Advanced Kotlin

Android Advanced

18.01.2025

Higher-order functions

A higher-order function is a function that takes functions as parameters, or returns a function.

```
Returns a list containing only elements matching the given predicate.
Samples: samples.collections.Collections.Filtering.filter
        // Unresolved
public inline fun <T> Iterable<T>.filter(predicate: (T) -> Boolean): List<T> {
    return filterTo(ArrayList<T>(), predicate)
}
```

Inline Functions

Each higher order function we created leads to a new object creation and memory allocation.

By “inlining” the function code, Kotlin improves performance by reducing the overhead associated with function calls.

https://habr.com/ru/articles/775120/

Extension functions

Kotlin provides the ability to extend a class or an interface with new functionality without having to inherit from the class or use design patterns such as Decorator. This is done via special declarations called extensions.

For example, you can write new functions for a class or an interface from a third-party library that you can't modify. Such functions can be called in the usual way, as if they were methods of the original class. This mechanism is called an extension function.

```
fun MutableList<Int>.swap(index1: Int, index2: Int) {
    val tmp = this[index1] // 'this' corresponds to the list
    this[index1] = this[index2]
    this[index2] = tmp
}
```

Property delegation

Property delegation is a powerful technique, that allows you to write code that takes over control of other properties, and helps this logic to be easily shared amongst different classes. This allows for robust, reusable logic that looks and feels like regular property access

lazy

Lazy initialization is a design pattern that we often come across in the software world. With lazy initialization, we can create objects only the first time that we access them, otherwise we don’t have to initialize them. It ensures that objects that are expensive to create are initialized only where they are to be used, not on the app startup.

lateinit

We may not want to initialize our values at declaration time, but instead want to initialize and use them at any later time in our application. However, before using our value, we must not forget that our value must be initialized before it can be used. Let’s make an example for better understanding!

Sealed Classes and Enum Classes

Sealed:

They are a type of class in Kotlin that are used to represent a closed(fixed) set of cases. They allow you to restrict the possible types of a value to a predefined set of cases, making it easier to handle cases that are known at compile time.

Enum:

They are used to represent a fixed set of "values", where each value is called an enum constant. Enum classes are used to define a set of possible values that are known at compile time, and can be used to simplify code and improve readability.