Основи Kotlin. Об’єктно-орієнтована парадигма.

Kotlin Class

Оголошення класу складається з імені класу, заголовка класу та тіла класу, оточеного фігурними дужками.

```
class Person { /*...*/ }
```

Як заголовок, так і тіло є необов'язковими; якщо клас не має тіла, фігурні дужки можна опустити.

Клас у Kotlin може мати первинний конструктор та один або декілька вторинних конструкторів. Первинний конструктор є частиною заголовка класу, він йде після імені класу та необов'язкових параметрів типу.

```
class Person constructor(firstName: String) { /*...*/ }
```

Якщо первинний конструктор не має анотацій або модифікаторів, ключове слово constructor можна опустити

Kotlin Class

```
class Customer

class Contact(val id: Int, var email: String)

fun main() {
    val customer = Customer()
    val contact = Contact(1, "mary@gmail.com")
    println(contact.id)
    contact.email = "jane@gmail.com"
}
```

Kotlin Class

Первинний конструктор не може містити коду. Код ініціалізації можна розмістити у блоках ініціалізаторів ***init{...}***. Під час ініціалізації екземпляру блоки ініціалізаторів виконуються у тому ж порядку, в якому вони з'являються у тілі класу.

Параметри цього конструктора можна використовувати у блоках ініціалізаторів. Вони також можуть бути використані в ініціалізаторах властивостей, оголошених у тілі класу.

Kotlin має синтаксис для оголошення властивостей та їх ініціалізації з первинного конструктора:

```
class Person(val firstName: String,
             val lastName: String, var age: Int)
```

Вторинні конструктори

В класі також можна оголошувати вторинні конструктори, вони повинні мати префікс ***constructor***:

```
class Person(val name: String) {
    val children: MutableList<Person> = mutableListOf()
    constructor(name: String, parent: Person) : this(name) {
        parent.children.add(this)
    }
}
```

Якщо клас має первинний конструктор, то вторинний конструктор повинен звертатися до первинного конструктора. Делегування іншому конструктору того самого класу здійснюється з допомогою ключового слова ***this.***

Вторинні конструктори

Якщо неабстрактний клас не має жодного конструктора (первинного чи вторинного), компілятор генерує первинний конструктор без параметрів. Видимість цього конструктора буде загальнодоступною. Якщо потрібен клас без відкритого конструктора, слід оголосити порожній первинний конструктор з видимістю не за замовчуванням.

Якщо всі параметри первинного конструктора мають значення за замовчуванням, компілятор JVM генерує додатковий конструктор без параметрів, що використовує значення за замовчуванням. Це полегшує використання Kotlin з такими бібліотеками, як Jackson або JPA, які створюють екземпляри класів за допомогою конструкторів без параметрів.

Розмежування доступу

Класи, об'єкти, інтерфейси, конструктори та функції, а також властивості та їхні сеттери можуть мати модифікатори видимості. Гетери завжди мають ту саму видимість, що і їхні властивості.

У Kotlin існує чотири модифікатори видимості: private, protected, internal та public. За замовчуванням буде встановлена загальнодоступна видимість.На цій сторінці ви дізнаєтеся, як модифікатори застосовуються до різних типів оголошених областей видимості.

Розмежування доступу

Для оголошень на рівні файлу (класи, об'єкти, інтерфейси та функції) діють такі правила:

- За замовчуванням використовується ***public***, що означає, що оголошення будуть видимі скрізь.
- Якщо ви позначите оголошення як ***private***, воно буде видимим лише всередині файлу, який містить це оголошення.
- Якщо ви позначите її як ***internal***, вона буде видима скрізь у тому ж модулі.
- Модифікатор ***protected*** недоступний для оголошень верхнього рівня.

Розмежування доступу

Для членів, оголошених всередині класу:

- ***private*** означає, що член класу є видимим лише всередині цього класу (включно з усіма його членами).
- ***protected*** означає, що член має таку саму видимість, як і член, позначений як private, але він також є видимим у підкласах.
- ***internal*** означає, що будь-який клієнт всередині цього модуля, який бачить клас, що оголошує, бачить його внутрішні члени.
- ***public*** означає, що будь-який клієнт, який бачить клас, що оголошує, бачить його загальнодоступні члени.

Зовнішній клас не бачить закритих членів своїх внутрішніх класів. Якщо перевизначається захищений або внутрішній член без явного вказання видимості, він матиме таку саму видимість, як і оригінальний.

Створення об’єктів

Щоб створити екземпляр класу, викликають конструктор, як звичайну функцію. В Kotlin немає ключового слова new.

```
val invoice = Invoice()
val customer = Customer("Joe Smith")
```

Клас може бути оголошено абстрактним разом з деякими або всіма його членами. Абстрактний член не має реалізації у своєму класі. Екземпляри абстрактного класу створювати не можна.

Inheritance

```
open class Dog {
// Kotlin classes are final by default.
// If you want to allow the class inheritance,
// mark the class with the open modifier.
    open fun sayHello() {
// Kotlin methods are also final by default. As with
// the classes, the open modifier allows overriding them.
        println("wow wow!")
    }
}
```

Inheritance

```
class Yorkshire : Dog() {
// A class inherits a superclass when you specify the :
// SuperclassName() after its name.
// The empty parentheses () indicate an invocation
// of the superclass default constructor.
    override fun sayHello() {
// Overriding methods or attributes requires the override
modifier.
        println("wif wif!")
    }
}
fun main() {
    val dog: Dog = Yorkshire()
    dog.sayHello()
}
```

Inheritance

Усі класи у Kotlin мають спільний суперклас ***Any***, який є суперкласом за замовчуванням для класів без оголошених супертипів.

Any має три методи: equals(), hashCode() та toString(), ці методи визначені для всіх класів Kotlin.

За замовчуванням, класи Kotlin є кінцевими - їх не можна успадковувати. Щоб зробити клас успадковуваним, позначте його ключовим словом ***open***.

Щоб оголосити явний супертип, помістіть тип після двокрапки в заголовку класу.

Якщо похідний клас має первинний конструктор, то базовий клас може (і повинен) бути ініціалізований у цьому конструкторі відповідно до його параметрів.

Якщо похідний клас не має первинного конструктора, то кожен вторинний конструктор повинен ініціалізувати базовий тип за допомогою ключового слова ***super.***

Інтерфейси

Інтерфейси в Kotlin можуть містити оголошення абстрактних методів, а також реалізації методів. Відмінність інтерфейсів від абстрактних класів полягає в тому, що вони не можуть зберігати стан. Вони можуть мати властивості, але вони повинні бути абстрактними або надавати реалізації аксесуарів. Інтерфейс визначається за допомогою ключового слова ***interface***:

```
interface MyInterface {
    fun bar()
    fun foo() {
        // optional body
    }
}
```

Імплементація інтерфейсу оголошується черз двокрапку, так як і спадкування класів, але клас може імплементувати декілька інтерфейсів.

Data Classes

Класи даних дозволяють легко створювати класи, які використовуються для зберігання значень. Такі класи автоматично забезпечуються методами копіювання, отримання рядкового представлення та використання екземплярів у колекціях. Можна перевизначити ці методи власною реалізацією в оголошенні класу.

```
data class User(val name: String, val id: Int) {
    override fun equals(other: Any?) =
        other is User && other.id == this.id
}
```

Data Classes

```
fun main() {
    val user = User("Alex", 1)
    println(user)

    val secondUser = User("Alex", 1)
    val thirdUser = User("Max", 2)
    println("user == secondUser: ${user == secondUser}")
    println("user == thirdUser: ${user == thirdUser}")

    // hashCode() function
    println(user.hashCode())
    println(secondUser.hashCode())
    println(thirdUser.hashCode())

    println(user.copy())
    println(user === user.copy())
    println(user.copy("Max"))
    println(user.copy(id = 3))
    println("name = ${user.component1()}")
    println("id = ${user.component2()}")
}
```

Null Safety

```
//Declares a non-null String variable.
    var neverNull: String = "This can't be null"
    // neverNull = null
// When trying to assign null to non-nullable variable,
// a compilation error is produced.

// Declares a nullable String variable.
    var nullable: String? = "You can keep a null here"
//Sets the null value to the nullable variable. This is OK.
    nullable = null

 nullable = "You can keep a null"

//Convert nullable to a non-null String variable.
 var n: Int = nullable!!.lenth
    // neverNull = null
```

Null Safety

```
// When inferring types, the compiler assumes non-null
// for variables that are initialized with a value.
    var inferredNonNull = "The compiler assumes non-null"
// When trying to assign null to a variable with inferred t
// ype, a compilation error is produced.
    //inferredNonNull = null
//Declares a function with a non-null string parameter.
    fun strLength(notNull: String): Int {
        return notNull.length
    }
//Calls the function with a String (non-nullable) argument. This is OK.
//When calling the function with a String? (nullable) argument, a
compilation error is produced.
    strLength(neverNull)

//Calls the function with a String (non-nullable) argument. This is OK.
//When calling the function with a String? (nullable) argument, a
compilation error is produced.
    //strLength(nullable)
```

Kotlin Koans

Kotlin Koans is a series of exercises to get you familiar with the Kotlin syntax and some idioms. Each exercise is created as a failing unit test, and your job is to make it pass. Here you can play with Koans online, but the same version of exercises is also available via JetBrains educational plugin right inside IntelliJ IDEA or Android Studio.

According to our surveys, Kotlin Koans is one of the most popular and most effective ways to get into Kotlin for people who already know Java. In just a few hours, you'll feel able to write idiomatic Kotlin code.

https://play.kotlinlang.org/koans/overview