ANDROID KOTLIN

Day 2

Himbauan (Announcements or Reminders)

- **Mohon pastikan HP dalam mode Silent untuk menghindari gangguan.**
  - *Please ensure that cell phones are in Silent mode to avoid interruptions.*
- **Bersama-sama menjaga ketenangan selama sesi pembelajaran berlangsung.**
  - *Together, maintain quiet during the learning session.*
- **Mengikuti kegiatan dengan aktif dan berkontribusi secara positif.**
  - *Actively participate and contribute positively.*
- **Angkat tangan & tunggu dipanggil o/ pengajar untuk mengajukan pertanyaan.**
  - *Raise your hand & wait to be called by the instructor to ask questions.*
- **Ajukan pertanyaan yang relevan dengan topik yang sedang dibahas.**
  - *Ask questions relevant to the topic being discussed.*

Kotlin Syntax Basics

Sub topics:

- **Kotlin Syntax and Functions**
- **Dependency Injection**

Introduction to Kotlin Syntax

- "Kotlin is designed to be concise and expressive."
- "It reduces boilerplate code, making your codebase cleaner and more readable."
- "Syntax is influenced by other modern languages like Scala and Swift."

Basic Syntax Rules in Kotlin

- "Semicolons are optional: Kotlin does not require semicolons at the end of statements."
- "Variable declaration uses `val` for immutable and `var` for mutable variables."
- "Type inference allows you not to declare the type explicitly if it can be inferred by the compiler."

Example: (demo)

Control Structures in Kotlin

- "Kotlin supports standard control structures like if, when (switch), for, and while."
- "The `when` statement is a powerful replacement for switch, supporting complex expressions."
- "Kotlin's loops (for, while) are similar to other languages but offer more functionalities like iterating over a range or collection."

Example: (demo)

Defining Functions in Kotlin

- "Functions in Kotlin are declared using the `fun` keyword."
- "Kotlin supports default and named arguments, enhancing function usability."
- "Functions can return values or be unit (void) if they return nothing."

Example: (demo)

Higher-Order Functions and Lambdas

- "Higher-order functions are functions that take functions as parameters or return a function."
- "Lambdas are a concise way to represent functions inline."
- "These features are powerful for writing expressive code, particularly with collection operations."

Example: (demo)

Dependency Injection

Understanding Dependency Injection (DI)

- "Dependency Injection is a design pattern used to implement IoC (Inversion of Control), allowing for better decoupling of the construction and the use of objects."
- "In simpler terms, DI allows classes to define their dependencies without constructing them. These dependencies are then 'injected' at runtime by an external entity (e.g., a DI framework)."

Benefits of Using Dependency Injection

- "Simplifies the management of cross-cutting concerns."
- "Enhances modularity and makes the code more maintainable."
- "Improves the testability of software components."
- "Allows for greater flexibility and scalability of applications."

Understanding Cross-Cutting Concerns

- **Definition and Impact"**
  - *Cross-cutting concerns affect multiple components and functionalities, making them challenging to manage within traditional module boundaries.*
- **"Common Examples"**
  - *Includes logging, security, error handling, and performance monitoring, each integral to robust software but orthogonal to business logic.*
- **"Challenges: Scattering and Tangling"**
  - *Cross-cutting code often leads to scattered implementations and tangled logic, complicating maintenance and evolution.*
- **"Simplification through Dependency Injection"**
  - *DI provides a structured way to manage these concerns separately, enhancing modularity and maintainability.*

Implementing DI in Kotlin

- "Kotlin supports DI without requiring any special language features, primarily due to its interoperability with Java and concise syntax."
- "Common DI libraries used in Kotlin include Dagger, Koin, and Hilt (specifically tailored for Android)."

Using Dagger in Kotlin

- "Dagger is a fully static, compile-time DI framework that uses code generation."
- "It is known for its performance and precision in dependency management."
- "Dagger requires setup of components and modules to organize dependencies."

Example : (demo)

Simplifying DI with Koin

- "Koin is a pragmatic lightweight dependency injection framework for Kotlin developers."
- "It leverages Kotlin’s DSL and functional features for a simpler setup than Dagger."
- "Koin does not use code generation and is purely written in Kotlin."

Example: (demo)

Best Practices in Dependency Injection

- "Define clear boundaries for the scope of dependencies."
- "Prefer constructor injection to field injection for better immutability and testability."
- "Keep DI containers simple and avoid over-configuration."

Example: (demo)

utopio

THANK YOU