(21212),
    Orange(212121)
}
```

2.8 Delegation

Listing 2.8 – Delegation.

```
// Delegation
interface IBase {
    fun print()
}

class BaseImpl(val x: Int): IBase {
    override fun print() { print(x) }
}

class Derived(b: IBase): IBase by b

/* You can declare a class as the delegation of another class
   to make callable for all the methods of it. */
/*
interface A { ... }
class B:  A { }
val b = B()
// initiate C, and delegate all methods of B defined in A
class C:  A by b
*/
```

2.9 Generic

Listing 2.9 – Generic.

```
class test2<T> { // Will not compile if <T> is removed from class definition
    fun tt(x: T) {
        println("Testing compilation in generics")
    }
}
```

2.10 Lazy Init

Listing 2.10 – Lazy Init.

```
// medium.com/@mohitsharma_49363/
//            android-kotlin-lazy-lateinit-and-delegates-9e5f01c561dc
// Lazy Init
val test: String by lazy {
    val testString = "some value"
    testString // must be last line as in lambda expression and is returned
}

fun doSomething() {
    println("Length of string is "+test.length)
```

CHAPTER 3

Condition & Loop

3.1 Condition

Listing 3.1 – Condition.

```
package template

internal class Logic() {

    fun foo (letter: Char) {
        println(letter)
    }

    fun foo (letter: Char, num: Int, area: Float = 100.00f,
             name: String = "Josh") {
        // -s for left flush, else right flush
        println("%c %2d %.2f %-8s".format(letter,num,area,name))
    }

    fun foo(name: String) {

        if (name.compareTo("Josh") == 0) {
            println("I am Josh")
        }
        else if (name.compareTo("Tua") == 0) {
            println("I am Tua")
        }
        else {
            println("I am somebody")
        }


        when(name) {
            // if satisfied, next conditions will not be evaluated
            is String   -> {
                println("Josh is a string")
            }
            "Josh"      -> {
                println("Hello Mr. $name")
            }
            "Tua"       -> {
                println("Hello Ms. $name")
            }
            else        -> {
                println("Good Morning")
            }
        }
    }
```

3.2 for

Listing 3.2 – for.

```
    fun forloop() {

        for (i in 1..10) { println(i) }
        for (i in 0 until 10) { println(i) }
        for (i in 2..10 step 2) { println(i) }
        for (i in 10 downTo 1) { println(i) }

        for (i in 1..10) {
            if (i in 5..8 step 2)
                print("$i ")
        }
    }
```

3.3 while

Listing 3.3 – while.

```
    fun whileloop() {
        var i = 0
        while(true) {
            println("val = $i")
            i++
            if (i > 10)
                break
        }
    }
}
```

CHAPTER 4

LAMBDA FUNCTION

4.1 Lambda Function

Listing 4.1 – Lambda Function.

```
package template

/* Kotlin functions are first-class or higher order functions  i.e.,
   they can be stored in variables and data structures, passed as
   arguments to and returned from other higher-order functions. */
class LambFun() {

    fun example(x: Int, y: Int, name: String) {

        val lambfunc1: (Int, Int) -> Int = { x, y  -> x + y }
        // val lambfunc1 = { x: Int, y: Int -> x + y }  // type inference
        var test = add(21,23,lambfunc1)
        println("Adding: ${test}")

        val lambfunc2: (String, Int, Int) -> Int = {
                s, a, b ->  println("1: ${s}")
                            var x = a + b
                            println("2: $a $b")
                            x // last line is returned
        }
        hof("Hello", lambfunc2, "Hello2")

        // Alternate syntax, lambda function is the last parameter in { }
        hof("Hello") {
                s, a, b ->  println("1: ${s}")
                            var x = a + b
                            println("2: $a $b")
                            a + b // last line is returned
        }
    }

    fun add(a: Int, b: Int, lambfun: (Int, Int) -> Int) : Int {
        val x = lambfun(a,b)
        println(x)
        return x;
    }

    fun hof(msg: String, lf: (String, Int, Int) -> Int, m: String = "ok") {
        println("msg = ${msg}")
        val result = lf("Josh", 2, 3) // Invoke the lambda with arguments
        println("result is $result")
    }

    fun hof( msg: String, lf: (String, Int, Int) -> Int) {
        println("msg = ${msg}")
        val result = lf("Josh", 2, 3) // Invoke the lambda with arguments
        println("result is $result")
    }
}
```

CHAPTER 5

HIGHER ORDER FUNCTION

5.1 Higher Order Function

Listing 5.1 – Higher Order Function.

```
package template

/* Kotlin functions are first-class or higher order functions  i.e.,
   they can be stored in variables and data structures, passed as
   arguments to and returned from other higher-order functions. */
class HOF() {

    fun example(msg: String, name: String) {
        foo(msg,name,this::bar) // this or instance of class
    }

    fun foo(msg: String, name: String, op: (String) -> Unit) {
        print("${msg} ")
        op(name)
    }

    fun bar(name: String) {
        println("${name}!")
    }

}
```

CHAPTER 6

EXTENSION FUNCTION

6.1 Extension Function

Listing 6.1 – Extension Function.

```
package template

/* Extend a class with new functionality without having to inherit from
   the class or use any type of design pattern such as Decorator.
   This is done via special declarations called extensions. To declare
   an extension function, we need to prefix its name with a receiver type,
   i.e. the type being extended. */

// add a swap function to MutableList<Int>:
fun MutableList<Int>.swap(index1: Int, index2: Int) {
    val tmp = this[index1] // 'this' corresponds to the list
    this[index1] = this[index2]
    this[index2] = tmp
}
```

6.2 apply,also,run,let

Listing 6.2 – apply,also,run,let.

```
fun testApply() {
    var per = Pers("Tim",22)
    var per2 = Pers("Tim",24)

    // No need to return self, No need for null check; No this received
    run {
        if (per.age == 20)
            per
        else
            per2
    }.printPer()

    // this received
    with(per) {
        per
    }.printPer()


    // No need to return self, need null check; send this as argument
    per?.run {
        age += 1
    }

    // send it
    per?.let {
            it -> it.age += 1
    }

    /* returns self, i.e., this
       send this */
    per?.apply {
        age += 1
    }.printPer()

    // send it
    val per9 = per?.also { it ->
        it.age += 1
    }.printPer()

    per?.apply {
        this.age = this.age + 10
    }.printPer()
}
```

6.3 data class

Listing 6.3 – data class.

```
data class Pers(var name: String, var age: Int) {
    fun printPer() {
        println(this.toString())
    }
}
```

CHAPTER 7

COLLECTION

7.1 Array

Listing 7.1 – Array.

```
package template

class Coll() {

    fun collect() {
        // TYPE     SIZE        MUTABLE
        // Array    Fixed       Yes
        // List     Flexible    No

        var col1 = arrayOf<String>("India", "USA", "China", "Australia")
        col1.plus("Nepal")
        col1.set(3, "New Zealand")
        println(col1.get(3))
        for (x in col1)
            println(x)

        col1 += "Nepal"
        for (x in col1)
            println(x)
```

7.2 Array of fixed size

Listing 7.2 – Array of fixed size.

```
        var tmp2: Array<String> = Array(10) { it -> "" }
        val emptyStringArray = arrayOf<String>()

        var x: Array<Int> = Array(5) { it -> 0 }
        var y: ArrayList<Int> = ArrayList()
```

7.3 List

Listing 7.3 – List.

```
        var col2 = listOf("Banana", "Kiwifruit", "Mango", "Apple")
        col2 += "Grape"
        println(col2)
```

7.4 Mutalble List

Listing 7.4 – Mutable List.

```
        col2 = mutableListOf<String>()
        col2.add("Apple")
        col2.add("Banana")
        col2.add("Apricot")
        col2.remove("Banana")
        println(col2)

        var coll = mutableListOf<String>("Banana", "Mango", "Apple")
```

7.5 Set

Listing 7.5 – Set.

```
        val col3 = mutableSetOf<String>("Lion", "Cat", "Hippo", "Hippo")
        col3.add("Dog")           // Add an element to the set
        col3.remove("Python")     // Remove a existing element from a set
        println(col3)             // Notice "Hippo" occurs only once
```

7.6 Slicing

Listing 7.6 – Slicing.

```
        var colslice = col1.slice(0..2)
        colslice = col1.slice((1) until 2)
```

7.7 Map

Listing 7.7 – Map.

```
        // Map has infix style f(10) is represented as f to 10
        // www.callicoder.com/kotlin-infix-notation/
        // inline - function call is substituted by function body
        val col4 = mutableMapOf("CHCH" to 50000, "GIS" to 36100)
        col4["DUN"] = 118500     // Add an entry to the map
        col4["CHCH"] = 389700    // Change an existing entry in the map
        col4.remove("GIS")       // Remove an entry from the map by key

        for ((key, value) in col4) {
            println("$key $value")
        }
```

7.8 Mutate Collection

Listing 7.8 – Mutate Collection.

```
        // To mutate a collection, you MUST use the iterator object
        var iter = coll.listIterator()
        while (iter.hasNext()) {
            var item = iter.next()

            if (item == "Banana")
                iter.remove()

            if (item == "Kiwifruit") {
                iter.remove()
                iter.add("KiwiFruits")
            }

            if (item == "Mango")
                iter.add("Guava")
        }
        println(coll)
```

7.9 Closure

Listing 7.9 – Mutate Collection.

```
        /* A closure is a function that carries an implicit binding to all
           the variables referenced within it.
           Note that sum is available inside the lambda function */
        var sum = 0
        var ints = listOf(1, 2, 3, 4)
        ints.filter { it > 0 }.forEach {
            sum += it
        }
        println(sum)
```

7.10 filter a list

Listing 7.10 – Filter a list.

```
        col2.filter { it.startsWith('A', ignoreCase = true) }
            .sortedBy { it }
        println(col2)

        // Movie
        var m1 = Movie("Ben Hur",       "Historical", 1000, 8.2f)
        val m2 = Movie("Quo Vadis",     "Historical", 1100, 7.8f)
        val m3 = Movie("Avenger",       "SciFi",      2100, 7.5f)
        val m4 = Movie("Patton",        "War",        1200, 7.8f)
        val m5 = Movie("Intersteller",  "SciFi",      9100, 8.5f)

        var mlMovie = mutableListOf<Movie>()
        mlMovie.add(m1)
        mlMovie.add(m2)
        mlMovie.add(m3)
        mlMovie.add(m4)
        mlMovie.add(m5)

        val mlname = mutableListOf<String>("Ben Hur", "Quo Vadis",
            "Avenger", "Patton", "Interstellar")

        var fmlkey = mlMovie
            .filter { it.genre == "SciFi"}
            .map { it.genre }
            .toSet()

        var fml = mlMovie
            .filter { ((it.genre == "SciFi") ||
                       (it.genre == "Historical")) }
            .filter { it.genre in fmlkey}
            .filter { it.rating > 8.0 }
            .map { it.genre }
            .sorted()
            .toSet()      // returns the movie genre only and as a set
        println(fml)
```

7.11 filter a map

Listing 7.11 – Filter a map.

```
        var m = mutableMapOf<String, Any>()
        var t = mutableListOf<Map<String, Any>>()
        m["id"] = 101
        m["name"] = "Jaideep"
        m["status"] = true
        t.add(m)

        m = mutableMapOf<String, Any>()
        m["id"] = 102
        m["name"] = "Josh"
        m["status"] = true
        t.add(m)

        var tmp = t.filter { it -> ( it["name"] == "Jaideep" ) &&
                                   ( it["id"] == 101 )}
        var tmp3 = t.filter { (it["id"] as Int >= 100) }

        println(tmp)

        println((tmp[0]).get("status"))
        println((tmp[0] as Map<String, Any>).get("status"))
```

7.12 fold & reduce

Listing 7.12 – fold & reduce.

```
        /* fold does the same thing as reduce.
           fold takes an explicit initial value whereas,
           reduce uses the 1st element from the list as the initial value.*/
        val total = listOf(1, 2, 3, 4, 5).fold(0,
            { total, next -> total + next })
        println("total: " + total)

        var mul = listOf(1, 2, 3, 4, 5).reduce({ mul, next -> mul * next })
        println("mul: " + mul)
    }
}
```

7.13 data class

Listing 7.13 – data class.

```
public data class Movie(val name: String, val genre: String, val views: Int,
                        val rating: Float)

public data class Per(val name: String, val age: Int, val genre: String)
```

CHAPTER 8

GENERIC

8.1 Generic

Listing 8.1 – Generic.

```
/* The class is a template for an object but concrete object have a type.
   Box is a class, while Box<Int> , Box<A> , Box<List<String>> are types
   generated by generic class Box. Note that primitives and arrays in Java
   do not have a class or interface but they are types.*/

/* proandroiddev.com/
        understanding-generics-and-variance-in-kotlin-714c14564c47
   www.i-programmer.info/programming/
        other-languages/12478-the-programmers-
            guide-to-kotlin-covariance-a-contravariance.html
   www.baeldung.com/kotlin-generics */

package template

class ParameterizedClass<T>(private val value: T) {
    fun getValue(): T {
        return value
    }
}

// Covariant
class ParameterizedProducer<out T>(private val value: T) {
    fun getValue(): T {
        return value
    }
}

// Contravariant
class ParameterizedConsumer<in T> {
    fun toString(value: T): String {
        return value.toString()
    }
}

class GenFunClass {

    init {
    }

    fun genFun(t: T) {
        println(t)
    }

    fun <T: Comparable<in T>> genFun(t: T) {
    }
}

data class T(val id: Int, val name: String)
```

CHAPTER 9

CONCURRENCY

9.1 Concurrency

Listing 9.1 – Concurrency.

```
/* kotlinlang.org/docs/tutorials/coroutines/coroutines-basic-jvm.html
   resocoder.com/2018/10/06/kotlin-coroutines-tutorial-
                stable-version-async-await-withcontext-launch/
   medium.com/@elizarov/blocking-threads-suspending-
                coroutines-d33e11bf4761*/

package template

import kotlinx.coroutines.*

import java.util.concurrent.ExecutorService
import java.util.concurrent.Executors

val t0 = System.currentTimeMillis()
private val executorService = Executors.newWorkStealingPool(100)
val dispatcher = executorService.asCoroutineDispatcher()


private fun printMemory() {
    GlobalScope.launch {
        while (true) {
            println("Memory used: " +
                    "${(Runtime.getRuntime().totalMemory() -
                            Runtime.getRuntime().freeMemory()) /
                                1024 / 1024} MB")
            delay(1000);
        }
    }
}
```

9.2 Suspending Function

Listing 9.2 – Suspending Function.

```
suspend fun printlnDelayed(message: String) {
    // Complex calculation
    delay(3000)
    println(message)
}
```

9.3 Blocking Function

Listing 9.3 – Blocking Function.

```
fun exampleBlocking() = runBlocking {
    println("one")
    printlnDelayed("two")
    println("three")
```

9.4 Run on another thread but still blocking the main thread

Listing 9.4 – Run on another thread but still blocking the main thread.

```
fun exampleBlockingDispatcher(){
    runBlocking(Dispatchers.Default) {
        println("one - from thread ${Thread.currentThread().name}")
        printlnDelayed("two - from thread ${Thread.currentThread().name}")
    }
    /* Outside of runBlocking to show that
       it's running in the blocked main thread */
    println("three - from thread ${Thread.currentThread().name}")
    // It still runs only after the runBlocking is fully executed.
```

9.5 Run blocking with custom dispatcher

Listing 9.5 – Run blocking with custom dispatcher.

```
/* runBlocking, customDispatcher, no need for job.join()
   but need to shutdown customDispatcher */
fun exampleLaunchCoroutineScope() = runBlocking {
    println("one - from thread ${Thread.currentThread().name}")

    val customDispatcher = Executors.newFixedThreadPool(2)
            .asCoroutineDispatcher()
    launch(customDispatcher) {
        printlnDelayed("two - from thread ${Thread.currentThread().name}")
        var cht1 = longCompute(10)
        println(cht1)
        var cht2 = longCompute(20)
        println(cht2)
    }

    println("three - from thread ${Thread.currentThread().name}")

    (customDispatcher.executor as ExecutorService).shutdown()
}
```

9.6 Async

Listing 9.6 – Async.

```
fun parallel() = runBlocking {
    val startTime = System.currentTimeMillis()

    val deferred1 = async { longCompute(10) }
    val deferred2 = async { longCompute(20) }
    val deferred3 = async { longCompute(30) }

    val sum = deferred1.await() + deferred2.await() + deferred3.await()
    println("asyn   c/await result = $sum")

    val endTime = System.currentTimeMillis()
    println("Time taken: ${endTime - startTime}")
}
```

9.7 Serial

Listing 9.7 – Serial.

```
fun serial() = runBlocking {
    val startTime = System.currentTimeMillis()

    val result1 = withContext(Dispatchers.Default) { longCompute(10) }
    val result2 = withContext(Dispatchers.Default) { longCompute(20) }
    val result3 = withContext(Dispatchers.Default) { longCompute(30) }

    val sum = result1 + result2 + result3
    println("async/await result = $sum")

    val endTime = System.currentTimeMillis()
    println("Time taken: ${endTime - startTime}")
}
```

9.8 Long running function

Listing 9.8 – Long running function.

```
suspend fun longCompute(startNum: Long): Long {
    delay(startNum*100)
    println(startNum)
    return startNum * 100
}
```

CHAPTER 10

DATABASE

10.1 CRUD

Listing 10.1 – CRUD.

```
// https://www.tutorialkart.com/kotlin/connect-to-mysql-database-from-kotlin-using-jdbc/
/**
@author    Jaideep Ganguly
@since     03/20/2018
 */

package template

import org.json.simple.JSONArray
import org.json.simple.JSONObject
import java.sql.*
import java.util.Properties


object ServerDB {

    internal var conn :Connection? = null
    internal var username = "root" // provide the username
    internal var password = "root" // provide the password
```

10.2 Connect

Listing 10.2 – Connect.

```
    fun getConnection() {

        val connectionProps = Properties()
        connectionProps.put("user", username)
        connectionProps.put("password", password)
//        println(connectionProps)

        try {
            Class.forName("com.mysql.jdbc.Driver").newInstance()
            conn = DriverManager.getConnection(
                "jdbc:" + "mysql" + "://" +
                        "127.0.0.1" +
                        ":" + "3306" + "/" +
                        "",
                connectionProps)
            println("DB connection opened")
        } catch (ex :SQLException) {
            // handle any errors
            ex.printStackTrace()
        } catch (ex :Exception) {
            // handle any errors
            ex.printStackTrace()
        }
    }
```

10.3 Select

Listing 10.3 – Select.

```
    fun select(sql: String, typ: String): JSONObject {

        var acol = sql.split("SELECT")[1].split("FROM")[0].split(",")
        var colName: Array<String> = Array( acol.size, { it -> " " })
        for (i in 0..acol.size-1) {
            colName.set(i,acol[i].trim(' '))
//            println(colName.get(i))
        }

        var atyp = typ.split(",")
        var colTyp = Array(atyp.size, {i -> ""})

        for (i in 0 .. colTyp.size-1) {
            colTyp.set(i,atyp[i].trim())
        }

        var stmt:      Statement? = null
        var resultset: ResultSet? = null

        var jsonArray = JSONArray()

        var ncol = 0
        var nrow = 0

        try {
            stmt = conn!!.createStatement()

            stmt.execute(sql)
            resultset = stmt.resultSet
            val rsmd :ResultSetMetaData = resultset!!.metaData
            ncol = rsmd.columnCount
            val col = arrayOfNulls<Any>(ncol)

            while (resultset!!.next()) {

                var jsonObj = JSONObject()

                for (i in 0..(ncol - 1)) {
                    if (colTyp.get(i).compareTo("Int") == 0) {
                        col[i] = resultset.getInt(i + 1).toString()
                    }
                    else if (colTyp.get(i).compareTo("String") == 0) {
                        col[i] = resultset.getString(i + 1).toString()
                    }
                    else if (colTyp.get(i).compareTo("Float") == 0) {
                        col[i] = resultset.getFloat(i + 1).toString()
                    }
                    else if (colTyp.get(i).compareTo("Date") == 0) {
                        col[i] = resultset.getDate(i + 1).toString()
                    }
                }

                for (j in 0..(ncol - 1)) {
//                    jsonObj.put(colName.get(j), col.get(j))
                    jsonObj.put(j.toString(), col.get(j))
                }

                jsonArray.add(jsonObj)
                nrow++
            }
```

```
61          } catch (ex: SQLException) { // handle any errors
62              ex.printStackTrace()
63          }
64          finally {
65              // release resources
66              if (resultset != null) {
67                  try {
68                      stmt?.close()
69                      resultset.close()
70                  } catch (sqlEx: SQLException) {
71                      sqlEx.printStackTrace()
72                  }
73
74                  resultset = null
75              }
76
77              if (stmt != null) {
78                  try {
79                      stmt.close()
80                  }
81                  catch (sqlEx: SQLException) {
82                      sqlEx.printStackTrace()
83                  }
84
85                  stmt = null
86              }
87
88              /*if (conn != null) {
89                  try {
90                      conn!!.close()
91                  }
92                  catch (sqlEx :SQLException) {
93                      sqlEx.printStackTrace()
94                  }
95
96                  conn = null
97              }*/
98          }
99
100         // Total Count
101         var sqltot = sql.split("LIMIT")[0]
102         stmt = conn!!.createStatement()
103         if (stmt!!.execute(sqltot)) {
104             resultset = stmt.resultSet
105         }
106         var ntot = 0
107         while (resultset!!.next()) {
108             ntot++
109         }
110
111         var jsonObj = JSONObject()
112         jsonObj.put("ncol",ncol)
113         jsonObj.put("nrow",nrow)
114         jsonObj.put("ntot",ntot)
115         jsonObj.put("rows",jsonArray)
116
117         // DEBUG
118 //        Util.log(sql)
119 //        Util.log(jsonObj.toString()+'\n')
120
121         return(jsonObj)
122     }
```

10.4 Create, Update & Delete

Listing 10.4 – Create, Update & Delete.

```
    fun crud(sql: String): JSONObject {

        var stmt: Statement? = null
        var jsonObj = JSONObject()
        try {
            stmt = conn!!.createStatement()
            stmt.execute(sql)
            jsonObj.put("nins",1)
        }
        catch (ex: SQLException) {
            ex.printStackTrace()
            jsonObj.put("nins",0)
        }

        // DEBUG
        Util.log(sql)
        Util.log(jsonObj.toString()+'\n')
        return(jsonObj)
    }
}
```

CHAPTER 11

Utility Functions

11.1 Utility Functions

Listing 11.1 – Utility Functions.

```
package template

import org.json.simple.JSONArray
import org.json.simple.JSONObject
import java.io.File
import com.github.kittinunf.fuel.Fuel
import org.json.simple.parser.JSONParser
import java.io.StringReader
import java.lang.Exception
import template.Util.log


object Util {

    var fw = File("/Users/jganguly/Sites/josh/ktfas/fas/debug.txt")

    // log
    fun log(str: String) {
        fw.appendText(str + "\n")
    }

    // Fuel.post
    fun post (url: String, jobj: String ): String {
        val (request, response, result) = Fuel.post(url)
            .appendHeader("Content-Type", "application/json")
            .body(jobj.toString())
            .response()

        val (payload, error) = result

        // Error handling can be improved
        error?.message?.let { println(it) }

        return response
            .body()
            .asString("application/json")
    }


    // replace only the exact match within the "." delimiters
    fun replace(str: String, toReplace: String) {
        val regex = "\\b${toReplace.replace(".", "\\.")}\\b".toRegex()
// Super important line of code
//        var newstr = str.replace("\\s+".toRegex()," ")
        println(str.replace(regex, toReplace))
    }


}
```

CHAPTER 12

Main

12.1 Main

Listing 12.1 – Main.

```
    println("***template_1***")
    mytest(3)
    val bs = BasicStruct()
    bs.foo(3,4)
    bs.foo('c')
    bs.fooNull()
    println()


    // template_2
    println("***template_2***")
    val a = A(10)
    println(a.id)
    val a2 = A(10,"Josh")
    println(a2.name)
    val foo = Foo("Josh",24)
    println(foo.name)
    //readLine()
    val foo3 = Foo3("Josh",10,"Hyd","TS")
    println()
    ObjectExample.hello()
    //readLine()


    // template_3
    println("***template_3***")
    val bl = Logic()
    bl.foo("Josh")
    val s = bl.foo("Josh",'J','G')
    println(s)
    bl.forloop()
    bl.whileloop()
    println()


    // template_4
    println("***template_4***")
    val c = Coll()
    c.collect()
    println()


    // template_5
    println("***template_5***")
    println()


    // template_6
    println("***template_6***")
    val hof = HOF()
```

```
51      hof.example("Hi","Josh")
52      println()
53
54
55      // template_7
56      println("***template_7***")
57      testApply()
58      println()
59      //readLine()
60
61      // template_8
62      // https://www.baeldung.com/kotlin-generics
63      println("***template_8***")
64      // Covariant
65      val pp1 = ParameterizedProducer(302.33)       // double
66      val pp2: ParameterizedProducer<Number> = pp1 // assigned to supertype number
67      println("Generic Out:" + pp2.getValue())
68
69      // Contravariant
70      val pp3 = ParameterizedConsumer<Number>()     // Super class Number
71      val pp4: ParameterizedConsumer<Double> = pp3 // assigned to subtype Double
72      val pp5: String = pp4.toString(3.45)
73      println("Generic In:" + pp5.toString())
74      println()
75
76      // template_9
77      println("***template_9***")
78      exampleBlocking()
79      exampleBlockingDispatcher()
80      exampleLaunchCoroutineScope()
81      parallel()
82      serial()
83
84  }
```

CHAPTER 13

Output

13.1 output

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/bin/java "-java
Hello World!

***template_1***
The value of x = 4
sum of x and y is 7
10 is Int
Int
kotlin.Int
c
b is null

***template_2***
10
Josh
In init of base class:id = 1000

In init of base class:id = 1000
In Foo2

***template_3***
I am Josh
Josh is a string
Gosh
1
2
3
4
5
6
7
8
9
10
0
1
2
3
4
5
6
7
```

```
8
9
2
4
6
8
10
10
9
8
7
6
5
4
3
2
1
5 7 val = 0
val = 1
val = 2
val = 3
val = 4
val = 5
val = 6
val = 7
val = 8
val = 9
val = 10

***template_4***
New Zealand
India
USA
China
New Zealand
India
USA
China
New Zealand
Nepal
[Banana, Kiwifruit, Mango, Apple, Grape]
[Apple, Apricot]
[Lion, Cat, Hippo, Dog]
CHCH 389700
DUN 118500
[Mango, Guava, Apple]
10
[Apple, Apricot]
[SciFi]
[{id=101, name=Jaideep, status=true}]
true
```

```
true
total: 15
mul: 120

***template_5***

***template_6***
Hi Josh!

***template_7***
Pers(name=Tim, age=24)
Pers(name=Tim, age=22)
Pers(name=Tim, age=25)
Pers(name=Tim, age=26)
Pers(name=Tim, age=36)

***template_8***
Generic Out:302.33
Generic In:3.45

***template_9***
one
two
three
one - from thread DefaultDispatcher-worker-1
two - from thread DefaultDispatcher-worker-1
three - from thread main
one - from thread main
three - from thread main
two - from thread pool-1-thread-1
10
1000
20
2000
10
20
30
asyn   c/await result = 6000
Time taken: 3011
10
20
30
async/await result = 6000
Time taken: 6019

Process finished with exit code 0
```