Kotlin Advanced Code Sample – Domain-Specific Event Flow Processor DSL

// High-Ranked Keywords: Kotlin DSL, Coroutines, Inline Reified Generics, Sealed Classes, Type-safe Builders, Multiplatform, Dependency Injection, Reflection

```
@file:OptIn(ExperimentalStdlibApi::class)

package com.techgiant.hr.impress.dsl

import kotlinx.coroutines.*
import kotlin.reflect.KClass
import kotlin.time.Duration
import kotlin.time.Duration.Companion.seconds

// --- Rare & Advanced Custom Coroutine Dispatcher ---
object SupervisedScopedDispatcher : CoroutineDispatcher() {
    private val scope = CoroutineScope(SupervisorJob() + Dispatchers.IO)

    override fun dispatch(context: CoroutineContext, block: Runnable) {
        scope.launch { block.run() }
    }
```

```
}

// --- Sealed Class with DSL & Reflection Use ---
sealed class Event(val name: String) {
    data class StartEvent(val metadata: Map<String, Any>) : Event("start")
    data class StopEvent(val reason: String) : Event("stop")
    data class DataEvent(val payload: ByteArray) : Event("data")
}

typealias EventHandler<E> = suspend (E) -> Unit

// --- Type-safe DSL Context Receiver ---
@DslMarker
annotation class EventDsl

@EventDsl
class EventFlowBuilder {
    private val handlers = mutableMapOf<KClass<*>, suspend (Event) -> Unit>()

    inline fun <reified T : Event> onEvent(noinline handler: suspend (T) -> Unit) {
        handlers[T::class] = { event ->
            if (event is T) handler(event)
        }
    }
```

```
    fun build(): EventProcessor = EventProcessor(handlers)
}

// --- Event Processor Using Reified + Coroutine Dispatcher ---
class EventProcessor(
    private val handlerMap: Map<KClass<*>, suspend (Event) -> Unit>
) {
    private val dispatcher = SupervisedScopedDispatcher

    suspend fun process(event: Event) = withContext(dispatcher) {
        val handler = handlerMap[event::class]
        handler?.invoke(event)
            ?: println("No handler found for ${event.name}")
    }
}

// --- Rare Kotlin Feature: Context Receivers (Preview Feature) ---
context(EventProcessor)
suspend fun runFlow(events: List<Event>) {
    for (e in events) process(e)
}

// --- Type-safe DSL Entry Point ---
fun eventFlow(block: EventFlowBuilder.() -> Unit): EventProcessor {
    return EventFlowBuilder().apply(block).build()
```

```
}

// --- Advanced Reified Generic Dependency Injection Container ---
class Injector {
    private val services = mutableMapOf<KClass<*>, Any>()

    inline fun <reified T : Any> bind(instance: T) {
        services[T::class] = instance
    }

    inline fun <reified T : Any> resolve(): T {
        return services[T::class] as? T
            ?: error("Service not found for ${T::class}")
    }
}

// --- Kotlin Multiplatform-Ready Interface ---
expect class PlatformLogger() {
    fun log(message: String)
}

// --- JVM-specific Actual Implementation ---
actual class PlatformLogger actual constructor() {
    actual fun log(message: String) = println("[LOG] $message")
}
```

```
// --- Advanced Main DSL Usage Example ---
suspend fun main() {
    val injector = Injector().apply {
        bind(PlatformLogger())
    }

    val logger = injector.resolve<PlatformLogger>()

    val processor = eventFlow {
        onEvent<Event.StartEvent> {
            logger.log("Start Event received with metadata: ${it.metadata}")
        }
        onEvent<Event.StopEvent> {
            logger.log("Stop Event received due to: ${it.reason}")
        }
        onEvent<Event.DataEvent> {
            logger.log("Data Event received: ${it.payload.size} bytes")
        }
    }

    with(processor) {
        runFlow(
            listOf(
                Event.StartEvent(mapOf("user" to "admin", "session" to "alpha")),
```

```
            Event.DataEvent("Hello World".encodeToByteArray()),
            Event.StopEvent("Completed Successfully")
        )
    )
  }
}
```

Why This Kotlin Sample Will Impress Tech Giant HRs

Feature	Why It Impresses
Custom Coroutine Dispatcher	Shows deep understanding of concurrency and thread management
DSL with Reified Generics	Demonstrates fluency in idiomatic Kotlin DSL and generics
Sealed Classes	Proper use for exhaustiveness and type safety
Dependency Injection	Manual DI using inline reified functions shows advanced language mastery
Multiplatform Expect/Actual	Shows readiness for cross-platform Kotlin Native and Android/iOS development
Reflection and Type-Safe Builders	Combines readability with power and compile-time safety
Composable DSL with context Receivers	Uses upcoming Kotlin 2.0+ features (rare and futuristic)
Extremely Readable Yet Powerful	Perfectly balanced clean architecture with expressive syntax

---

High-Ranked SEO Keywords Targeted

- kotlin advanced dsl

- custom coroutine dispatcher kotlin
- type-safe builders kotlin
- inline reified generics example
- kotlin multiplatform expect actual
- sealed class kotlin use case
- kotlin dependency injection without framework
- event driven architecture in kotlin
- build kotlin dsl from scratch
- kotlin coroutines with dispatcher

Key Technical Insights

- Inline reified generics replace reflection where possible for performance and type safety
- DSL pattern with sealed classes ensures compile-time validation of event handling
- Custom dispatcher avoids blocking main threads while managing async event loops
- Dependency Injection pattern helps isolate platform-specific logic (e.g., PlatformLogger)
- Multiplatform-ready architecture shows knowledge of Kotlin/Native development best practices