Практична робота 8. Розробка мобільного додатку в Android Studio з використанням мови Kotlin

Теоретичні відомості

Kotlin (Котлін) – статично типізована мова програмування, що працює поверх JVM і розробляється компанією JetBrains. Також компілюється у JavaScript. Мову названо на честь острова Котлін у Фінській затоці, на якому розміщено частину Кронштадта.

Автори ставили собі за мету створити лаконічнішу та типо-безпечнішу мову, ніж Java, і простішу, ніж Scala. Наслідками упрощення, порівняно зі Scala стали також швидка компіляція та краща підтримка IDE.

Мова розробляється з 2010 року, публічно представлена у липні 2011. Початковий код було відкрито у лютому 2012 року. У лютому було випущено milestone 1, який містив плагін для IDEA. У червні - milestone 2 з підтримкою Android. У грудні 2012 року вийшов milestone 4 та забезпечивши підтримку Java 7. На листопад 2015 року основні можливості мови стабілізовані, готується реліз версії 1.0. У грудні 2015 року з'явився реліз-кандидат версії 1.0, а 15 лютого 2016 року відбувся реліз версії 1.0.

З 17 травня 2017 року входить до списку офіційно підтримуваних мов для розробки застосунків для платформи Android.

З 7 травня 2019 року є рекомендованою мовою програмування для розробки Android застосунків.

Kotlin намагається скоротити кількість коду під час створення нових класів. У Kotlin вирішили реалізувати інший підхід до класів. Наприклад, стандартні оголошення отримують модифікатори final і public. Вкладені класи за замовчанням є внутрішніми та інші особливості.

У Kotlin немає жорсткої прив'язки класів та файлів. Ви можете створити кілька класів в одному файлі та вибрати будь-яке ім'я для цього файлу. При цьому також не важливо розміщення файлів з класами на диску, можна навіть не

звертати увагу на імена пакетів і дотримуватися точної ієрархії вкладеності. Але краще дотримуватися правил Java.

Для класу використається ключове слово class. З ним можна використовувати також ключові слова: data, open, internal, abstract, public.

Клас являє собою шаблон, що визначає властивості та функції, пов'язані з об'єктами цього типу. Android вже містить безліч готових класів, але все передбачити неможливо і вам доведеться створювати власні класи. Наприклад, ми можемо створити клас для кота.

Якщо програма призначена для зберігання інформації про котів, ви можете визначити клас Cat для створення власних об'єктів Cat. У таких об'єктах швидше за все вам знадобиться зберігати ім'я кота (name), його вага (weight) та порода (breed). Ви можете вигадати свої варіанти для ваших конкретних завдань.

У більшості випадків клас містить властивості та функції. Хоча Kotlin дозволяє створити порожній безглуздий клас. Назва класу має починатися з літери (прийнято з великої літери), після літери можна використовувати цифри та символ підкреслення.

Створимо найпростіший клас для розуміння.

```
class Cat
```

У Kotlin ключове слово public при створенні класу не потрібне, за замовчуванням клас має даний модифікатор і не потрібно вказувати його.

Викликаємо клас у коді.

```
val cat = Cat()
```

Один рядок коду для створення класу, один рядок для виклику.

У Java для класів використовуються поля, Kotlin оперує властивостями. Властивість – те, що об'єкт знає себе. Кіт, що себе поважає, знає, як його звуть, скільки він важить і якої він породи. Клас Car (машина) матиме свій набір властивостей – марка, колір, швидкість тощо.

Те, що об'єкт може зробити, - це його функції (в методах Java). Вони визначають поведінку об'єкта та можуть використовувати властивості об'єкта. Клас Cat може містити функцію meow().

```
class Cat {
    fun meow() = "Meow!"
}
```

Дізнаємось, що сказав кіт.

```
val cat = Cat()
val spokenWord = cat.meow()
println(spokenWord)
```

Створимо клас із параметрами.

```
class Cat(val name: String, var weight: Int, val breed: String) {}
```

Ми зараз не тільки створили клас, але й задали йому три властивості, використовуючи ключові слова val та var (це важливо).

Функції пишуться усередині фігурних дужок. Створимо функцію sleep(). Якщо це ще маленьке кошеня, то уві сні він сопе, а солідний кіт уже хропе.

```
class Cat(val name: String, var weight: Int, val breed: String) {
    fun sleep() {
        println(if (weight < 3) "sniffs!" else "snoring!")
    }
}
```

Створюємо об'єкт Барсик, використовуючи всі властивості класу.

```
var barsik = Cat("Barsyk", 4, "Manchkin")
```

Доступ до будь-якої властивості досягається через оператор . (крапка):

```
println(barsik.name)
println(barsik.weight)
println(barsik.breed)
```

Якщо властивість задано ключовим словом var, ми можемо як прочитати властивість, а й встановити нове значення. У нашому випадку такою властивістю є weight. Давайте нагодуємо кота, щоб збільшити його вагу.

```
barsik.weight = 6
println(barsik.weight)
```

Якщо ви захочете змінити властивості із ключовим словом val, то у вас нічого не вийде. Компілятор повідомить про помилку (error: val cannot be reassigned). Спробуйте це зробити самостійно, щоб побачити на власні очі.

Функція викликається також через оператор "крапка", але не забувайте про круглі дужки.

```
barsik.sleep()
```

Ми можемо створити масив об'єктів нашого класу.

```
val cats = arrayOf(Cat("Barsyk", 4, "Manchkin"), Cat("Mursyk", 5, "Bengal"))
cats[1].weight = 6 // нагодували другого кота
println(cats[1].name) // а нагадайте, як його звати?
println(cats[1].weight) // і скільки він важить зараз?
cats[0].sleep() // як спить перший кіт?
```

Створення об'єкта дуже схоже виклик функції - використовуються круглі дужки, тільки замість назви функції пишемо ім'я класу. Насправді це конструктор. Докладніше про конструкторів поговоримо пізніше. Поки що потрібно запам'ятати – конструктор містить код, необхідний ініціалізації об'єкта. Часто використовують конструктори визначення властивостей об'єкта і присвоєння їм значень. Об'єкт іноді називають екземпляром класу, яке властивості – змінними екземплярів.

Створення класу з готовими властивостями як параметрів дуже зручно і часто використовується прийом. Kotlin допомагає писати код у зручному компактному вигляді. Але іноді потрібно з властивістю щось зробити – перевірити на умову, присвоїти значення за умовчанням тощо. У цьому випадку ви можете використовувати стиль, прийнятий Java.

```
class WildCat(name: String, weight: Int, breed: String){
    val name = name
    var weight = weight
    val breed = breed
}
```

```
val manul = WildCat("Manulik", 4, "Uknown")
println(manul.name)
```

У конструкторі ми використовуємо ключові слова val і var - у разі name, weight і breed не є властивостями класу, а є звичайними параметрами конструктора. А властивості задаються вже усередині фігурних дужок. Не дуже красиво, що імена властивостей збігаються з іменами параметрів, код стає нечитаним. Деякі програмісти вважають за краще уникати такої схожості та використовують наступний підхід.

```
class WildCat(name_param: String, weight_param: Int, breed_param: String){
   val name = name_param
   var weight = weight_param
   val breed = breed_param
}
val manul = WildCat("Manulik", 4, "Uknown")
println(manul.breed)
```

Так легше розрізняти де властивість і де параметр.

Інший популярний спосіб – знак підкреслення.

```
class WildCat(_name: String, _weight: Int, _breed: String){
   val name = _name
   var weight = _weight
   val breed = _breed
}
```

Визначення властивостей у тілі класу забезпечує більшу гнучкість, ніж просте додавання в конструктор, оскільки у разі не потрібно ініціалізувати кожну властивість значенням параметра конструктора.

Припустимо, ви хочете визначити для нової властивості значення за замовчуванням, не включаючи в конструктор. Додамо до класу властивість activities та ініціалізуємо його масивом, який за умовчанням містить значення «Play».

```
class Cat(val name: String, var weight: Int, val breed: String) {
```

```
    var activities = arrayOf("Play")
    fun sleep() {
        println(if (weight < 3) "sniffs!" else "snoring!")
    }
}
```

Інший варіант - потрібно змінити значення параметра конструктора перед тим, як надавати його властивості. Наприклад, ми хочемо, щоб властивість breed виводила рядок у верхньому регістрі. У параметрі конструктора ми задаємо звичайний рядок, а потім створюємо нову версію рядка через функцію toUpperCase().

```
class Cat(val name: String, var weight: Int, breed_param: String) {
    var activities = arrayOf("Play")
    val breed = breed_param.toUpperCase()
    ...
}
```

Зверніть увагу, що ми прибрали з конструктора властивість breed (немає ключового слова val) та замінили параметром breed_param. У тілі класу створили властивість breed, що приймає параметр конструктора та перетворює його.

Зазначений спосіб ініціалізації властивостей добре працює, якщо ви хочете присвоїти просте значення або вираз. А якщо знадобиться зробити щось складніше? Чи потрібно виконати додатковий код, який складно додати до конструктора?

Тут нам допоможе блоки ініціалізації, які виконуються при ініціалізації об'єкта відразу після виклику конструктора та забезпечуються префіксом init. Блоків init може бути декілька. Вони виконуються у порядку, у якому визначаються у тілі класу, чергуючись з ініціалізаторами властивостей.

```
class Cat(val name: String, var weight: Int, breed_param: String) {
    // Спочатку буде створено властивості, задані в конструкторі
    // Потім виконається перший блок ініціалізації
```

```
    init {
        println("Cat $name was created.")
    }
    // Потім буде створено властивості після завершення першого блоку ініціалізації.
    var activities = arrayOf("Play")
    val breed = breed_param.toUpperCase()
    // Тепер настає черга виконання другого блоку ініціалізації
    init {
        println("Breed: $breed.")
    }
    ...
}
```

Важливий момент - властивості, що задаються в тілі класу, повинні ініціалізуватися перед їх використанням у коді. Ви не зможете пропустити цей крок, оскільки компілятор відмовиться компілювати ваш код. Спробуйте вигадати самостійно нову властивість для кота, але не ініціалізувати його.

Фактично при створенні властивості та її ініціалізації, ви призначаєте властивості значення за замовчуванням. Наприклад, для рядкових властивостей можна використовувати порожній рядок, чисел 0 і т.п.

Можна створити клас без конструктора – після імені класу не використовуємо круглі дужки. Тоді потрібно додати лише фігурні дужки для блоку.

```
class Cat{
    fun sleep(){
        println("Cat is sleeping»)
    }
}
```

Коли ви визначаєте клас без архітектора, компілятор генерує його за вас. Він додає порожній конструктор (конструктор без параметрів) у відкомпільований код, який нічого не робить. Ваш код буде рівносильний коду:

```
class Cat(){
    ...
}
```

І створювати об'єкт вам все одно доведеться з використанням круглих дужок.

```
val cat = Cat()
```

Якщо ви успадковуватиметеся від подібного класу, не створюючи своїх конструкторів, слід явно викликати конструктор суперкласу, незважаючи на те, що він не має параметрів.

```
class Milk: Food()
```

Можна обійтись без фігурних дужок.

```
class Cat(val name: String, var weight: Int, val breed: String)
var barsik = Cat("Barsyk", 4, "Manchkin")
```

Подібний конструктор вважається основним або первинним (primary) у Kotlin. Він оголошується поза тілом класу. Додаткові вторинні конструктори оголошуються вже у тілі класу.

Вимоги до звіту

1. Назва та мета роботи.
2. Лістинг створеної програми в Android Studio за допомогою командних елементів керування, елементів вибору, елементів введення та елементів відображення.
3. Результати роботи програми в емуляторі телефону.

Завдання для виконання

№	Клас
1	Співробітник, 3 поля
2	Студент, 2 поля
3	Товар, 3 поля
4	Пес, 2 поля
5	Геометрична фігура, 2 поля
6	Програмне забезпечення, 3 поля
7	Апаратне забезпечення, 3 поля
8	Місто, 2 поля
9	Країна, 2 поля
10	Книга, 3 поля
11	Співробітник, 3 поля – 3 класи спадкоємця
12	Студент, 2 поля – 2 класи спадкоємця
13	Товар, 3 поля – 4 класи спадкоємця
14	Пес, 2 поля – 3 класи спадкоємця
15	Геометрична фігура, 2 поля – 3 класи спадкоємця
16	Програмне забезпечення, 3 поля – 2 класи спадкоємця
17	Апаратне забезпечення, 3 поля – 3 класи спадкоємця
18	Місто, 2 поля – 2 класи спадкоємця
19	Країна, 2 поля – 2 класи спадкоємця
20	Книга, 3 поля – 4 класи спадкоємця