**Rangkuman dari Kotlin Object-Oriented Programming**

Okay, kita sudah selesai belajar dan mengetahui mengenai paradigma Object-Oriented pada Kotlin. Untuk me-refresh apa yang sudah kita pelajari, mari kita merangkum beberapa hal penting mengenai paradigma ini.

* **Object-Oriented Programming** masih menjadi salah satu paradigma atau teknik pemrograman yang banyak digunakan dalam pengembangan aplikasi.
* Gambaran umum dari Object-Oriented Programming adalah sebuah **blueprint**, di mana *blueprint* ini adalah sebuah **Class** (kelas).
* Pada Object-Oriented Programming terdapat beberapa komponen penting yaitu **Object**, **Class**, **Attribute** dan **Behaviour**.
* **Object** atau Objek merupakan hasil realisasi dari sebuah *blueprint* atau *class* yang tentunya memiliki fungsi dan juga properti sama seperti *blueprint*-nya. Salah satu contoh Object yang sering kita pakai adalah nilai primitif pada Kotlin, yaitu **String**, **Integer**, **Char** dan **Boolean**. Kegunaan dari Objek sendiri adalah untuk mengakses berbagai properti dan fungsi pada kelas.
* **Class** merupakan sebuah *blueprint*. Setiap kelas memiliki atribut dan *behaviour*. Dalam Kotlin, attributes lebih sering disebut dengan *properties*, sedangkan *behaviour* sering disebut *functions*. Berikut penjelasan singkat dari setiap komponen:
  + **Class**: Merupakan sebuah blueprint yang terdapat properti dan fungsi di dalamnya.
  + **Properties**: Karakteristik dari sebuah kelas, memiliki tipe data.
  + **Functions**: Kemampuan atau aksi dari sebuah kelas.
* Konstruktor merupakan fungsi spesial yang digunakan untuk menginisialisasi properti yang terdapat pada sebuah kelas.
* Terdapat 3 (tiga) tipe konstruktor pada Kotlin, yaitu **primary constructor**, **secondary constructor** dan **default constructor**.
  + **Primary Constructor**
    - Seperti namanya, jika kita akan membuat suatu objek dari sebuah kelas dan kelas tersebut memiliki *constructor* di dalamnya, maka konstruktor tersebut adalah *primary constructor* dan diharuskan untuk mengirim nilai sesuai properti yang dibutuhkan.
    - Primary Constructor juga dapat memiliki nilai *default*, dengan begitu jika kita tidak menetapkan nilai untuk parameter tersebut maka properti tersebut akan memiliki nilai *default*.
    - Kotlin menyediakan blok *init* yang memungkinkan kita untuk menuliskan properti di dalam *body class* ketika kita menggunakan *primary* *constructor*.
  + **Secondary Constructor**
    - Secondary constructor digunakan ketika kita ingin menginisialisasi sebuah kelas dengan cara yang lain.
    - Dapat memiliki lebih dari satu *secondary constructor*.
  + **Default Constructor**
    - Kotlin secara otomatis membuat sebuah *default constructor* pada kelas jika kita tidak membuat sebuah konstruktor secara manual.
* Teknik membuat sebuah kelas yang memang bertugas untuk mengatur atau mengelola fungsi *getter* dan *setter* untuk sebuah properti kelas pada Kotlin dinamakan **Delegate**.
* **Extension** **properties**pada Kotlin sama halnya seperti melakukannya pada Extension function. Kita dapat menambahkan sebuah properti tanpa harus membuat sebuah kelas yang mewarisi kelas tersebut. Hal ini dilakukan dengan deklarasi khusus yang disebut dengan Extension.
* **Visibility Modifier** adalah **hak akses**pada Kotlin. Dengan menentukan hak akses tersebut, kita dapat membatasi akses data pada sebuah kelas. Berikut macam-macam hak akses dan penjelasan singkatnya yang dapat digunakan pada Kotlin:
  + **Public**: Hak akses yang cakupannya paling luas. Anggota yang diberi *modifier* ini dapat diakses dari manapun.
  + **Private**: Hak akses yang cakupannya paling terbatas. Anggota yang menerapkannya hanya dapat diakses pada *scope* yang sama.
  + **Protected**: Hak akses yang cakupannya terbatas pada hirarki kelas. Anggota hanya dapat diakses pada kelas turunannya atau kelas itu sendiri.
  + **Internal**: Hak akses yang cakupannya terbatas pada satu modul. Anggota yang menggunakannya tidak dapat diakses diluar dari modulnya tersebut.
* Pada Object-Oriented Programming terdapat **Inheritance**/Pewarisan. Inheritance dapat mencegah kita melakukan perulangan kode dengan cara mengelompokkan properti dan fungsi yang sama. Pada Inheritance, terdapat **parent class** dan juga **child class.**
* Pada Kotlin menggunakan dua atau lebih fungsi dengan nama yang sama disebut dengan **overloading**. Overloading dapat dilakukan selama fungsi itu memiliki parameter yan berbeda.
* **Abstract Class** merupakan gambaran umum dari sebuah kelas. Abstract Class tidak dapat direalisasikan dalam sebuah objek.
* **Interfaces** merupakan suatu konsep sifat umum yang nantinya digunakan oleh suatu kelas agar dapat memiliki sifat tersebut.
* Seluruh konten pada Kotlin, seperti kelas dan fungsi, dibungkus dalam sebuah *package*. Package tersebut digunakan untuk mengelompokkan kelas, fungsi dan variabel yang mempunyai kemiripan fungsionalitas.
* **Exception** adalah event (kejadian) yang dapat mengacaukan jalannya suatu program. Pada Kotlin semua *exception* bersifat **Unchecked**, yang artinya *exception* terjadi karena kesalahan pada kode kita. Kode yang baik yaitu kode yang terhindar dari segala bentuk kejadian dengan efek buruk pada aplikasi kita. Kejadian tersebut pada programming disebut **Exception**. Hal seperti ini seharusnya dihindari. Oleh karena itu kita harus mengetahui cara menangani suatu *exception* (**Exception Handling**).
* Berikut ini beberapa contoh Unchecked Exception yang sering mengganggu jalannya program kita:
  + **ArithmeticException**
    - Merupakan *exception* yang terjadi karena kita **membagi** suatu bilangan dengan nilai **nol**.
  + **NumberFormatException**
    - Disebabkan karena terjadi kesalahan dalam **format** angka.
  + **NullPointerException**
    - Terjadi karena sebuah variabel atau objek memiliki nilai **null**, padahal seharusnya objek atau variabel tersebut **tidak boleh** null.
* **Exception** **Handling** dapat diterapkan dengan beberapa cara. Di antaranya adalah dengan menggunakan **try-catch**, **try-catch-finally**, dan **multiple catch**.