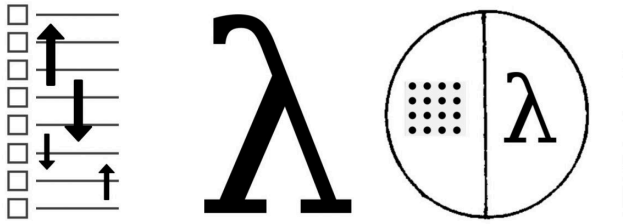


Programming paradigms

L05: Modern Lang – Kotlin

by Michał Szczepanik

Why new lang was needed?

Frist Law of Software Quality

$$E=mc^2$$

$$\text{Error}=(\text{more code})^2$$

otlin

- created by JetBrains (July 2011)
- open sourced (February 2012) :)
- statically typed programming language
- runs on JVM
- can be compiled to JavaScript source code
- or native

Modern & Pragmatic

concise

safe & *fast*

expressive

can be easily mixed with

Let's have **fun** with Hello

```
fun main(args : Array<String>) {
    val scope = "world"
    println("Hello, $scope!")
}
```

How to start ?

```
public class MainActivity extends ActionBarActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }


    @Overri
    public
        //
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }
```

Enter action or option name: Include non-menu actions (⇧⌘A)

Convert Java F Kotlin

Convert Java File to Kotlin File (⌥⇧⌘J) Code

Lang syntax

```
fun sum(a: Int, b: Int): Int {
    return a + b
}

fun sum(a: Int, b: Int) = a + b
```

Lambdas

```
val sum: (Int, Int) -> Int = {x: Int, y: Int -> x+y}

val sum = {x: Int, y: Int -> x + y}

fun apply(i: Int, f: (Int) -> Int) = f(i)

apply(9, {x -> x + 60})

apply(9) {x -> x + 60}
```

Lambdas

```
firstName.validateWith { !it.notEmpty }
lastName.validateWith { !it.notEmpty }
```

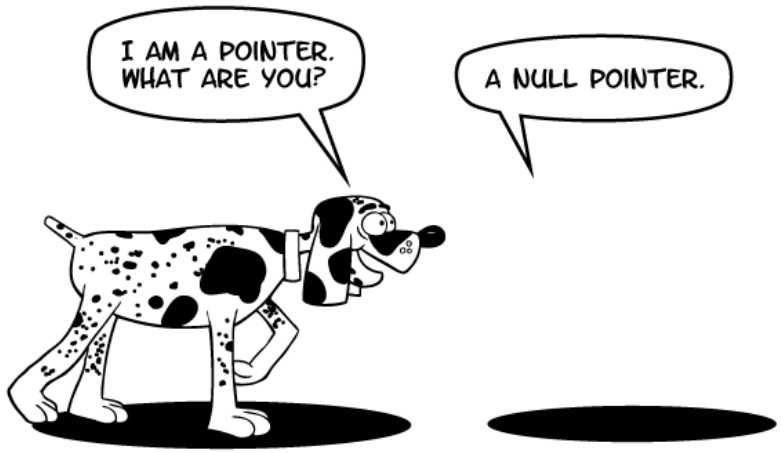
```
val notEmpty: (String) -> Boolean = { !it.notEmpty }
firstName.validateWith(notEmpty)
lastName.validateWith(notEmpty)
```

Null Safety

(Nullable types and Non-Null Types)

```
var a: String = "abc"
a = null // compilation error

var b: String? = "abc"
b = null // ok
```



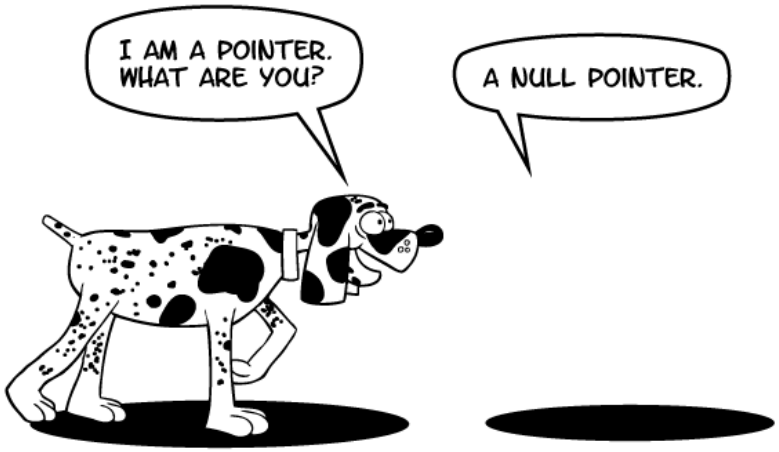
Null Safety

```
if(b != null){
        return b.lenght()
}

b?.lenght()

b!! //NPE

val l = b?.length() ?: 0
```



Extension Functions

```
fun Activity toast(   msg: CharSequence,
                      dtn: Int = Toast.LENGTH_SHORT){
        Toast.makeText(this, msg, dtn)
}


toast("Short Toast!!")
toast("Long Toast!!", Toast.LENGTH_LONG)
```

Default & Named Arguments

```
fun reformat(      str: String,
                   normalizeCase: Boolean = true,
                   upperCaseFirstLetter: Boolean = true,
                   divideByCamelHumps: Boolean = false,
                   wordSeparator: Char = ' ') {
...
}


reformat(str)


reformat(str, upperCaseFirstLetter = false )
```

Singleton

```
public object TheSingleton {

  public fun drink() {
    ...
  }

}

TheSingleton.drink()
```

Extension Function Expresions

```
db.beginTransaction();
try {
  db.delete("user", "name = ?", new String[] {"Michael"});
  db.setTransactionSuccessful();
} finally {
  db.endTransaction();
}
```

Extension Function Expresions

```
fun SQLiteDatabase.inTransaction(func: () -> Unit){
  beginTransaction()
  try {
    func()
    setTransactionSuccessful()
  } finally {
    endTransaction()
  }
}

db.inTransaction {
  db.delete("user", "name = ?", new String[] {"Michael"})
}
```

Extension Function Expresions

```
inline fun SQLiteDatabase.inTransaction(
                          func: SQLiteDatabase.() -> Unit){
  beginTransaction()
  try {
    func()
    setTransactionSuccessful()
  } finally {
    endTransaction()
  }
}

db.inTransaction {
  delete("user", "name = ?", new String[] {"Michael"})
}
```

Explosive Placeholders

```
fun foo(): String{
      if(something()){
            doSomething()
      } else {
            TODO("somethingElse()") // NotImplementedError
      }
}
```

Semantic Validation

```
fun join(sep: String, strings: List<String>): String {
        require(sep.length < 2) { "sep is too short" }
        ...
}
```

Code Block Measurement

```
var timeNS = measureTimeNanos{
        foo()
        bar()
}

println("foo() and bar() take $timeNS")
```

Deprecation

```
@Deprecated("use strings.joinToString(sep)", level = ...)
fun join(sep: String, strings: List<String>): String {
        ...
}
```

Levels:

ERROR

HIDDEN

Deprecation

```
@Deprecated("use strings.joinToString(sep)",
replaceWith = replaceWith("strings.joinToString(sep)")
fun join(sep: String, strings: List<String>): String {
        ...
}
```

Thank you for your attention