PAM **Pengembangan Aplikasi Mobile**

Laporan Praktikum 3

NIM
11S21010

Nama
Gabriel Cesar Hutagalung

Tanggal dibuat
13 February 2024, 15:11

score
100

Praktikum 3 (100%)

100 / 100

Code Analysis

20 / 20

A. Kotlin OOP

100 / 100

1. Class & Object

100 / 100

Catatan

Di Kotlin, seperti dalam banyak bahasa pemrograman lainnya, konsep dasar dari pemrograman berorientasi objek adalah "class" dan "object".

Class:
- **Class** adalah blueprint atau cetakan untuk membuat objek. Ia mendefinisikan atribut (variabel) dan metode (fungsi) yang akan dimiliki oleh objek yang dibuat dari kelas tersebut.
- Dalam sebuah class, Anda dapat mendefinisikan properti (atau atribut) dan metode. Properti adalah data yang disimpan dalam sebuah objek, sedangkan metode adalah perilaku atau tindakan yang dapat dilakukan oleh objek tersebut.
- Di Kotlin, definisi class dimulai dengan kata kunci "class", diikuti dengan nama kelas dan tubuh kelas di dalam kurung kurawal.

Contoh sederhana:
```kotlin
class Person {
var name: String = ""
var age: Int = 0

fun speak() {
println("Hello, my name is $name and I am $age years old.")
}
}
```

Object:
- **Object** adalah instance (kejadian) dari sebuah class. Ketika Anda membuat objek dari sebuah class, Anda membuat sebuah salinan dari blueprint tersebut yang dapat digunakan dalam program Anda.
- Objek adalah sesuatu yang nyata. Misalnya, jika kita memiliki sebuah class `Person`, kita bisa membuat objek `person1`, `person2`, dan seterusnya, yang mewakili orang-orang yang berbeda.

Contoh:
```kotlin
fun main() {
val person1 = Person()
person1.name = "John"
person1.age = 30
person1.speak() // Output: Hello, my name is John and I am 30 years old.
}
```

Dalam contoh di atas, `Person` adalah sebuah class yang memiliki dua properti (`name` dan `age`) dan satu metode (`speak`). Ketika kita membuat objek `person1`, kita memberikan nilai untuk properti-propertinya dan memanggil metodenya.

2. Encapsulation

100 / 100

Catatan

Encapsulation adalah salah satu konsep dalam pemrograman berorientasi objek yang mengacu pada pembungkusan atau pengelompokan data (variabel) dan metode (fungsi) yang beroperasi pada data tersebut ke dalam satu unit tunggal yang disebut class. Tujuan utama dari encapsulation adalah untuk menyembunyikan detail implementasi dari pengguna dan mendorong penggunaan data dan metode dengan cara yang aman dan terstruktur.

Dalam Kotlin, encapsulation dapat dicapai menggunakan properties (properti). Properties dalam Kotlin memungkinkan kita untuk mendefinisikan getter dan setter untuk mengakses dan mengubah nilai properti. Dengan menggunakan properties, kita dapat mengontrol akses ke data dan memastikan bahwa data hanya diakses atau diubah melalui metode yang ditentukan.

Berikut adalah contoh sederhana penggunaan encapsulation dalam Kotlin menggunakan properties:

```kotlin
class Car {
// Properti private untuk menyembunyikan akses langsung dari luar kelas
private var speed: Int = 0

// Getter untuk mendapatkan nilai speed
fun getSpeed(): Int {
return speed
}

// Setter untuk mengubah nilai speed
fun setSpeed(newSpeed: Int) {
speed = newSpeed
}
}

fun main() {
val car = Car()

// Mengatur nilai speed menggunakan setter
car.setSpeed(60)

// Mendapatkan nilai speed menggunakan getter
println("Current speed: ${car.getSpeed()}") // Output: Current speed: 60
}
```

Dalam contoh di atas, properti `speed` dienkapsulasi di dalam kelas `Car`. Karena properti `speed` adalah private, ia hanya dapat diakses dan diubah melalui metode `getSpeed()` dan `setSpeed()` yang telah ditentukan dalam kelas `Car`. Ini membantu dalam mencegah akses langsung ke properti `speed` dari luar kelas dan memastikan bahwa perubahan nilai dilakukan secara terkontrol. Ini adalah contoh sederhana dari bagaimana encapsulation dapat diterapkan dalam Kotlin untuk memastikan keamanan data dan mempromosikan desain yang bersih dan terstruktur.

3. Inheritance

100 / 100

Catatan

All classes in Kotlin have a common superclass, Any, which is the default superclass for a class with no supertypes declared:

class Example // Implicitly inherits from Any
Any has three methods: equals(), hashCode(), and toString(). Thus, these methods are defined for all Kotlin classes.

By default, Kotlin classes are final – they can't be inherited. To make a class inheritable, mark it with the open keyword:

open class Base // Class is open for inheritance
To declare an explicit supertype, place the type after a colon in the class header:

open class Base(p: Int)

class Derived(p: Int) : Base(p)
If the derived class has a primary constructor, the base class can (and must) be initialized in that primary constructor according to its parameters.

4. Polymorphism

100 / 100

Catatan

In Kotlin, polymorphism refers to **the ability of a function or method to take on multiple forms depending on the context in which it is called**. It is one of the fundamental principles of object-oriented programming. Kotlin, being an object-oriented programming language, supports both static and dynamic polymorphism.

5. Interface

100 / 100

Catatan

Interfaces in Kotlin can contain declarations of abstract methods, as well as method implementations. What makes them different from abstract classes is that interfaces cannot store state. They can have properties, but these need to be abstract or provide accessor implementations.

6. ObjectCompanion

100 / 100

Catatan

Di Kotlin, jika ingin menulis sebuah fungsi atau member variabel di suatu kelas agar bisa dipanggil tanpa melalui sebuah objek, kita dapat melakukannya dengan menulis member atau method tersebut di dalam companion object

suatu kelas. Jadi, dengan mendeklarasikan companion object, kita bisa mengakses member dari suatu kelas tanpa melalui objek.

Menulis sebuah companion object dapat dilakukan dengan menggunakan keyword companion object diikuti dengan namanya.

7. Package

100 / 100

Catatan

All the contents, such as classes and functions, of the source file are included in this package. So, in the example above, the full name of `printMessage()` is `org.example.printMessage`, and the full name of `Message` is `org.example.Message`.

If the package is not specified, the contents of such a file belong to the *default* package with no name.

B. Studi Kasus Aplikasi Todolist

100 / 100

1. Membuat Entity

100 / 100

Catatan

membuat entity berguna untuk menyimpan data sementara yang nantinya akan digunakan pada todolist kita. Entity ini merupakan salah satu bagian dari konsep clean architecture.

2. Membuat Repository

100 / 100

Catatan

Dalam Clean Architecture, repository adalah salah satu dari beberapa lapisan (layer) yang digunakan untuk mengelola akses data. Repository bertanggung jawab untuk memisahkan logika bisnis dari teknologi penyimpanan data yang spesifik.

3. Membuat Service

100 / 100

Catatan

Dalam konteks Clean Architecture, "service" adalah salah satu dari beberapa konsep yang membantu dalam memisahkan tanggung jawab dan menjaga struktur yang terorganisir dalam aplikasi. Namun, perlu dicatat bahwa istilah "service" bisa memiliki beberapa makna tergantung pada konteksnya.

4. Membuat Util

100 / 100

Catatan

Dalam konteks Clean Architecture, "util" (singkatan dari utility) dapat merujuk pada berbagai hal tergantung pada kebutuhan dan konteks aplikasi. Secara umum, util atau utilitas adalah kelas atau fungsi yang menyediakan

fungsionalitas umum yang dapat digunakan di berbagai bagian aplikasi. Namun, penggunaan utilitas dalam Clean Architecture perlu diperhatikan agar tidak melanggar prinsip-prinsip desainnya.

5. Membuat View

100 / 100

Catatan

Dalam Clean Architecture, "View" merujuk pada bagian dari aplikasi yang bertanggung jawab untuk menampilkan antarmuka pengguna kepada pengguna dan mengumpulkan masukan dari pengguna. Lapisan View ini terletak di bagian paling luar dari arsitektur, yang merupakan antarmuka pengguna langsung.

6. App

100 / 100

Catatan

Dalam Clean Architecture, "app" merujuk pada bagian utama dari aplikasi yang meliputi semua komponen yang diperlukan untuk menjalankan aplikasi, termasuk lapisan presentasi, lapisan bisnis, lapisan data, dan lapisan infrastruktur. Namun, dalam konteks Clean Architecture, "app" bukanlah lapisan tertentu, melainkan keseluruhan struktur dan komponen yang membentuk aplikasi.

Code Auto Grader

80 / 80

1. Aplikasi Todolist

100 / 100

Test Case	Status	Score
TC1	PASS	20 / 20
TC2	PASS	20 / 20
TC3	PASS	20 / 20
TC4	PASS	20 / 20
TC5	PASS	20 / 20
Total Score		100 / 100