образование ПОЛИТЕХ одноклассники экосистема

Kotlin

Kotlin – это

1. Современный
2. Прагматичный
3. Статически типизированный
4. Объектно-ориентированный
5. Кросс-платформенный

язык программирования.

Разработка началась в 2010 году и ведется до сих пор.

Kotlin

- Полностью совместим с Java
- Вы можете использовать код на Java и Kotlin в одном проекте
- Компилируется в байт код так же, как и Java
- Является приоритетным языков для разработки под Android с 2019 года

Kotlin

Почему появился?

Перед JetBrains встала задача стать более продуктивными, сохранив полную совместимость со старыми разработками на языках, в первую очередь, Java, а затем JavaScript, C++.

Kotlin

Отличия от Java

1. Лаконичность
2. Null safety
3. Smart cast
4. Global functions
5. Extensions
6. Lambdas
7. Нет проверяемых exceptions

Kotlin

Синтаксис

Использует элементы языков Java, Scala, C#, Groovy, C++, JavaScript…

Kotlin

Null safety

```
var s: String = "i'm string"
s = null // ошибка

var optString: String? = "foo"
optString = null
```

Kotlin

Null safety

```kotlin
var s: String? = "some string"
val size = s?.length ?: 0
if (s != null) {
    val firstChat = s.first() // 's'
}
```

- Вызовы через ?
- Smart cast

Kotlin

Синтаксис

Функции:

- fun
- arguments
- throws
- expression fun
- fun in fun body
- extensions
- Inline fun

Kotlin

Переменные

```
var a = 4
a = 8
println("a = $a") // a = 2


val b = 15
b = 16
```

`var` - изменяемая переменная

`val` - неизменяемая

Kotlin

Переменные: тип

```
class File {
    var name: String = ""
}
```

Тип указывается через символ «:»

Kotlin

Функции

```
fun helloWorld() {
    println("Hello, world!")
}
```

Ключевое слово `fun`

Kotlin

Функции: аргументы и возвращаемый тип

```
fun add(array: IntArray, element: Int): IntArray {
    return array + element
}
```

Kotlin

Функции: аргументы по умолчанию

```
fun updateColor(color: Color = Color.RED) {
    this.color = color
}
```

Kotlin

Функции: вызов

```
fun drawView() {
    updateColor(color = Color.BLACK)
    updateColor(Color.WHITE)
}
```

Kotlin

Функции-расширения

```kotlin
fun Int.isZero(): Boolean = this == 0

fun MutableList<Int>.swap(index1: Int, index2: Int) {
    val tmp = this[index1]
    this[index1] = this[index2]
    this[index2] = tmp
}
```

Kotlin

Классы: open/final

```kotlin
open class MyClass {

}

class MySuperClass : MyClass() {

}

class MySecondSuperClass : MySuperClass() // ошибка! MySuperClass является final
```

Kotlin

Классы: `kotlin.Any` и `java.lang.Object`

В kotlin все классы унаследованы по умолчанию от `kotlin.Any`.

Это аналог `java.lang.Object`.

Kotlin

Классы: объявление

```kotlin
class Student(
    val firstName: String = "John",
    val lastName: String = "Doe"
) {

    constructor(map: Map<String, String>) : this(
        firstName = map.getOrDefault( key: "firstName",  defaultValue: "John"),
        lastName = map.getOrDefault( key: "lastName" ,  defaultValue: "Doe")
    )

    constructor(string: String) : this(map = string.toMap())

    init {
        println("created")
        // other logic
    }
}
```

- Имеет primary и вторичные конструкторы
- Объявление property в primary конструкторе
- Логика по инициализации внутри `init`

Kotlin

Классы: properties

```
class Cat {
    var name: String = "Семён"
        set(value) {
            println(value)
            if (value.isEmpty()) return
            field = value
        }

    val age: Int
        get() = 10

    val lifeMoments: List<Moment> by lazy {
        longAndHeavyCalculationsOfLifeMoments()
    }

    lateinit var owner: Human
}
```

- для проперти можно объявить свои setter и getter
- lazy помогает вычислять объемную информацию лениво, а не при создании объекта
- lateinit var позволяет присвоить объекту значение позже

Kotlin

Классы: модификаторы доступа

- Public используется по-умолчанию, код будет виден отовсюду;
- Private: будут видны только внутри класса/файла;
- Protected: видны наследникам класса и внутри файла;
- Internal: видно повсюду в рамках одного gradle-модуля;
- В отличие от java в kotlin нет package-only-visible модификатора.

Kotlin

Синглтон (object)

```kotlin
object Logger {
    fun log(tag: String, message: String, error: Throwable? = null) {
        println("$tag: $message ${error?.stackTrace}")
    }
}
```

Kotlin

Sealed class/interface

```
sealed class State {
    object Loading : State() {
        const val message = "Loading"
    }
    class Loaded(val items: List<String>): State()
    class Failure(val throwable: Throwable): State()
}
```

```
when (state) {
    State.Loading ->
        println("Loading..")
    is State.Loaded ->
        // в данном случае IDE подскажет, что state - это экземпляр State.Loaded
        println("Success ${state.items}")
    is State.Failure ->
        // и здесь
        println("Failure! ${state.throwable}")
}
```

Kotlin

Companion object

```
class Counter {
    companion object {
        // аналогично private static final String TAG = "Counter";
        private const val TAG = "Counter"
        // const поддерживает только примитивы
        private const val instance = Counter()
    }
}
```

- Здесь следует размещать константы;
- константа только примитив!

Kotlin

Интерфейсы

```
interface ThemedView {
    fun applyTheme(theme: Theme)
    fun getTheme(): Theme? {
        return null
    }
}
```

- Как в java
- Поддерживаются дефолтные реализации методов
- Может содержать companion object

Kotlin

Enum

```kotlin
enum class MimeType(val mimeType: String, val priority: Int) {
    JPEG( mimeType: "jpeg",  priority: 1),
    PNG( mimeType: "png",  priority: 1),
    MP4( mimeType: "mp4",  priority: 7);

    val contentType by lazy {
        "application/$mimeType"
    }

    companion object {
        fun mimeTypeBy(priority: Int): MimeType? {
            return values().firstOrNull { it.priority == priority }
        }
    }
}
```

- Все возможности из java
- Только чуть лаконичнее

Kotlin

Data класс

```
data class Image(
    val id: Long = 0,
    val width: Int = 0,
    val height: Int = 0,
    val size: Int = 0,
    val path: String? = null
)
```

```
val image = Image()
val anotherImage = Image()
assert(image == anotherImage) // всегда true

val copiedImage = image.copy(
    id = 10,
    width = 100,
    height = 300
)
assert(image != copiedImage)

// destructive declaration
val (id, width, height) = copiedImage
assert(id == 10L)
assert(width == 100)
assert(height == 300)
```

Kotlin

Data класс: ограничения

- Нельзя наследоваться
- Наличие проймами конструктора с перечисленными проперти
- Наличие минимум одного проперти

Kotlin

Как подружить с Java?

```kotlin
class CustomView @JvmOverloads constructor(
    context: Context,
    attrs: AttributeSet? = null,
    @AttrRes defStyleAttr: Int = 0
) : FrameLayout(context, attrs, defStyleAttr) {

    @JvmField
    val index: Int = 0

    companion object {
        @JvmStatic
        fun create(context: Context): CustomView = CustomView(context)
    }
}
```

- JvmOverloads для автоматического создания нужных методов
- JvmField для доступа без геттеров/сеттеров
- JvmStatic для вызова без Companion

Kotlin

Kotlin: что еще есть полезного?

- Pair:
- Глобальные функции: listOf, mapOf
- Форматирование строк
- Scope functions: apply, let, also, with
- Custom operators
- Sequence
- Delegated properties

Kotlin

Kotlin: что еще есть полезного?

```
val entry: Pair<String, String> = "key" to "value"
```

С помощью функции to удобно создавать объекты Pair

Kotlin

Kotlin: что еще есть полезного?

```
val list = listOf("1", "a", "ccc")
val map = mapOf(1 to "1", "2" to "2")
```

Kotlin

Kotlin: что еще есть полезного?

```kotlin
val list = listOf("1", "a", "ccc")
val map = mapOf(1 to "1", "2" to "2")

val intRange = 1..10
val otherIntRange = 1 until 10

intRange.forEach { i ->
    println(i)
}

for (i in 0..10 step 2) {
    println(i) // 0, 2, 4, 6, 8, 10
}
```

Kotlin

Kotlin: что еще есть полезного?

```
val i = 10
val string = "count = $i"
```

Kotlin

```kotlin
class CustomView @JvmOverloads constructor(
    context: Context,
    attrs: AttributeSet? = null,
    @AttrRes defStyleAttr: Int = 0
) : FrameLayout(context, attrs, defStyleAttr) {
    var title: TextView? = null
    val intent: Intent

    init {
        val title: TextView = TextView(context).also { it: TextView
            title = it
        }

        with(title) { this: TextView
            setBackgroundColor(Color.RED)
            text = "Some text"
            textSize = 12f
        }

        intent = Intent().apply { this: Intent
            putExtra( name: "color",  value: "red")
            putExtra( name: "url",  value: "https://ok.ru")
        }

        this.title?.let { it: TextView
            val char: Char = it.text[1]
            println(char)
        }
    }
}
```

Scoped functions

- also
- with
- apply
- let

Спасибо!

образование

ПОЛИТЕХ

одноклассники
экосистема