Fresher Android

Kotlin Advance Concept

Lesson Objectives

- Destructuring Declarations
- Collections
- Ranges
- Type Checks and Casts
- This expressions
- Equality
- Operator overloading
- Null Safety
- Exceptions
- Calling Java from Kotlin
- Calling Kotlin from Java
- Documenting Kotlin Code

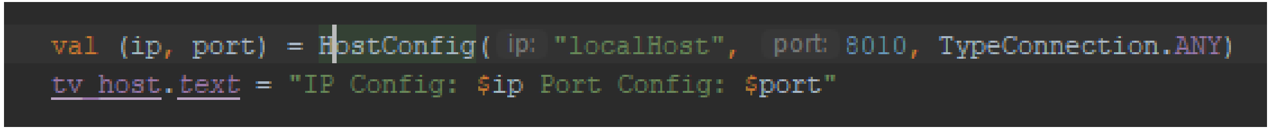
Section 1

FUNCTIONS

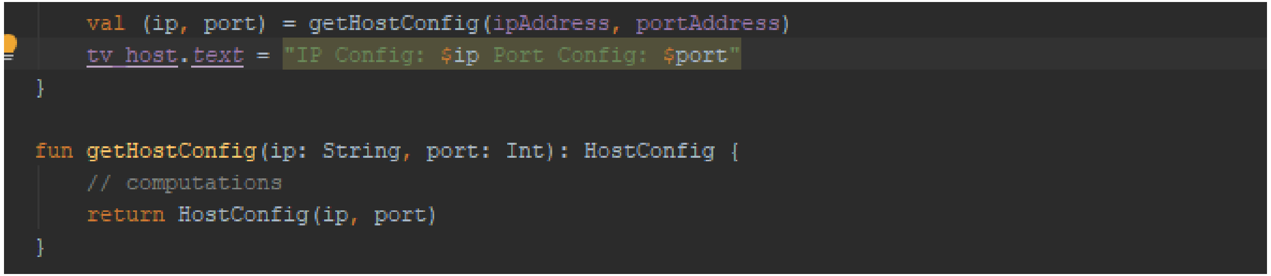
#1. Destructuring Declarations

- Kotlin includes more functions of other language programming
- Kotlin allow creates simultaneous more variable

⇒ **This syntax is called a destructuring declaration**



```
  val (ip, port) = HostConfig( ip: "localHost",  port: 8010, TypeConnection.ANY)
  tv_host.text = "IP Config: $ip Port Config: $port"
```



```
    val (ip, port) = getHostConfig(ipAddress, portAddress)
    tv_host.text = "IP Config: $ip Port Config: $port"
}

fun getHostConfig(ip: String, port: Int): HostConfig {
    // computations
    return HostConfig(ip, port)
}
```

Refer: https://kotlinlang.org/docs/reference/multi-declarations.html

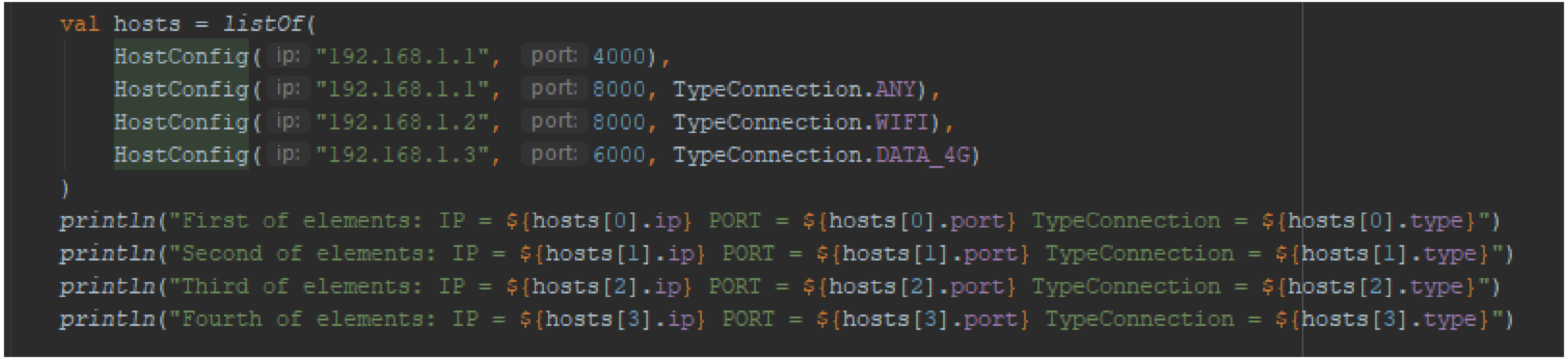
#2. Collections

- Collections are a common concept for most programming languages.
- A collection usually contains a number of objects (this number may also be zero) of the same type. Objects in a collection are called elements or items
- The Kotlin Standard Library provides implementations for basic collection types:
    1. List
    2. Set
    3. Map
- A pair of interfaces represent each collection type:
    1. A read-only interface that provides operations for accessing collection elements.
    2. A mutable interface that extends the corresponding read-only interface with write operations: adding, removing, and updating its elements.

Refer: https://kotlinlang.org/docs/reference/collections-overview.html

#2.1 Collection - List

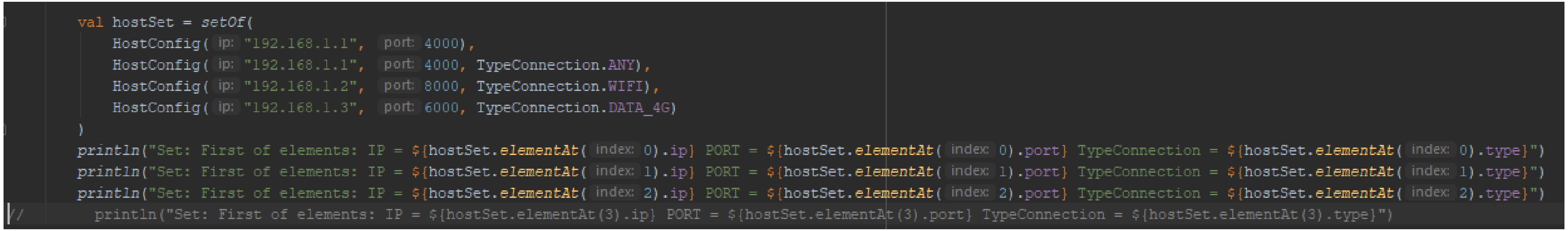
- List is an ordered collection with access to elements by indices – integer numbers that reflect their position



```
val hosts = listOf(
        HostConfig( ip: "192.168.1.1",  port: 4000),
        HostConfig( ip: "192.168.1.1",  port: 8000, TypeConnection.ANY),
        HostConfig( ip: "192.168.1.2",  port: 8000, TypeConnection.WIFI),
        HostConfig( ip: "192.168.1.3",  port: 6000, TypeConnection.DATA_4G)
)
println("First of elements: IP = ${hosts[0].ip} PORT = ${hosts[0].port} TypeConnection = ${hosts[0].type}")
println("Second of elements: IP = ${hosts[1].ip} PORT = ${hosts[1].port} TypeConnection = ${hosts[1].type}")
println("Third of elements: IP = ${hosts[2].ip} PORT = ${hosts[2].port} TypeConnection = ${hosts[2].type}")
println("Fourth of elements: IP = ${hosts[3].ip} PORT = ${hosts[3].port} TypeConnection = ${hosts[3].type}")
```

#2.2 Collection - Set

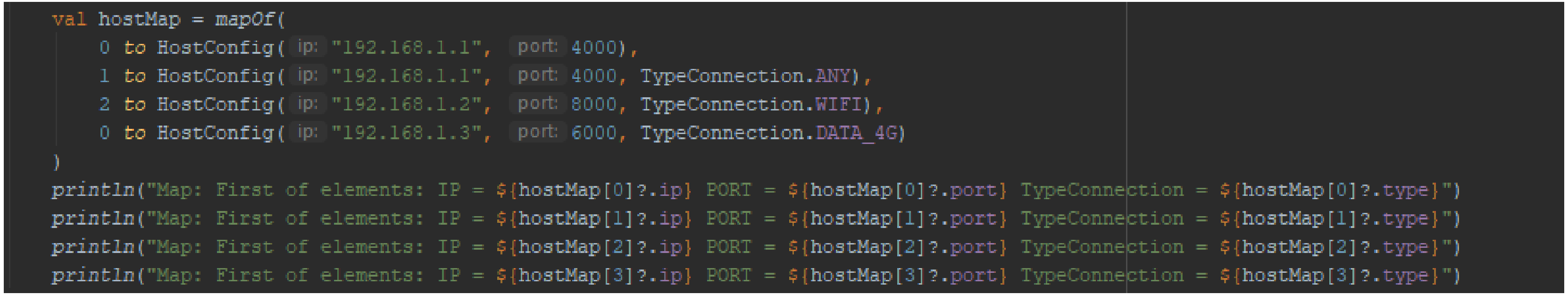
- Set is a collection of unique elements. The order of set elements has no significance



```
val hostSet = setOf(
    HostConfig( ip: "192.168.1.1",  port: 4000),
    HostConfig( ip: "192.168.1.1",  port: 4000, TypeConnection.ANY),
    HostConfig( ip: "192.168.1.2",  port: 8000, TypeConnection.WIFI),
    HostConfig( ip: "192.168.1.3",  port: 6000, TypeConnection.DATA_4G)
)
println("Set: First of elements: IP = ${hostSet.elementAt( index: 0).ip} PORT = ${hostSet.elementAt( index: 0).port} TypeConnection = ${hostSet.elementAt( index: 0).type}")
println("Set: First of elements: IP = ${hostSet.elementAt( index: 1).ip} PORT = ${hostSet.elementAt( index: 1).port} TypeConnection = ${hostSet.elementAt( index: 1).type}")
println("Set: First of elements: IP = ${hostSet.elementAt( index: 2).ip} PORT = ${hostSet.elementAt( index: 2).port} TypeConnection = ${hostSet.elementAt( index: 2).type}")
//    println("Set: First of elements: IP = ${hostSet.elementAt(3).ip} PORT = ${hostSet.elementAt(3).port} TypeConnection = ${hostSet.elementAt(3).type}")
```

#2.3 Collection - Map

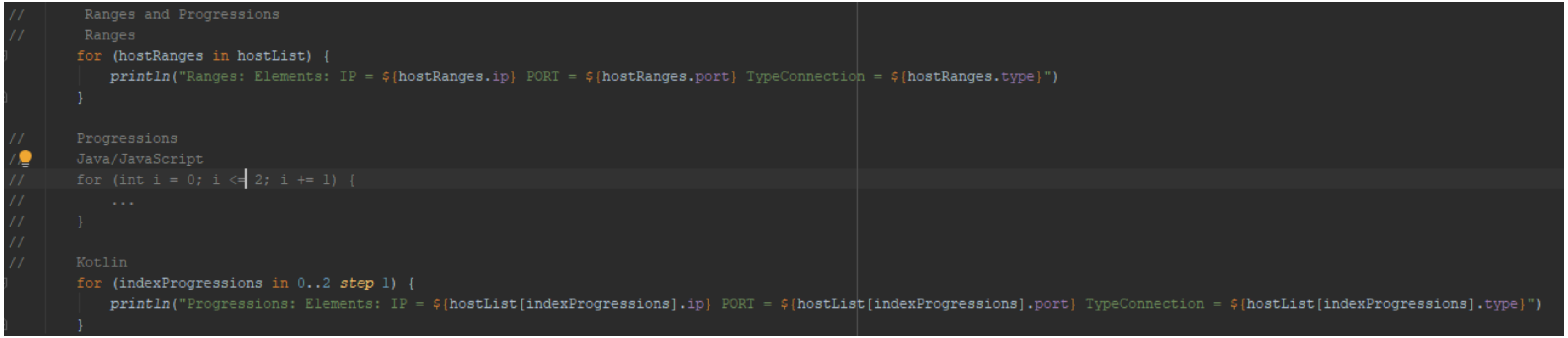
- Map is a set of key-value pairs. Keys are unique, and each of them maps to exactly one value. The values can be duplicates.



```
val hostMap = mapOf(
    0 to HostConfig( ip: "192.168.1.1",  port: 4000),
    1 to HostConfig( ip: "192.168.1.1",  port: 4000, TypeConnection.ANY),
    2 to HostConfig( ip: "192.168.1.2",  port: 8000, TypeConnection.WIFI),
    0 to HostConfig( ip: "192.168.1.3",  port: 6000, TypeConnection.DATA_4G)
)
println("Map: First of elements: IP = ${hostMap[0]?.ip} PORT = ${hostMap[0]?.port} TypeConnection = ${hostMap[0]?.type}")
println("Map: First of elements: IP = ${hostMap[1]?.ip} PORT = ${hostMap[1]?.port} TypeConnection = ${hostMap[1]?.type}")
println("Map: First of elements: IP = ${hostMap[2]?.ip} PORT = ${hostMap[2]?.port} TypeConnection = ${hostMap[2]?.type}")
println("Map: First of elements: IP = ${hostMap[3]?.ip} PORT = ${hostMap[3]?.port} TypeConnection = ${hostMap[3]?.type}")
```

#3. Ranges and Progressions

- Kotlin lets you easily create ranges of values using the *rangeTo()* function from the *kotlin.ranges* package and its operator form “..”
- Usually, *rangeTo()* is complemented by in or !in functions.

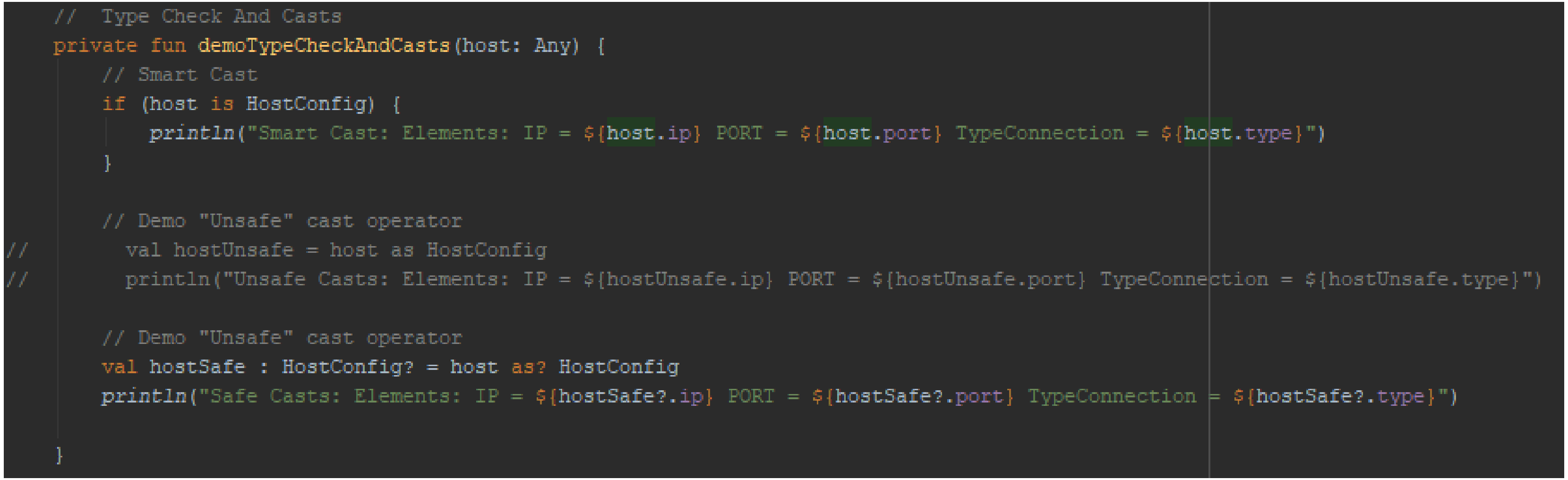


```
//      Ranges and Progressions
//      Ranges
       for (hostRanges in hostList) {
           println("Ranges: Elements: IP = ${hostRanges.ip} PORT = ${hostRanges.port} TypeConnection = ${hostRanges.type}")
       }

//     Progressions
//     Java/JavaScript
//     for (int i = 0; i <= 2; i += 1) {
//         ...
//     }
//
//     Kotlin
       for (indexProgressions in 0..2 step 1) {
           println("Progressions: Elements: IP = ${hostList[indexProgressions].ip} PORT = ${hostList[indexProgressions].port} TypeConnection = ${hostList[indexProgressions].type}")
       }
```

Refer: https://kotlinlang.org/docs/reference/ranges.html

#4. Type Checks and Casts

- In Kotlin, We can check whether an object conforms to a given type at runtime by using the *is* operator or its negated form *!is*

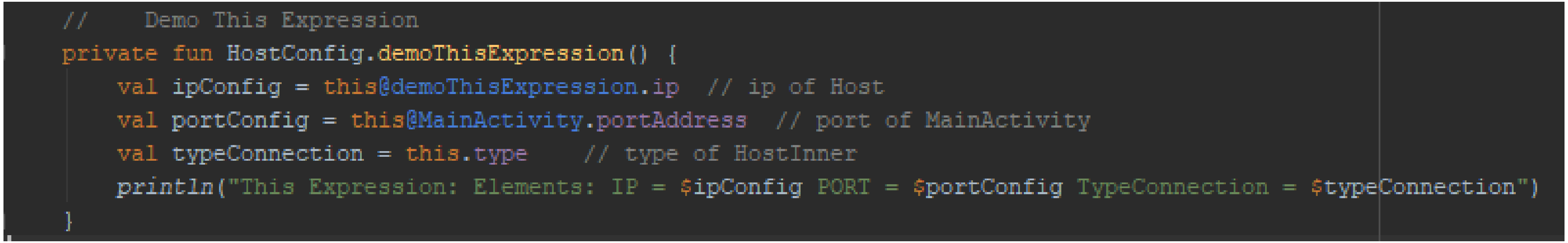


```
    //  Type Check And Casts
    private fun demoTypeCheckAndCasts(host: Any) {
        // Smart Cast
        if (host is HostConfig) {
            println("Smart Cast: Elements: IP = ${host.ip} PORT = ${host.port} TypeConnection = ${host.type}")
        }

        // Demo "Unsafe" cast operator
//          val hostUnsafe = host as HostConfig
//          println("Unsafe Casts: Elements: IP = ${hostUnsafe.ip} PORT = ${hostUnsafe.port} TypeConnection = ${hostUnsafe.type}")

        // Demo "Unsafe" cast operator
        val hostSafe : HostConfig? = host as? HostConfig
        println("Safe Casts: Elements: IP = ${hostSafe?.ip} PORT = ${hostSafe?.port} TypeConnection = ${hostSafe?.type}")

    }
```

Refer: https://kotlinlang.org/docs/reference/typecasts.html

#5. This expressions

- To denote the current receiver, we use this expressions:
- In a member of a class, this refers to the current object of that class.
- In an extension function or a function literal with receiver this denotes the receiver parameter that is passed on the left-hand side of a dot.

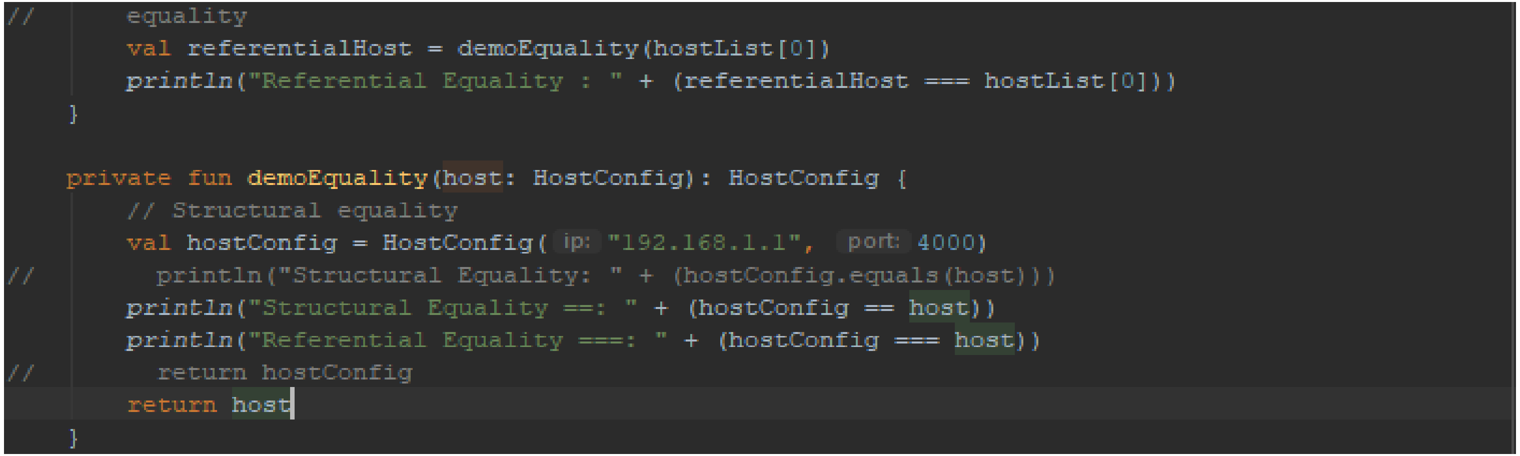


```
//    Demo This Expression
private fun HostConfig.demoThisExpression() {
    val ipConfig = this@demoThisExpression.ip  // ip of Host
    val portConfig = this@MainActivity.portAddress  // port of MainActivity
    val typeConnection = this.type    // type of HostInner
    println("This Expression: Elements: IP = $ipConfig PORT = $portConfig TypeConnection = $typeConnection")
}
```

Refer: https://kotlinlang.org/docs/reference/this-expressions.html#this-expression

#6. Equality

- In Kotlin there are two types of equality:
- Structural equality (a check for equals())
- Referential equality (two references point to the same object)



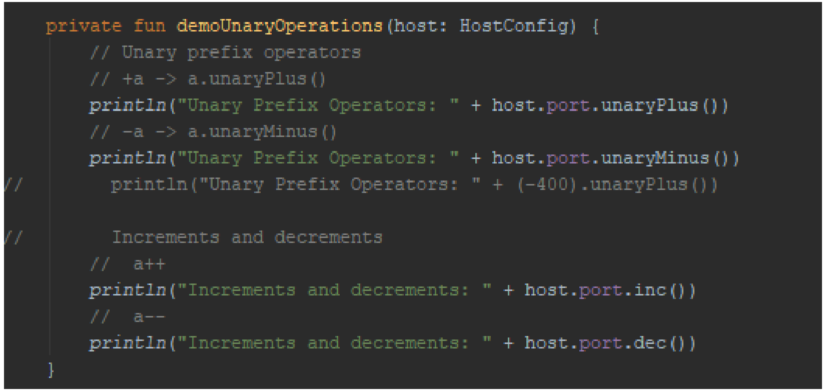
```
//      equality
        val referentialHost = demoEquality(hostList[0])
        println("Referential Equality : " + (referentialHost === hostList[0]))
    }

    private fun demoEquality(host: HostConfig): HostConfig {
        // Structural equality
        val hostConfig = HostConfig( ip: "192.168.1.1",  port: 4000)
//        println("Structural Equality: " + (hostConfig.equals(host)))
        println("Structural Equality ==: " + (hostConfig == host))
        println("Referential Equality ===: " + (hostConfig === host))
//        return hostConfig
        return host
    }
```

Refer: https://kotlinlang.org/docs/reference/equality.html#equality

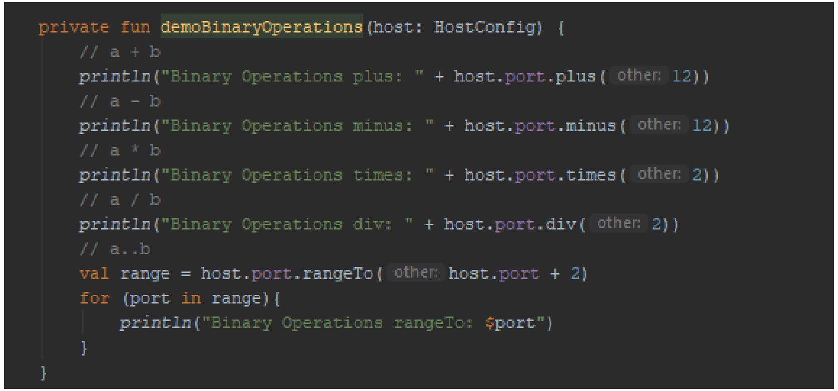
#7. Operator overloading

- Kotlin allows us to provide implementations for a predefined set of operators on our types:

- *Unary operations*
- *Binary operations*



```
    private fun demoUnaryOperations(host: HostConfig) {
        // Unary prefix operators
        // +a -> a.unaryPlus()
        println("Unary Prefix Operators: " + host.port.unaryPlus())
        // -a -> a.unaryMinus()
        println("Unary Prefix Operators: " + host.port.unaryMinus())
//          println("Unary Prefix Operators: " + (-400).unaryPlus())

//          Increments and decrements
        //  a++
        println("Increments and decrements: " + host.port.inc())
        //  a--
        println("Increments and decrements: " + host.port.dec())
    }
```

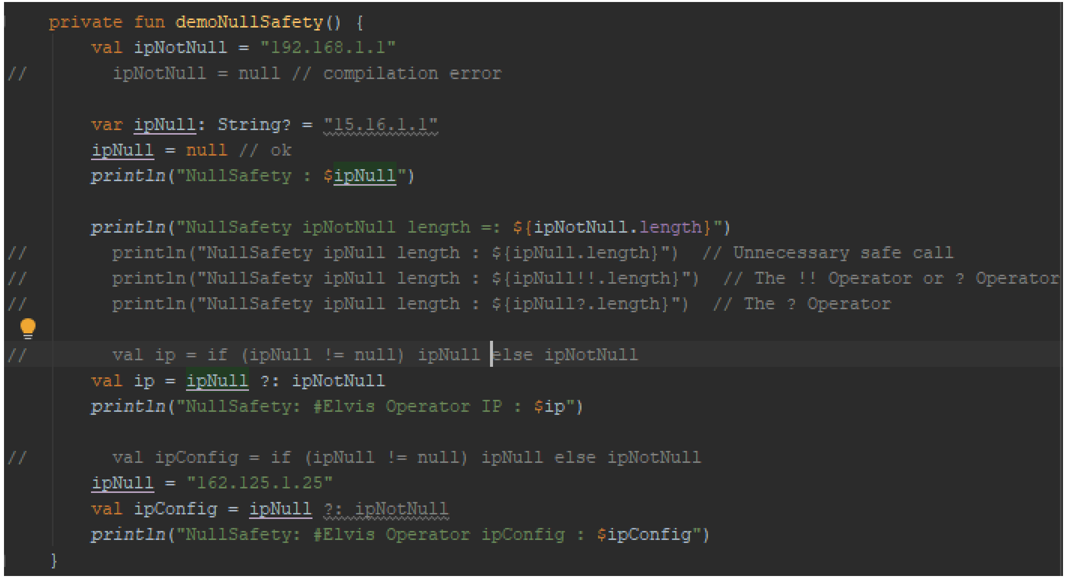


```
    private fun demoBinaryOperations(host: HostConfig) {
        // a + b
        println("Binary Operations plus: " + host.port.plus( other: 12))
        // a - b
        println("Binary Operations minus: " + host.port.minus( other: 12))
        // a * b
        println("Binary Operations times: " + host.port.times( other: 2))
        // a / b
        println("Binary Operations div: " + host.port.div( other: 2))
        // a..b
        val range = host.port.rangeTo( other: host.port + 2)
        for (port in range){
            println("Binary Operations rangeTo: $port")
        }
    }
```

Refer: https://kotlinlang.org/docs/reference/operator-overloading.html#operator-overloading

#7. Operator overloading

- Kotlin's type system is aimed at eliminating the danger of null references from code, also known as the *The Billion Dollar Mistake*

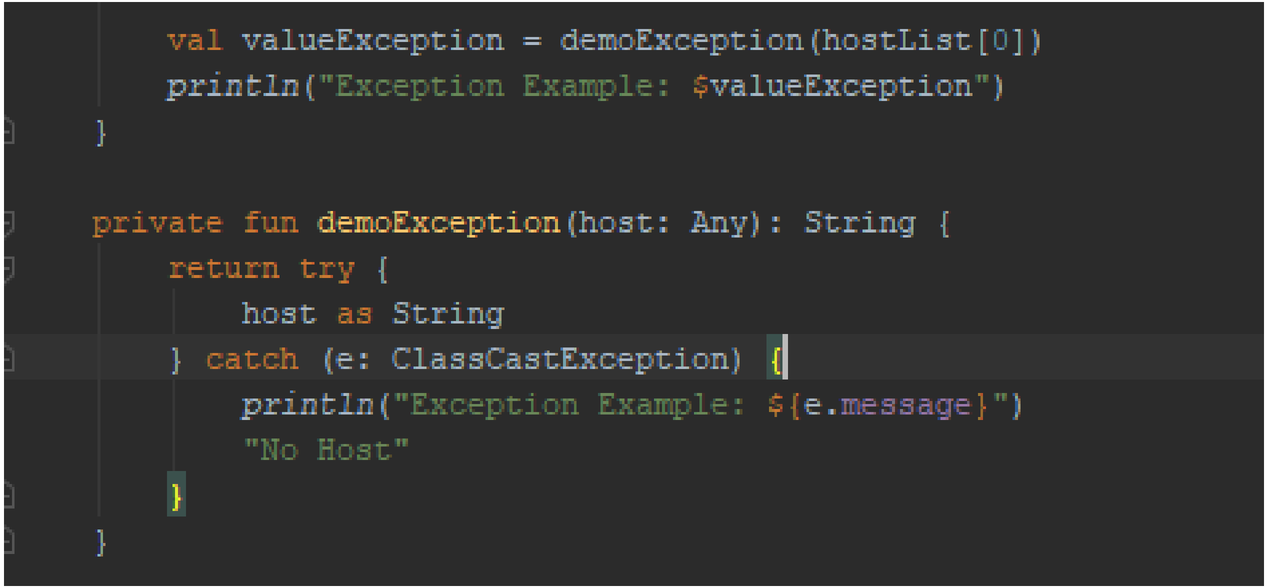


```
    private fun demoNullSafety() {
        val ipNotNull = "192.168.1.1"
//          ipNotNull = null // compilation error

        var ipNull: String? = "15.16.1.1"
        ipNull = null // ok
        println("NullSafety : $ipNull")

        println("NullSafety ipNotNull length =: ${ipNotNull.length}")
//          println("NullSafety ipNull length : ${ipNull.length}")  // Unnecessary safe call
//          println("NullSafety ipNull length : ${ipNull!!.length}")  // The !! Operator or ? Operator
//          println("NullSafety ipNull length : ${ipNull?.length}")  // The ? Operator

//          val ip = if (ipNull != null) ipNull else ipNotNull
        val ip = ipNull ?: ipNotNull
        println("NullSafety: #Elvis Operator IP : $ip")

//          val ipConfig = if (ipNull != null) ipNull else ipNotNull
        ipNull = "162.125.1.25"
        val ipConfig = ipNull ?: ipNotNull
        println("NullSafety: #Elvis Operator ipConfig : $ipConfig")
    }
```

Refer: https://kotlinlang.org/docs/reference/null-safety.html

#8. Exceptions

- Exceptions pretty much work like they do in Coding. You throw (raise) one with *throw*

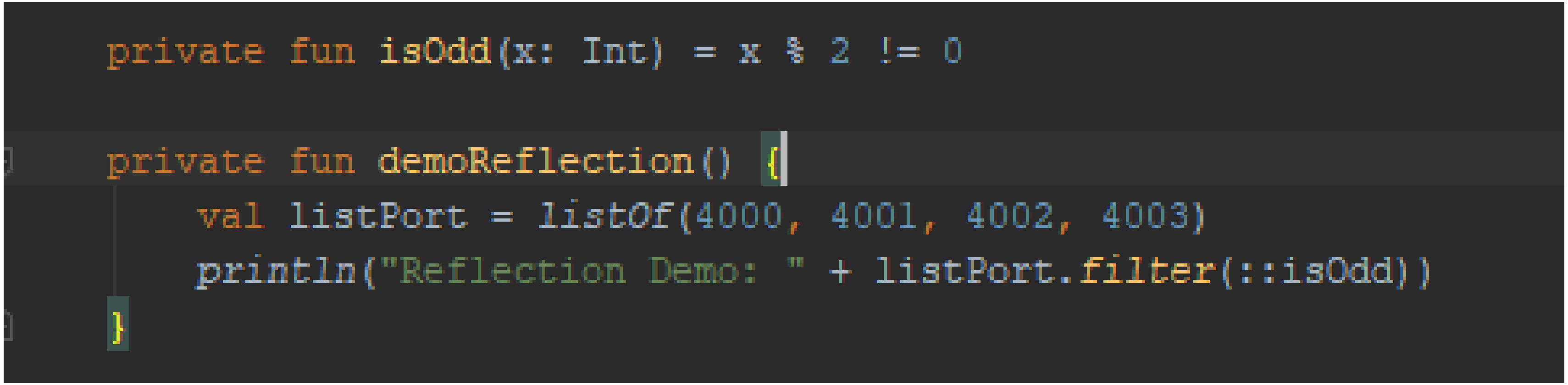


```
        val valueException = demoException(hostList[0])
        println("Exception Example: $valueException")
    }

    private fun demoException(host: Any): String {
        return try {
            host as String
        } catch (e: ClassCastException) {
            println("Exception Example: ${e.message}")
            "No Host"
        }
    }
```

Refer: https://kotlinlang.org/docs/tutorials/kotlin-for-py/exceptions.html

#9. Reflection

- Reflection is a set of language and library features that allows for introspecting the structure of your own program at runtime.

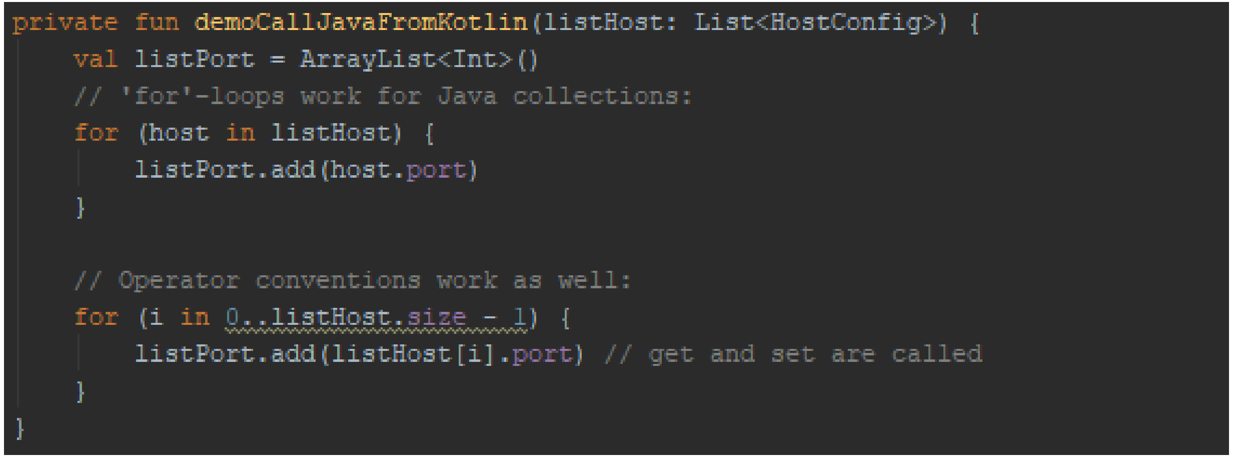


```
private fun isOdd(x: Int) = x % 2 != 0

private fun demoReflection() {
    val listPort = listOf(4000, 4001, 4002, 4003)
    println("Reflection Demo: " + listPort.filter(::isOdd))
}
```

Refer: https://kotlinlang.org/docs/reference/reflection.html#reflection

#10. Calling Java from Kotlin

- Kotlin is designed with Java Interoperability in mind. Existing Java code can be called from Kotlin in a natural way, and Kotlin code can be used from Java rather smoothly as well. In this section we describe some details about calling Java code from Kotlin.

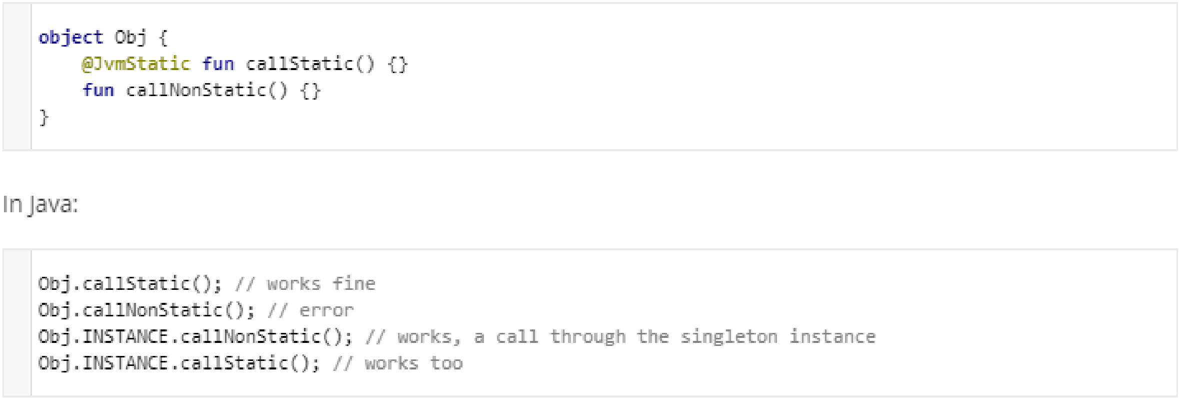


```
private fun demoCallJavaFromKotlin(listHost: List<HostConfig>) {
    val listPort = ArrayList<Int>()
    // 'for'-loops work for Java collections:
    for (host in listHost) {
        listPort.add(host.port)
    }

    // Operator conventions work as well:
    for (i in 0..listHost.size - 1) {
        listPort.add(listHost[i].port) // get and set are called
    }
}
```

Refer: https://kotlinlang.org/docs/reference/java-interop.html

#11. Calling Kotlin from Java

- Kotlin code can be easily called from Java.



```
object Obj {
    @JvmStatic fun callStatic() {}
    fun callNonStatic() {}
}
```

In Java:

```
Obj.callStatic(); // works fine
Obj.callNonStatic(); // error
Obj.INSTANCE.callNonStatic(); // works, a call through the singleton instance
Obj.INSTANCE.callStatic(); // works too
```

Refer: https://kotlinlang.org/docs/reference/java-to-kotlin-interop.html

Functions. Summary

- Destructuring Declarations
- Collections
- Ranges
- Type Checks and Casts
- This expressions
- Equality
- Operator overloading
- Null Safety
- Exceptions
- Calling Java from Kotlin
- Calling Kotlin from Java
- Documenting Kotlin Code
- Q&A

Assignment

- Create a program with some functions as below:

1. Users input a list Host Config (ip, port, type connection) (min 4 items). Each Entered Host Config will generate 3 Host Configs include ip, type connection and port + 1 automatically.
2. Show information of all host config in list with one of the following conditions:
   - IP
   - Port
   - Type connection
   - Host Config
   - <None>

References

- https://kotlinlang.org/docs/reference/multi-declarations.html
- https://kotlinlang.org/docs/reference/collections-overview.html
- https://kotlinlang.org/docs/reference/ranges.html
- https://kotlinlang.org/docs/reference/typecasts.html
- https://kotlinlang.org/docs/reference/this-expressions.html#this-expression
- https://kotlinlang.org/docs/reference/equality.html#equality
- https://kotlinlang.org/docs/reference/operator-overloading.html#operator-overloading
- https://kotlinlang.org/docs/reference/null-safety.html
- https://kotlinlang.org/docs/tutorials/kotlin-for-py/exceptions.html
- https://kotlinlang.org/docs/reference/reflection.html#reflection
- https://kotlinlang.org/docs/reference/java-interop.html
- https://kotlinlang.org/docs/reference/java-to-kotlin-interop.html

Lesson Summary

- *Function*
- *Assignment*

Thank you