Samsung R&D Institute Poland

Kotlin Tech Guide

Poszerzaj i utrwalaj znajomość zagadnień technicznych, korzystając z praktycznych wskazówek naszych ekspertów.

Samsung Tech Guides

Drogi kandydacie!
Droga kandydatko!

Zapraszamy Cię do zapoznania się z Kotlin Technical Guide - technicznym przewodnikiem poświęconym jednemu z najbardziej popularnych języków programowania!

Znajdziesz w nim tematy, które warto powtórzyć przed spotkaniem technicznym w rekrutacji do Samsung R&D!

Dodatkowo nasi inżynierowie przygotowali przykładowe zadania testowe, które pomogą Ci sprawdzić swoją wiedzę.

Kotlin Technical Guide - Dla kogo?

dla kandydatów aplikujących na stanowiska, w których wymagana jest znajomość Kotlina,

dla kandydatów zaproszonych już na spotkanie techniczne do Samsung R&D,

dla początkujących inżynierów chcących poszerzyć/utrwalić swoją wiedzę z Kotlina,

dla osób zainteresowanych w przyszłości rekrutacją do Samsung R&D.

Powodzenia!

Kotlin - podstawy

Praktyczne wykorzystanie języka wymaga dobrego opanowania jego podstawowych założeń. Konieczne jest uprzednie zapoznanie się z następującymi zagadnieniami:

- val a var, różnice między nimi,
- typy danych dostępne standardowo,
- klasa String i operacje na niej,
- typy Enum,
- instrukcje sterujące i pętle,
- scope functions.

Hierarchia klas

W języku obiektowym, jakim jest Kotlin, kluczową rolę odgrywają klasy. Niezbędne w pracy jest, aby wiedzieć, jak działa w języku Kotlin ich hierarchia, w szczególności:

- tworzenie typów pochodnych,
- różnice między interfejsem a klasą abstrakcyjną,
- jak działa rzutowanie typów,
- na czym polega polimorfizm,
- czym są anonimowe klasy wewnętrzne,
- czym są typy szablonowe (ang. Generics).

Struktury danych

Język Kotlin udostępnia implementacje powszechnie stosowanych struktur danych. Od kandydatów oczekuje się, by znać przeznaczenie i sposób ich użycia, w szczególności:

- rozróżnienie użycia wersji mutowalnej i niemutowalnej,
- co jest interfejsem, a co konkretną implementacją,
- w jakich sytuacjach stosuje się poszczególne struktury, jakie są ich mocne i słabe strony (np. złożoność czasowa przy konkretnych operacjach).

Przetwarzanie współbieżne

Niejednokrotnie do czynienia mamy z sytuacjami, w których tworzony program nie może lub nie powinien wykonywać operacji w sposób sekwencyjny. Konieczne jest wykonywanie wielu zadań równocześnie. Przykładem niech będą sytuacje, w których aplikacja musi reagować na zdarzenia interfejsu użytkownika. Kandydaci powinni wykazać się znajomością mechanizmów umożliwiających wdrożenie takich rozwiązań:

- pojęcie wątków i związaną z nim klasę Thread oraz podstawowe operacje na niej (takie jak sleep czy wait),
- rozwiązania zawarte w korutynach i ich użycie,
- (opcjonalnie) klasyczne mechanizmy współbieżności, używane w językach JVM (Future,Callback, synchronizacja bloków).

System kontroli wersji (Git™)

Aby efektywnie zarządzać bazą kodu i synchronizować pracę wielu osób wykorzystuje się istniejące systemy kontroli wersji. Wykorzystywanym narzędziem jest przede wszystkim Git, tak więc kluczowe jest, aby znać:

podstawowe założenia (takie jak branch, czym różni się merge od rebase),

komendy (np. checkout, add, commit) w stopniu umożliwiającym sprawną pracę,

sposób pisania treściwej, lecz zarazem znaczącej treści commitu,

narzędzie bazujące na systemie kontroli wersji.

podstawowy „flow" pracy w projekcie z głównym branchem main (lub master), do którego składane są Pull Requesty (PR) bądź Code Review (CR),

Operacje na plikach

Większość programów nie obędzie się bez wczytywania bądź zapisywania plików lub strumieni danych z zewnątrz programu. Przydatne do tego będzie:

- zaznajomienie się z wbudowanymi bibliotekami umożliwiające wygodny odczyt i zapis,
- znajomość formatu JSON oraz sposobu przełożenia (serializacji) go na obiekty Kotlin,
- użycie folderu resources w strukturze projektu.

Środowisko programi-styczne

W codziennej pracy wykorzystywane są dziś powszechnie środowiska programistyczne, ułatwiające w znacznym stopniu pisanie i budowanie kodu. W toku pracy niezbędne będzie, aby znać:

- środowiska IntelliJ IDEA oraz Android Studio [Android jest znakiem towarowym Google LLC],
- integrację z narzędziami budowania takimi jak Maven i Gradle,
- sposób uruchomienia swojego programu z konsoli (zarówno Linux jak i Windows).

Typowo dzień pracy programisty w znacznej mierze wypełnia również komunikacja ze współpracownikami. Z tego względu warto poznać narzędzia wspomagające zarządzanie zadaniami, bądź wymianę informacji, zarówno z członkami zespołu, jak i z przełożonym:

- Jira,
- Slack.

Obsługa błędów

Integralną częścią logiki programu jest obsługa sytuacji niepożądanych, które mogą pojawić się w trakcie jego wykonywania. Przydatnym jest wiedzieć:

- kiedy obsłużyć najczęstsze typy błędów (IndexOutOfBoundsException, IOException...),
- jaka jest hierarchia błędów i jak obsłużyć kilka scenariuszy jednocześnie,
- jak pokierować dalszą egzekucją po wystąpieniu błędu, (opcjonalnie) w jaki sposób obsłużyć błędy w korutynach.

Analiza statystyczna

Aby zredukować liczbę defektów w kodzie, stosuje się techniki automatycznej analizy. Pozwala ona ujawnić błędy, których istnienia developerzy często nie są świadomi. Od kandydatów oczekuję się, że będą wiedzieć:

Dobre praktyki

Miarą jakości kodu jest nie tylko jego działanie, ale także łatwość w jego utrzymaniu, szybkiego zrozumienia przez innych oraz modyfikacji. W tym celu konieczna jest czytelność i możliwie proste wprowadzanie zmian. W związku z tym kandydujący powinni posiadać rozeznanie w kwestii inżynierii kodu, w szczególności:

- ogólne praktyki dotyczące czystości kodu, takie jak dobór długości metod,
- nazewnictwo zmiennych,
- rozumieć, dlaczego komentarze w kodzie należy stosować oszczędnie i ostrożnie,
- wzorce projektowe np. Builder lub Singleton.

Testy jednostkowe

Testowanie jest często traktowane jako rzecz drugorzędna. Tymczasem testy pełnią ważną funkcję w cyklu życia projektu, z tego względu osoba kandydująca powinna znać:

- znaczenie unit testów dla utrzymania projektu oraz pojęcie regresji,
- podstawy użycia biblioteki JUnit, takie jak metody assert i adnotacje, np. @Test, @BeforeEach,
- ogólne założenia metodyki TDD.

Elementy programowania funkcyjnego w Kotlinie

Kotlin jako język, obok podejścia typowo obiektowego, umożliwia wykorzystanie elementów programowania funkcyjnego. Rozwiązania te stosowane są na szeroką skalę, dlatego też kandydujący powinni znać następujące zagadnienia:

- gdzie wykorzystuje się wyrażenia lambda,
- czym charakteryzują się funkcje wyższego rzędu (ang. High-order functions),
- podstawowe operacje na kolekcjach, takie jak map, flatmap, filter,
- mechanizm działania sekwencji w porównaniu do zwykłych strumieni danych.

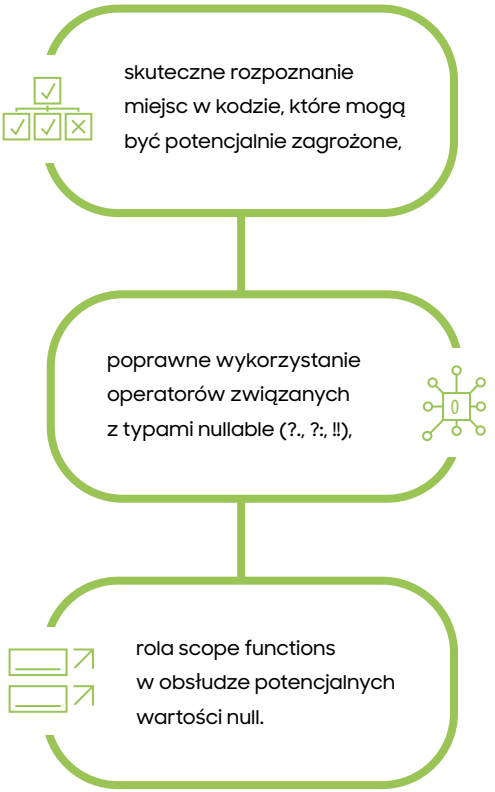
Obiektowość

Obiektowość to nie tylko metoda strukturyzacji kodu, lecz też sposób formowania zależności pomiędzy danymi, przepływu informacji oraz dynamicznego wstrzykiwania implementacji. Pomocne do tego będzie zaznajomienie się z:

- filozofią obiektowości oraz sposobem rozwiązywania problemów i zależności pomiędzy obiektami,
- akronimem SOLID w celu wyjaśnienia i zastosowania poszczególnych jego składowych,
- różnicami pomiędzy klasą statyczną a klasą POJO,
- pojęciem - „klasa z danymi" (data class lub DTO),
- konsekwencjami wstrzykiwania obiektów przez dziedziczenie bądź kompozycje,
- rolą interfejsów.

Obsługa wartości null

Konieczne jest, aby rozumieć pojęcie „null", sposób jego wykorzystania, związane z nim problemy. Potrzebna będzie znajomość mechanizmów języka Kotlin w tym zakresie:



1.
Dane są tablice Stringów:

```
val authors = arrayOf("Parker","Parker","King","Homer",
"Hemingway","Goethe","Homer")
val books = arrayOf("Devices and Desires","Evil for Evil",
"The Dark Tower","Odyssey","For whom the Bell Tolls",
"Faust","Iliad")
```

Indeks nazwiska autora w tablicy authors, odpowiada indeksowi książki, którą napisał w tablicy books. Wynik działania programu powinien prezentować nazwisko danego autora tylko raz oraz tylko jeden z pojawiających się jego utworów.

2.
Dana jest mapa państw na ich stolice:

```
val countryToCapital = mapOf(
    "Poland" to "Warsaw",
    "France" to "Paris",
    "Germany" to "Berlin",
    "Spain" to "Madrid",
    "Czech Republic" to "Prague",
    "Italy" to "Rome"
```

- Należy napisać kod, który zwróci listę tych krajów, których stolica zaczyna się na literę „P".
- Na ile innych sposobów można wykonać to zadanie? Dlaczego są gorsze od zaproponowanego?

3.
Dany jest następujący fragment JSON:

```
company:"Samsung",
employee_count:3,
"employees":[
        {"name":"Helen","email":"helen@mail.com"},
        {"name":"Bob","email":"bob@mail.com"},
        {"name":"Hailey","email":"hailey@mail.com"}
    ]
```

- Należy stworzyć klasy w Kotlinie, do których można by zdeserializować ten fragment JSON.
- Jakich narzędzi i technik można użyć do samego procesu serializacji/deserializacji?
- Następnie, do obiektu stworzonego na bazie powyższej klasy, należy dodać nowego pracownika na pozycję 3, zachowując pozostałych pracowników.
- Analogicznie, usunąć obiekt reprezentujący pracownika Bob.
- Czy da się uogólnić proces dodawania i usuwania do postaci funkcji, która działa na przekazanej w argumencie pozycji w liście?

4.
Dana jest klasa:

```
class Untested(private var x: Int) {
    fun adjustX(newX: Int): Untested {
        x = newX
        return this
    }

    override fun equals(obj: Any?): Boolean {
        return (this === obj
                || obj is Untested && obj.x == x)
    }
}
```

Zadanie polega na napisaniu testu jednostkowego, który sprawdzi, czy wartość x dla instancji Untested tworzonej z argumentem 5, po wywołaniu metody adjustX() z argumentem 8 będzie rzeczywiście różna od początkowej.

Dodatkowo można sprawdzić, czy adjustX() na pewno zwraca ten sam obiekt co ten, na którym została wywołana.

5.
Dane są imiona klientów:

```
val customers = arrayOf("Harmony","Helga",
"Hermione", "Helen","Hailey")
```

Zadanie polega na napisaniu kodu, który umożliwia przechowywanie struktury:

- zachowującej kolejność alfabetyczną imion,
- umożliwiającej łatwe (mało kosztowne algorytmicznie), usuwanie elementów z jakiejkolwiek pozycji.

6. Dany jest ciąg znaków:

```
val sentence = "We will steer SRPOL together
to our future success."
```

Jak można zaimplementować rozwiązanie, które:

- zwraca sumaryczną długość 2 ostatnich słów?
- zwraca sumaryczną liczbę wystąpień znaków „r" oraz „R"?

7.

Dana jest metoda obtainStringOrNull(), która z równym prawdopodobieństwem zwraca String lub null.

Nie używając instrukcji „if" należy zamienić zwrócony String na same wielkie litery lub jeżeli zwrócony String okaże się nullem, zastąpić go przez „nothing".

8.

Rozwiązaniem tego zadania jest kod, który wyświetli asynchronicznie kolejne liczby od 1 do 10.

- Następnie wykona działanie sumowania w każdym wątku i zwróci wynik.
- Na koniec zsumuje wszystkie wyniki.

9.
Dana jest klasa StringProvider, a wewnątrz niej metoda z następującą treścią:

```
fun getResourceString(manager:
ExternalResourcesManager): String? {
        val provider = manager.accessStringProvider()
        return provider.fetchAvailableString()
    }
```

- Jak obsłużyć wyjątki, żeby program nie został zamknięty?
- Jak obsłużyć wyjątki w taki sposób, żeby zwrócić jak najbardziej precyzyjną informację o napotkanym problemie?

10.
Mając poniższą sygnaturę metody:

```
fun <T, R> Collection<T>.fold(
    initial: R,
    operation: (acc: R, nextElement: T) -> R
): R
```

- Wytłumacz, czym są jej poszczególne argumenty.
- Do rozwiązania jakiego problemu można użyć funkcję fold? Zaproponuj krótkie rozwiązanie wraz z kodem.

11.

Należy napisać kod, który zwróci odpowiedź na pytanie:
czy zawartość stworzonych obiektów typu Bug jest równa?

```
data class Device(val location: String)
data class Bug(val type: String, val deviceAddress: Device)

fun main() {

    val deviceLocal = Device( location: "Warsaw")
    val deviceRemote = Device( location: "Berlin")

    val bugA = Bug( type: "OutOfMemory", deviceLocal)
    val bugB = Bug( type: "NullPointerException", deviceRemote)
    val bugC = Bug( type: "OutOfMemory", deviceLocal)
}
```

- Jak powinno się zmienić tę funkcję, żeby nie uwzględniać wartości obiektu device w porównaniu?
- Jak należy zmodyfikować funkcję zwracającą równość, jeśli sygnatura klasy będzie jak poniżej:

```
data class Bug(val type: String, val deviceAddress:
List<Device>)
```

Samsung R&D Institute Poland

Samsung Tech Guides