# SIMPLE KOTLIN

# Variable declaration

Kotlin sử dụng 2 cách chính để khai báo một biến là **val** và **var**

* Dùng val cho các biến có giá trị không bao giờ thay đổi (tương tự như const bên java)
* Dùng var cho các biến có thể thay đổi giá trị được

var number: Int = 10

ngoài Int thì trong Kotlin cũng có một số kiểu dữ liệu khác như: Byte, Short, Long, Float, Double, String

# Null safety

Ở một số ngôn ngữ khác, một biến được khai báo và không gán bất cứ giá trị nào thì sẽ mang giá trị null. Tuy nhiên trong Kotlin một biến không thể mang giá trị mặc định là null được.

Để gán giá trị null cho một biến nó phải được khai báo là nullable(bằng cách khai báo thêm dấu ?)

Val languageName: String? = null

Với khai báo “**?**” bạn có thể gán biến languageNam cho một giá trị String nào đó hoặc cho một giá trị null mà vẫn biên dịch chính xác

Nếu không xử lý biến nullable hợp lệ sẽ dễ mắc phải NullPointerException và chương trình bị crash

# Conditionals

Câu lệnh if else trong Kotlin tương đối giống với trong java. Trong java khi sử dụng câu lệnh điều kiện bạn có thể bị mắc lỗi các lệnh dễ lặp lại(ví dụ lệnh print ra màn hình trong các khối if, else). Để giải quyết vấn đề này thì Kotlin cho phép giải quyết vấn đề theo cách bên dưới

Var answerString:String = if(answer == true){

“You are right!!”;

}else {

“You are wrong!!”

}

println(answerString);

Lưu ý là mỗi câu lệnh điều kiện đều sẽ trả về một kết quả trong dòng lệnh cuối cùng của nó, thế nên bạn không cần phải gọi return để trả về kết quả.

Nếu trường hợp câu lệnh điều kiện của bạn phát triển một cách phức tạp hơn, có thể xem xét thay thế câu lệnh if-else bằng lệnh **when** được ví dụ như sau:

Var type:String = when{

Age < 6 -> “You are a kid.”

Age < 18 -> “You are a teenager.”

Else -> “You are an adult.”

}

println(type)

các lệnh trước -> thể hiện cho điều kiện trong if và các lệnh sau -> thể hiện cho câu lệnh được thực hiện khi điều kiện đúng

# Functions

Để khai báo function trong Kotlin ta sử dụng từ khóa **fun** theo sau đó là tên hàm, tiếp đến xác định loại dữ liệu của input và cuối cùng là kiểu dữ liệu trả về(output).

fun isEvenNumber(number: Int): String{

val returnString:String = if(number%2 == 0){

“This is even number”

}

Else {

“This is not even number”

}

Return returnString

}

Để gọi hàm đã định nghĩa thì chỉ cần gọi tên hàm với các input truyền vào tương tự trong java.

# Simplifying function declarations

Với ví dụ trên có một cách để làm đơn giản hóa hàm đó là bỏ luôn phần biến cục bộ *returnString*

fun isEvenNumber(number: Int):String{

return if (number % 2 == 0){

“This is even number”

}

Else {

“This is not even number”

}

}

Hoặc có một cách khác là thay phần trả về return bằng toán tử gán

fun isEvenNumber(number: Int):String=if(number % 2 == 0) {

“This is even number”

}else{

“This is not even number”

}

# Anonymous functions

Không phải tất cả các hàm đều được định danh bởi một cái tên. Một số hàm được định nghĩa trực tiếp bởi input và output. Các hàm này được gọi với cái tên là anonymous function

Bạn có thể tham chiếu tới anonymous function bằng cách sử dụng 1 tham chiếu gọi là anonymous function later.

Val stringLengthFunc: (String) -> Int = {input -> input.length}

Val stringLength: Int = stringLengthFunc(“Android”)

Đúng như tên gọi của nó Anonymous function có thể chứa bất cứ các biểu thức nào. Giá trị trả về của hàm này sẽ là kết quả của câu lệnh cuối cùng.

Với ví dụ trên ta thấy String được đặt trong ngoặc tròn cho thấy input đầu vào là một biến dạng String và output trong trường hợp này là một giá trị Int. Vì tham số đầu vào chưa được đặt tên nên ta sẽ gọi tên của nó mặc định là input. Để gọi hàm thì ta sử dụng tên gọi như bình thường để gọi thực hiện hàm đó

# Higher order functions

Một hàm có thể sử dụng một hàm khác như tham số đầu vào, khi đó hàm đó được gọi là higher order function. Đây là một hình thức khá là hữu dụng và thường được dùng trong callback interface(java)

fun lengthString(str: String, getLength: (String) -> Int): Int{

return getLength(str)

}

Để gọi thực hiện cho hàm lengthString bạn có thể truyền vào một String và một hàm thỏa mãn với input và output ví dụ như bên dưới

lengthString(“Android”,{input -> input.length})

nếu mà tham số đầu vào cuối cùng là một anonymous function, bạn có thể viết nó ra bên ngoài dấu ngoặc đơn như bên dưới

lengthString(“Android”) {input -> input.length}

# Class

Để định nghĩa một class có thể sử dụng từ khóa class để khải báo

## Properties

Nếu các thuộc tính bên trong class chỉ khai báo như biến bình thường thì coi như nó đang ở trạng thái public và có thể được truy cập vào từ bên ngoài class và đặc biệt lưu ý là thuộc tính bên trong class không thể được gán lại giá trị từ bên ngoài

class Car{

val wheels = listOf<Wheel>()

}

Val car = Car() // constructor a Car

Val wheels = car.wheels //retrieve the wheels value from the Car

Nếu muốn tùy chỉnh cho các thuộc tính trong class bạn có thể định nghĩa một constructor sẽ chỉ định cách mà các thuộc tính trong class được khởi tạo

class Car(val wheels:List<Wheel>)

# Class functions and encapsulation

Nếu bạn muốn điều chỉnh cách để một thuộc tính được tham chiếu, bạn có thể cung cấp tùy chỉnh getter và setter. Ví dụ nếu bạn muốn thể hiện getter của một thuộc tính trong khi giới hạn quyền truy cập vào setter của nó bạn có thể tiến hành chỉ định private cho setter. Ví dụ như bên dưới

Class Car(val wheels:List<Wheel>){

private val doorLock: DoorLock = …

var gallonOfFuelInTank: Int = 15

private set

fun unlockDoor(key: Key): Boolean{

// return true if key is valid for door lock, false otherwise

}

}

# USE COMMON KOTLIN PATTERNS WITH ANDROID

# Work with fragments

## Inheritance

Như đã đề cập ở phần trên, bạn có thể khai báo một lớp trong kotlin với từ khóa class. Nếu một class con kế thừ từ một class nào đó thì có thể sử dụng toán tử “:” để thể hiện tính kế thừa

Class LoginFragment: Fragment()

Khi đó với class LoginFragment bạn có thể thực hiện việc ghi đè lại các phương thức có trong lớp cha là Fragment

Override fun onCreateView(

inflater: LayoutInflater,

container: ViewGroup?,

savedInstanceState: Bundle?): View?{

return inflater.inflate(R.layout.login\_fragment, container,false)

}

Để tham chiếu tới một hàm trong lớp cha, bạn có thể sử dụng từ khóa super như bên dưới

Override fun onViewCreated(view: View, savedInstanceState: Bundle?){

super.onViewCreated(view, savedInstanceState)

}

# Nullability and initialization

Như đã từng để cập ở trước thì trong Kotlin không cho phép khai báo biến mà không khởi tạo giá trị. Tức là khi bạn lấy ví dụ cho một class, bạn có thể tham chiếu tới bất cứ thuộc tính nào có thể truy cập đến được của nó.

Đối tượng View trong Fragment, tuy nhiên lại chưa được dán vào cho tới khi gọi fragment#onCreateView, vậy nên bạn sẽ cần phải có một cách nào đó để trì hoãn lại việc khởi tạo cho View. Và lateinit là một cách để thực hiện được điều đó. Khi sử dụng lateinit, bạn cần phải khởi tạo các thuộc tính sớm nhất khi có thể.

Class LoginFragment: Fragment(){

Private lateinit var usernameEditText: EditText

Private lateinit var passwordEditText: EditText

Private lateinit var loginButton: Button

Private lateinit var statusTextView: TextView

Override fun onViewCreated(view: View, saveInstanceState: Bundle?){

Super.onViewCreated(view, saveInstanceState)

usernameEditText = view.findViewById(R.id.username\_edit\_text)

…

}

}

# SAM Conversion

Bạn có thể lắng nghe sự kiện click trong android bằng cách implement *OnclickListener* interface. Đối tượng Button có chứa một hàm setOnClickListener() phải implement từ class OnClickListener

Ngoài ra, OnClickListener cũng có một phương thức trừu tượng là onClick() mà bạn cần implement. Bởi vì phương thức setOnClickListener() luôn lấy OnClickListener làm đối số và vì OnClickListener luôn có cùng một phương thức trừu tượng. Việc triển khai này có thể được biểu diễn bằng hàm ẩn danh trong Kotlin. Quá trình này được biết đến như là một cách chuyển đổi phương thức trừu tượng đơn hay với tên gọi khác là phương thức chuyển đổi SAM.

SAM conversion làm cho code của bạn trở nên sạch hơn đáng kể

loginButton.setOnClickListener{

val authSuccessful: Boolean = viewModel.authenticate

(usernameEditText.text.toString(),password.text.toString())

if(authSuccessful){

//navigate to next screen

}else{

StatusTextView.text = requireContext().getString(R.string.auth\_failed)

}

}

# Companion objects(\*)

Companion object tương tự với đối tượng static trong java, nó cung cấp một cơ chế để định nghĩa các biến và hàm được liên kết về mặt khái niệm với một loại dữ liệu nhưng lại không bị trói buộc vào một đối tượng cụ thể.

Class LoginFragment: Fragment(){

Companion object{

Private const val TAG = “LoginFragment”

}

}

Bạn cũng có thể định nghĩa cho TAG ở bên trên cùng của file. Companion object giúp kết nối các biến, hàm, và sự định nghĩa của một lớp mà không đề cập đến bất cứ trường hợp cụ thể nào của lớp đó.

# Property delegation(\*)

Khi khởi tạo các thuộc tính, bạn có thể lặp lại một số mẫu phổ biến của android, ví dụ như kết nối đến ViewModel với một Fragment. Để tránh trùng lặp mã, bạn có thể thử sử dụng cú pháp ủy quyền thuộc tính của Kotlin (Kotlin’s property delegation syntax)

Private val viewModel: LoginViewModel by viewModels()

Ủy quyền thuộc tính cung cấp một implementation chung mà bạn có thể tái sử dụng xuyên suốt app của bạn. Android KTX cung cấp một số đại diện cho thuộc tính của bạn. viewModels ví dụ sẽ lấy lại một ViewModel nằm trong phạm vi của Fragment hiện tại

# Nullability

Tương tự như nullability đã trình bày trong phần SIMPLE KOTLIN(có thể lướt lên để coi lại)

# Interoperability

## Platform type

Nếu bạn dùng Kotlin để tham chiếu tới một thành phần tên là name và chưa được chú thích được định nghĩa trong Java, khi đó trình biên dịch sẽ không biết được liệu rằng String sẽ được tương ứng với ***String*** hay ***String?*** Khi đó sự mơ hồ này sẽ được biểu diễn thông qua platform type là ***String!***

String! Tức là bạn có thể đại diện cho cả String hoặc String? Và trình biên dịch sẽ cho bạn gán cả 2 loại giá trị của kiểu dữ liệu này. Lưu ý rằng nếu bạn gán giá trị null cho kiểu String thì sẽ xuất hiện NullPointerException

Để giải quyết vấn đề này bạn có thể sử dụng chú thích nullability(có thể sử dụng cả trong java và kotlin

Private final @Nullable String accessId;

Public final @NotNull String name;

## Handle nullability(\*)

## Android API changes(\*)