

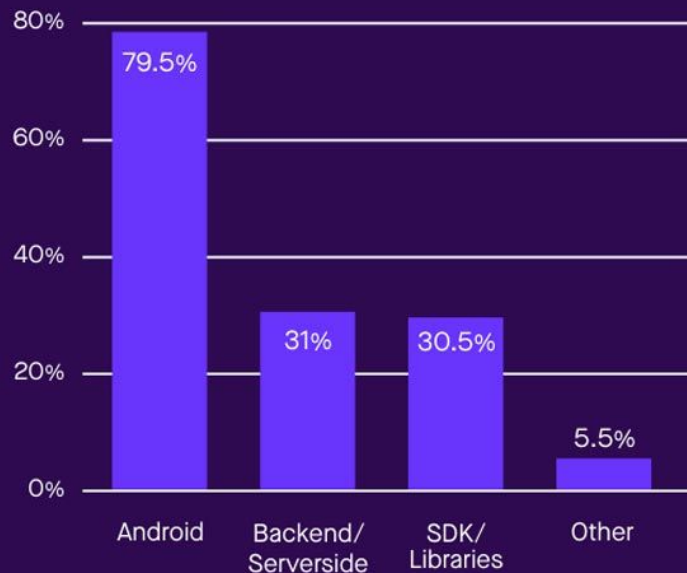
Kotlin

Jialiang Liang & Jianyu Cui

Introduction

Kotlin is a cross-platform, statically typed, general-purpose programming language with type inference. Kotlin is designed to interoperate fully with Java, and the JVM version of Kotlin's standard library depends on the Java Class Library, but type inference allows its syntax to be more concise.

What Kotlin is used for



Names, Binding & Scopes

Kotlin is a statically-typed programming language, that requires declarations to list variable names and specifies a data type.

Kotlin contains two types of keywords: hard keywords, which are excluded from usage of identifiers, and soft keywords, which are allowed to use as identifiers only in a certain context, such as *by*, *catch*, *get*, *finally*, *field*.

Scopes: global and local.

Data Types

1. Numbers – Byte, Short, Int, Long, Float, Double
2. Boolean – True, false
3. Characters
4. Arrays
5. Strings

Type	Bits	Min value	Max value
Long	64	-9223372036854775808	9223372036854775807
Int	32	-2147483648	2147483647
Short	16	-32768	32767
Byte	8	-128	127

Type	Bits	Notes
Double	64	16-17 significant digits (same as float in Python)
Float	32	6-7 significant digits
Char	16	UTF-16 code unit (see the section on strings - in most cases, this is one Unicode character, but it might be just one half of a Unicode character)
Boolean	8	true or false

Nullable types and Non-Null Types

In Kotlin, the type system distinguishes between references that can hold null (nullable references) and those that can not (non-null references). For example, a regular variable of type `String` can not hold null:

```
! var a: String = "abc" // Regular initialization means non-null by default
! a = null // compilation error
```

! Null can not be a value of a non-null type String

To allow nulls, we can declare a variable as nullable string, written *String?*:

```
! var b: String? = "abc" // can be set null
  b = null // ok
  print(b)
```

null

Expressions & Assignment Statement

In Kotlin, *if* is an expression unlike Java (In Java, *if* is a statement). For example,

```
fun main(args: Array<String>) {  
  
    val a = 12  
    val b = 13  
    val max: Int  
  
    max = if (a > b) a else b  
    println("$max")  
}
```

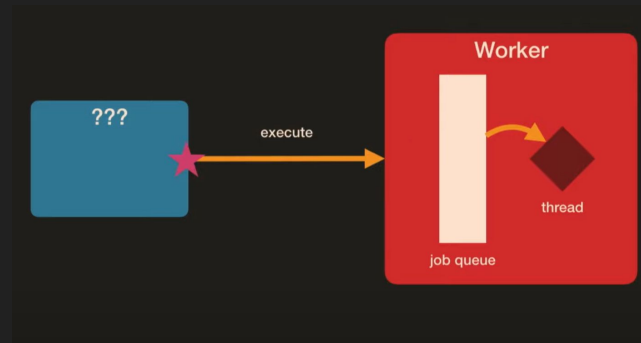
Calls of Java functions that do not defined any return type are not expressions. Kotlin value assignment ($a = 1$) is not an expression in Kotlin, while it is in Java because over there it returns assigned value (in Java you can do $a = b = 2$ or $a = 2 * (b = 3)$). All usages of control structures (*if*, *switch*) in Java are not expressions, while Kotlin allowed *if*, *when* and *try* to return values.

Concurrency

Kotlin runtime does not encourage a classical thread-oriented concurrency model with mutually exclusive code blocks and conditional variables, as this model is known to be error prone and unreliable.

Instead of threads, the Kotlin runtime offers the concept of *Worker*, which is similar to `ExecutorService`

Worker is a concurrently executed control flow streams with an associated request queue. Workers can exchange Kotlin object with another worker, so that each mutable object is owned by a single worker, but the ownership can be transferred.



OO Programming

Kotlin is an object oriented programming language just like Java, which allows us to solve complex problems using objects.

Different from Java, Kotlin allows two types of constructors, the primary and secondary constructor, to initialize the properties of a class.

```
fun main(args: Array<String>){  
    val obj = Student ("Ajeet", 30)  
}  
  
class Student{  
    constructor(name: String, age: Int){  
        println("Student Name: ${name.toUpperCase()}")  
        println("Student Age: $age")  
    }  
}
```

Access Modifiers:

Public

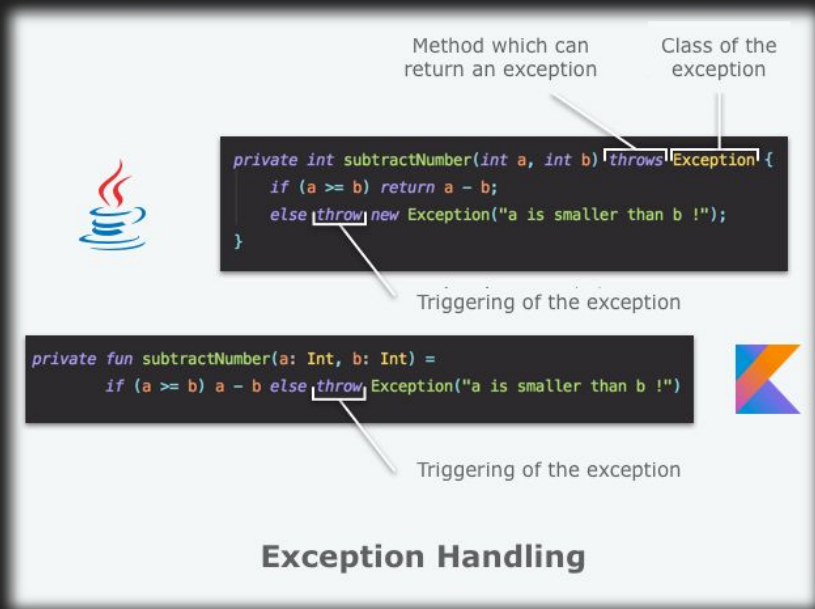
Private

Protected: Visible inside class and subclasses.

Internal: Visible inside the same module.

Exception Handling & Event Handling

- Although Kotlin inherits the concept of exception from Java, it doesn't support checked exceptions like Java.
- In Kotlin, there are only **unchecked** exceptions that are thrown during the runtime execution of the program. All exception classes descend from the class *Throwable*. Kotlin uses the `throw` keyword to throw an exception object.



Method which can return an exception

Class of the exception

```
private int subtractNumber(int a, int b) throws Exception {  
    if (a >= b) return a - b;  
    else throw new Exception("a is smaller than b !");  
}
```

Triggering of the exception

```
private fun subtractNumber(a: Int, b: Int) =  
    if (a >= b) a - b else throw Exception("a is smaller than b !")
```

Triggering of the exception

Exception Handling

Exception Handling & Event Handling

Listener - watches for an event to be fired

Handler - responsible for dealing with the event.

```
cancelImage.setOnClickListener(object : View.OnClickListener { //1
    override fun onClick(v: View?) { // 2
        dismiss()
    }
})
```

{ Kotlin in Android }

```
// in java
Button button = (Button) findViewById(R.id.my_button_id);
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Toast.makeText(MainActivity.this, "clicked", Toast.LENGTH_SHORT).show();
    }
});
```

```
// in kotlin
val button = findViewById(R.id.my_button_id) as Button
button.setOnClickListener { toast("clicked") }
```

Functional Programming

- Higher-Order Functions
- lambda expressions
 - In Kotlin, lambdas can contain multiple statements, which make them useful for more complex tasks than the single-expression lambdas of Python. The last statement must be an expression, whose result will become the return value of the lambda . A lambda expression is enclosed in curly braces, and begins by listing its parameter names and possibly their types (unless the types can be inferred from context).



A diagram showing the Kotlin lambda syntax `{ x: Int, y: Int -> x + y }` on a dark background. A bracket above the text `x: Int, y: Int` is labeled "Parameters". Another bracket above the text `-> x + y` is labeled "Body".

Always between curly brackets!



Lambda Syntax

Q&A

<https://kotlinlang.org/docs>