BAHASA PEMROGRAMAN KOTLIN
I

BAHASA PEMROGRAMAN KOTLIN
I
Dasar

P. Tamami
BPPKAD Kab. Brebes

Untuk Istriku yang selalu
memberi semangat, dan
anak-anak yang selalu
ceria

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

KATA PENGANTAR

Saat melihat keunggulan dari bahasa pemrograman Java yang mudah untuk di*maintenance*, dapat berjalan di berbagai *platform*, berorientasi objek, dan beberapa keunggulan lain, ada beberapa penyempurnaan yang dilakukan oleh bahasa pemrograman Kotlin, yang sama-sama berjalan di atas JVM.

Dalam buku ini akan dijelaskan dasar dari pemrograman Kotlin yang menawarkan penulisan kode yang lebih ringkas, menjamin kesalahan seluruh kelas dari *exception* **null**, dan yang tidak kalah penting adalah integrasinya dengan sistem yang dibangun dengan menggunakan bahasa Java.

Silahkan menikmati buku yang kurang dari sempurna ini, dan berharap penulis mendapatkan kritik yang membangun guna perubahan isi buku ini ke arah yang lebih sempurna.

4 Mei 2017

Penulis

BAB 1

MEMULAI

Perlu diketahui bahwa Kotlin ini adalah bahasa pemrograman yang berjalan di atas JVM, sehingga diperlukan Java Runtime untuk menjalankannya.

Cara termudah untuk memasangkan atau meng*install compiler* Kotlin adalah dengan mengunduh di halaman https://github.com/JetBrains/kotlin/releases/, kemudian melakukan *unzip* dan menambahkan direktori `bin` ke dalam *path* sistem.

Untuk memastikan bahwa Kotlin sudah terpasang dan dapat digunakan, kita seharusnya dapat menjalankan perintah berikut di konsol pada Linux atau *command prompt* milik Windows, berikut perintahnya :

```
1 kotlinc -version
```

Perintah tersebut sebetulnya untuk mencetak informasi tentang versi *compiler* Kotlin yang aktif. Dan seharusnya akan muncul informasi yang kurang lebih sebagai berikut :

```
1 info: Kotlin Compiler version 1.1.2-2
```

Tentunya versi yang keluar akan berbeda tergantung apa yang kita *install*.

Percobaan berikutnya adalah menampilkan versi *runtime environment* dari Kotlin, jika perintah `kotlinc` digunakan untuk melakukan *compile* (kompilasi) terhadap

kode yang kita ketik / tulis menjadi bahasa biner, fungsi dari *runtime environment* adalah menerjemahkan bahasa biner hasil *compile* oleh `kotlinc` menjadi bahasa *native* sesuai sistem operasi yang digunakan, inilah prinsip yang digunakan bahasa pemrograman Java yang tetap digunakan oleh Kotlin, karena memang Kotlin masih menggunakan JRE (*Java Runtime Environment*).

Perintah untuk melihat versi *runtime environment* dari Kotlin adalah sebagai berikut :

```
kotlin -version
```

Dengan hasil keluaran di layar monitor seperti ini :

```
Kotlin version 1.1.2-2 (JRE 1.8.0_121-b13)
```

Versi Kotlin seharusnya sama dengan versi *compiler*-nya. Sedangkan muncul tambahan informasi `JRE 1.8.0_121-b13`, inilah yang menunjukan bahwa Kotlin masih menggunakan JRE untuk menjalankan programnya, karena memang sebelum melakukan instalasi Kotlin, Java harus di*install* terlebih dahulu.

1.1 Katakan Hai

Setelah melakukan percobaan dasar seperti di atas, kita akan mencoba menjalankan kode pertama yang kita buat dengan Kotlin. Berikut adalah langkahnya :

1. Membuka editor teks seperti notepad, atom, notepad++, atau aplikasi sejenis.
2. Mengetikan kode berikut :

```
fun main(args: Array<String>) {
    println("Hai, selamat datang")
}
```

3. Simpanlah dengan nama apapun, berikan ekstensi `kt`, misal kita beri nama *file* tersebut dengan `Test.kt`.
4. Buka konsol atau *command prompt* dan aktifkan ke direktori tempat kita simpan *file* `Test.kt` tadi.
5. *Compile file* `Test.kt` tersebut dengan perintah berikut :

```
kotlinc Test.kt
```

6. Hasil dari *compile* tersebut adalah berupa *file* `TestKt.class`
7. Untuk menjalankan hasil program yang telah kita *compile*, gunakan perintah berikut :

```
kotlin TestKt
```

8. Kemudian akan program / aplikasi akan menghasilkan keluaran sebagai berikut :

```
1 Hai, selamat datang
```

9. Sampai titik ini, kita berhasil menjalankan kode yang telah kita buat.

Jadi sebetulnya, untuk memulai koding dengan bahasa Kotlin cukup sederhana, tinggal siapkan *berekstensi* `kt`, kemudian sertakan blok kode program berikut :

```
1 fun main(args: Array<String>) {
2   ...
3 }
```

Seluruh program yang dibangun dengan Kotlin akan berawal dari fungsi `main` ini.

1.2 Sintak Dasar

1.2.1 Deklarasi Paket

Sama seperti bahasa pemrograman Java, deklarasi paket berada di awal kode seperti contoh berikut :

```
1 package nama.paket
2
3 import java.net.*
4 ...
```

Perbedaannya adalah bahwa nama paket tidak perlu disesuaikan atau disamakan dengan nama direktorinya seperti pada pemrograman Java. *File* kode sumber dapat ditempatkan dimanapun pada *drive*.

1.2.2 Deklarasi Fungsi

Deklarasi fungsi tanpa parameter dan tanpa nilai balikkan (*return*) akan terlihat seperti contoh kode berikut :

```
1 fun cetak(): Unit {
2   println("Hai, apa kabar")
3 }
```

Atau deklarasi `Unit` dapat dihilangkan dengan kode akan terlihat seperti ini :

```
1 fun cetak() {
2   println("Hai, apa kabar")
3 }
```

Untuk deklarasi fungsi dengan parameter akan terlihat seperti contoh kode berikut :

```
1 fun tambah(a: Int, b: Int): Int {
2   return a + b
3 }
```

Fungsi yang sama seperti diatas dapat dibuat lebih ringkas dengan nilai balikan *return* yang sudah diprediksi oleh Kotlin, kodenya menjadi seperti berikut ini :

```
1 fun tambah(a: Int, b: Int) = a + b
```

Untuk pembahasan lebih lanjut mengenai fungsi, akan dijabarkan dalam bagian tersendiri dalam buku ini.

1.2.3 Deklarasi Variabel

Deklarasi variabel dapat dilakukan untuk 2 (dua) cara. Yang pertama adalah variabel yang hanya dapat diisi satu kali, dan ada yang dapat diisi berkali-kali.

Kode untuk deklarasi variabel yang hanya dapat diisi 1 (satu) kali adalah sebagai berikut :

```
1 val a: Int = 2
2 // atau
3 val c = 2
4 // atau
5 val d: Int
6 d = 5
```

Untuk deklarasi variabel yang dapat diubah, kodenya adalah sebagai berikut :

```
1 var e = 2
2 e *= 2
```

1.2.4 Deklarasi Komentar

Seperti bahasa pemrograman Java dan Javascript, Kotlin juga menyediakan komentar dalam bentuk komentar baris dan komentar multi-baris. Kode untuk komentar satu baris adalah sebagai berikut :

```
1 // ini komentar 1 baris
```

Untuk kode komentar multi-bari adalah sebagai berikut :

```
1 /*  ini komentar
2     multi baris */
```

Namun tidak seperti bahasa pemrograman Java, komentar di Kotlin dapat bersarang bertingkat.

1.3 Logat

Beberapa logat yang biasa digunakan di Kotlin adalah seperti di bawah ini.

1.3.1 Membuat Kelas Data

Kelas data ini biasa digunakan untuk pembuatan kelas *entity*. Contoh kodenya adalah sebagai berikut :

```
1 data class Pegawai(val nim: String, val nama: String)
```

Dengan menambahkan deklarasi `data` di depan kelas, maka untuk kelas Pegawai ini akan disediakan fungsi-fungsi berikut secara otomatis :

- *Getters* dan *Setter* untuk seluruh properti
- *Method* `equals`.
- *Method* `hashCode`
- *Method* `toString`
- *Method* `copy`

1.3.2 Nilai *Default* Untuk Parameter Fungsi

Pada saat deklarasi fungsi, sebetulnya parameter dapat kita isikan dengan nilai *default* seperti berikut :

```
1 fun isiData(nama: String, kelamin: Int = 0) {
2   ...
3 }
```

Nantinya parameter `kelamin` akan terisi otomatis dengan `0`
Contoh kodenya adalah sebagai berikut :

```
1 fun main(args: Array<String>) {
2   val nama = "tamami"
3   println("Halo, $nama")
4
5   isiData(nama)
6 }
7
8 fun isiData(nama: String, kelamin: Int = 0) {
9   println(kelamin)
10 }
```

Hasil keluarannya adalah sebagai berikut :

```
1 Halo, tamami
2 0
```

Penjelasannya adalah sebagai berikut, pada baris pertama menghasilkan keluaran teks `Halo, tamami`, yang sebetulnya hasil dari eksekusi perintah kode pada baris ke-3, yaitu :

```
1 println("Halo, $nama")
```

Dimana pemanggilan variabel `$nama` pada baris ke-2 dari *source code* terjadi, dan yang ditampilkan di layar monitor adalah isi dari variabel `$nama`, yaitu `tamami`.
Sedangkan pada baris kedua dari hasil keluaran, yaitu `0`, adalah hasil dari eksekusi kode pada bari ke-9, di dalam fungsi `isiData`, tepatnya pada perintah berikut :

```
1 println(kelamin)
```

Kenapa hasil keluarannya adalah `0`, alurnya adalah seperti ini, pada saat pemanggilan fungsi `isiData(nama)` pada baris ke-5, parameter `nama` pada fungsi `isiData` ini terisi dengan nilai `tamami`, karena parameter kedua, yaitu `kelamin` tidak disertakan pada pemanggilannya pada baris ke-5, sehingga parameter `kelamin` akan terisi otomatis dengan nilai `0` sebagaimana deklarasinya pada baris ke-8.

1.3.3 *Interpolasi* Teks

Interpolasi atau penyisipan teks akan terlihat seperti baris perintah berikut ini :

```
1 val nama: String = "tamami"
2 println("name $nama")
```

Nantinya sisipan teks dengan kode `$nama` akan terisi oleh variabel `nama` yang telah dideklarasikan sebelumnya.

1.3.4 Pemeriksaan Instan

Pada bahasa Kotlin, kita dapat melakukan pemeriksaan tipe data secara instan, formatnya adalah seperti kode berikut :

```
1 when(x) {
2   is String -> ...
3   is Int -> ...
4   is KelasSaya -> ...
5   else -> ...
6 }
```

Artinya nanti isi dari variabel `x` akan dipilah, apakah merupakan tipe data `String`, `Int`, merupakan instan dari kelas `KelasSaya`, atau berupa tipe data atau kelas lain.

Contoh nyata dari penggunaan kode di atas adalah sebagai berikut :

```
1 fun main(args: Array<String>) {
2   val x: Any = 2
3
4   when (x) {
5     is String -> println("Jawaban String")
6     is Int -> println("Jawaban Int")
7     else -> println("lainnya")
8   }
9 }
```

Pada kode di atas, tipe data dari variabel `x` adalah `Any`, yang artinya bisa berupa tipe data apapun, atau instan dari kelas apapun. Lalu diisikan nilai awal berupa angka 2 (dua).

Selanjutnya kode akan melakukan seleksi tipe data `x` pada baris ke-4 dengan perintah `when (x)`, kemudian melakukan pemeriksaan, apabila tipe data dari variabel `x` adalah `String` maka akan dicetak seperti pada baris ke-5 dari kode di atas, tapi ternyata memang tipe data yang tepat adalah pada baris ke-6 sehingga program yang kita bangun akan mencetak `Jawaban Int` di layar, karena variabel `x` berisi angka 2 (dua).

1.3.5 Penggunaan *Range*

Penggunaan *range* biasanya untuk melakukan iterasi atau perulangan, beberapa contohnya akan dikerjakan dengan kode berikut :

```
1 for(i in 1..10) { println("data ke-$i = $i") }
```

Kode tersebut akan menghasilkan keluaran di monitor dimana nilai `i` dari 1 sampai dengan 10 sebagai berikut :

```
1 data ke-1 = 1
2 data ke-2 = 2
3 data ke-3 = 3
4 data ke-4 = 4
5 data ke-5 = 5
6 data ke-6 = 6
7 data ke-7 = 7
8 data ke-8 = 8
9 data ke-9 = 9
10 data ke-10 = 10
```

Contoh penggunaan *range* yang lain adalah seperti kode berikut :

```
1 for(i in 1 until 10) { println("data ke-$i = $i") }
```

Sama seperti kode sebelumnya, hanya saja kali ini nilai `i` adalah antara 1 sampai dengan 9, angka 10 tidak masuk dalam kualifikasi proses cetak ke monitor. Berikut adalah hasil keluarannya di monitor :

```
1 data ke-1 = 1
2 data ke-2 = 2
3 data ke-3 = 3
4 data ke-4 = 4
5 data ke-5 = 5
6 data ke-6 = 6
7 data ke-7 = 7
8 data ke-8 = 8
9 data ke-9 = 9
```

Contoh penggunaan *range* dengan beberapa pola lompatan atau kelipatan angka adalah sebagai berikut :

```
1 for(i in 1..10 step 3) { println("data ke-$i") }
```

Maksudnya adalah mencetak deretan angka yang dimulai dari 1 dengan kelipatan 3 sampai nilai `i` sama dengan 10. Berikut adalah hasil keluaran dari kode tersebut :

```
1 data ke-1
2 data ke-4
3 data ke-7
4 data ke-10
```

Contoh penggunaan *range* untuk perulangan yang mundur ke nilai yang lebih kecil adalah sebagai berikut :

```
1 for(i in 10 downTo 1) { println("data ke-$i") }
```

Penjelasan dari kode tersebut dijelaskan dengan hasil keluaran di layar monitor sebagai berikut :

```
data ke-10
data ke-9
data ke-8
data ke-7
data ke-6
data ke-5
data ke-4
data ke-3
data ke-2
data ke-1
```

1.3.6 *Read-only List*

Artinya adalah membuat *list* yang tidak dapat diubah isinya, contoh deklarasinya adalah sebagai berikut :

```
val list = listOf("data A", "data B", "data C")
```

Untuk lebih jelasnya, kita akan melihat kode contoh sebagai berikut :

```
fun main(args: Array<String>) {
  val list = listOf("A", "B", "C");

  for(i in list) {
    println("data $i")
  ]
}
```

Kode tersebut, pada baris ke-2 adalah menyiapkan objek *list* dan diisikan langsung dengan data menggunakan perintah `listOf`, selanjutnya seluruh data dicetak ke layar monitor sebagaimana tampilan berikut :

```
data A
data B
data C
```

1.3.7 *Read-only Map* dan Cara Mengaksesnya

Map sebetulnya adalah kelas koleksi yang berisi pasangan kunci (*key*) dan isi (*value*), contohnya adalah sebagai berikut :

```
val map = mapOf("key1" to "nilai1", "key2" to "nilai2", "key3" to "
    nilai3")
```

Untuk melakukan akses data pada *map* ini adalah sebagai berikut :

```
println(map["key"]) // untuk mengambil nilainya
map["key"] = nilai // untuk mengganti atau mengisi nilai
```

Kode lengkap untuk percobaan *Map* ini adalah sebagai berikut :

```
fun main(args: Array<String>) {
  val map = mapOf("key1" to "nilai1", "key2" to "nilai2", "key3" to "
    nilai3")

```

```
4   println(map["key2"])
5
6   for(i in map) {
7     println(i)
8   }
9 }
```

Hasil keluarannya adalah sebagai berikut :

```
1 nilai2
2 key1=nilai1
3 key2=nilai2
4 key3=nilai3
```

Hasil dari kode pada baris ke-4 adalah hasil keluaran pada baris ke-1, sedangkan hasil iterasi atau perulangan dari kode pada baris ke-6 sampai dengan baris ke-8 adalah pada baris ke-2 sampai dengan baris ke-4 pada hasil keluaran.

1.3.8 Jalan Pintas Perintah `if not null`

Ini digunakan sebetulnya untuk melakukan pemeriksaan terhadap isi dari suatu variabel apakah terisi atau tidak (*null*), kodenya adalah sebagai berikut :

```
1 import java.io.File
2
3 fun main(args: Array<String>) {
4   val file = File("Test.kt").listFiles()
5
6   println(file ?. getTotalSpace())
7 }
```

Pada kode tersebut kita melihat bahwa aplikasi yang kita bangun menggunakan pustaka Java yaitu `java.io.File`, hal ini memungkinkan karena memang Kotlin adalah turunan dari Java yang dapat menggunakan kelas-kelas Java untuk membangun aplikasi. Ini adalah salah fasilitas yang Kotlin sediakan.

Yang perlu diperhatikan adalah pada baris ke-6, yaitu pada bagian `file?.getTotalSpace()`, dimana tanda tanya (`?`) yang ada pada bagian ini menunjukan seleksi atau pemeriksaan terhadap kondisi *null*, atau kosongnya isi dari variabel yang diminta.

Bila isi variabel `file` ada isinya, maka akan dicetak ukuran dari *file* tersebut, namun bila nihil, maka akan mencetak `null` ke layar monitor.

Sebagai contoh apabila isi dari variabel `file` adalah `null` adalah sebagai berikut :

```
1 import java.io.File
2
3 fun main(args: Array<String>) {
4   val file: File?
5   file = null
6   println(file?.getTotalSpace())
7 }
```

Hasil dari kode di atas adalah informasi `null` yang dicetak ke layar monitor.

1.3.9 Jalan Pintas Perintah `if not null and else`

Cara ini sebetulnya pengembangan dari kasus sebelumnya, dengan penambahan penanganan apabila memang hasilnya adalah `null`, berikut kodenya :

```
1 import java.io.File
2
3 fun main(args: Array<String>) {
4   val file: File?
5   file = null
6   println(file ?. getTotalSpace() ?: "NIHIL")
7 }
```

Yang perlu diperhatikan adalah pada baris ke-6, dimana ada tambahan tanda (`?:`), apabila variabel `file` tidak bernilai `null` maka akan dipanggil *method* `getTotalSpace()`, namun bila bernilai `null` maka akan dicetak kata `NIHIL`.

1.3.10 Eksekusi Perintah `if null`

Kondisi ini digunakan untuk melakukan eksekusi bila nilai dari suatu variabel bernilai `null`. Contoh kode untuk kasus ini adalah sebagai berikut :

```
1 fun main(args: Array<String>) {
2   val data = mapOf("nama" to "tamami", "umur" to 36)
3   val email = data["email"] ?: throw IllegalStateException("Email
      tidak ada")
4 }
```

Inti dari kode ini sebetulnya ada pada baris ke-3, pada saat variabel `email` akan diisikan sebuah nilai yang diambilkan dari variabel *map* `data` berupa *email*, namun bila nilai untuk *email* ini `null` maka akan dilemparkan keluar sebagai `IllegalStateException`.

Hasil keluaran dari kode tersebut tentu saja sebuah *exception* dengan pesan berupa `Email tidak ada`.

1.3.11 Eksekusi Perintah `if not null`

Kondisi ini adalah kebalikan dari eksekusi sebelumnya, yaitu bila nilainya tidak `null` maka akan melakukan eksekusi yang tertuang dalam blok kode. Berikut ada contoh kodenya :

```
1 fun main(args: Array<String>) {
2   val data = mapOf("nama" to "tamami", "umur" to 33)
3   data?.let {
4     println(data["nama"])
5   }
6 }
```

Blok kode untuk kondisi ini sebetulnya ada pada baris ke-3 sampai dengan baris ke-5, pada baris ke-3, akan dilakukan pemeriksaan, apakah variabel `data` berupa `null` atau bukan, bila bukan `null` maka akan dikerjakan perintah pada baris ke-4. Bila berupa `null` maka blok kode ini akan dilewati begitu saja.

1.3.12 Kembalikan Pada Perintah `when`

Pada bagian ini, sebuah *method* akan mengembalikan nilai dari hasil seleksi pada perintah `when`. Berikut kode lengkapnya :

```
1  fun main(args: Array<String>) {
2    println("Kode warna " + kodeWarna("hijau"))
3  }
4
5  fun kodeWarna(warna: String): Int {
6    return when(warna) {
7      "merah" -> 1
8      "hijau" -> 2
9      "biru" -> 3
10     else -> throw Exception("Salah Warna")
11   }
12 }
```

Pada bagian ini intinya ada pada *method* atau fungsi `kodeWarna`, yaitu bila parameter `warna` terisi oleh teks `merah` maka akan mengembalikan nilai `1`, bila terisi oleh teks `hijau` maka akan mengembalikan nilai `2` dan seterusnya.

Pada baris ke-2, karena dipanggil fungsi `kodeWarna` dengan parameter `warna` berisi teks `hijau`, maka hasil keluaran yang terjadi di layar monitor adalah sebagai berikut :

```
1 Kode warna 2
```

1.3.13 Ekspresi `try catch`

Pada bagian ini ditujukan untuk penanganan *exception* dengan `try catch` agar aplikasi yang dibangun lebih bersih dari informasi kesalahan yang diproduksi oleh bahasa pemrograman. Contoh kode untuk penjelasannya adalah sebagai berikut :

```
1  fun main(args: Array<String>) {
2    try {
3      println("kode warna : " + kodeWarna("ungu"))
4    } catch(e: Exception) {
5      println("error : " + e.printStackTrace())
6    }
7  }
8
9  fun kodeWarna(warna: String): Int {
10   return when(warna) {
11     "merah" -> 1
12     "hijau" -> 2
13     "biru" -> 3
14     else -> throw Exception("Salah Warna")
15   }
16 }
```

Pada baris ke-3, sengaja diberikan isi teks `ungu` pada parameter fungsi `kodeWarna` agar *exception* yang ada dapat dipicu untuk dieksekusi.

Hasilnya adalah karena parameter teks `ungu` yang dikirim, maka yang diterima adalah *exception* pada baris ke-14, sehingga kode akan langsung dilempar ke baris ke-5 karena pemanggilan fungsi `kodeWarna` menghasilkan *exception* tersebut.

1.3.14 Ekspresi `if`

Pada bagian ini, sebetulnya berfungsi untuk kondisional biasa, untuk memilah sebagaimana bentuk *when* sebelumnya. Contoh kodenya adalah sebagai berikut :

```
1 fun main(args: Array<String>) {
2   val i = 37
3
4   if(i % 2 == 0) {
5     println("itu bilangan genap")
6   } else {
7     println("itu bilangan ganjil")
8   }
9 }
```

Tentu saja hasil dari kode tersebut akan mencetak baris teks berikut :

```
1 itu bilangan ganjil
```

Karena memang nilai dari variabel `i` diperiksa, apabila setelah dibagi dengan angka 2 tidak memiliki sisa bagi, maka itu pertanda bahwa nilai `i` adalah bilangan genap, namun bila selain dari itu, akan dieksekusi perintah dalam blok `else`.

1.4 Adat

Pada tiap bahasa pemrograman, ada sebuah adat atau kebiasaan atau *style* dari kode agar tetap mudah terbaca bagi orang lain yang membuka kode sumbernya, begitu pula di Kotlin, ada beberapa kebiasan atau adat yang perlu kita latih agar kode kita mudah untuk dibaca oleh orang lain. Berikut adalah diantaranya :

- Gaya Penamaan

  Gaya penamaan akan mirip seperti bahasa Java, yaitu :

  - menggunalan `camelCase` untuk semua nama variabel atau fungsi (diusahakan menghilangkan garis bawah (_))
  - tipe atau kelas diawali dengan huruf besar
  - fungsi dan properti diawali dengan huruf kecil
  - menggunakan 4 spasi untuk indentasi
  - fungsi yang bersifat public harus memiliki dokumentasi seperti yang muncul pada dokumentasi Kotlin

- Titik Dua

  Penggunaan titik dua menggunakan spasi sebelum tanda titik dua ketika memisahkan tipe dan super-tipe, dan tidak menggunakan spasi sebelum tanda titik dua ketika memisahkan instan dan tipe. Berikut contohnya :

```
1 interface MyInterface<out T : Any> : Bar {
2   fun myFungsi(a: Int): T
3 }
```

- Lambda

  Ekspresi Lambda atau disebut labmda saja sebetulnya adalah fasilitas yang memudahkan penulis kode program untuk mempersingkat kode. Penulisannya seharusnya tiap penggunaan kurung kurawal buka dan tutup, disertakan spasi sebelum dan sesudahnya. Begitu lupa perlakukan untuk tanda panah yang memisahkan antara parameter dan isi lambda, harus diberikan spasi sebelum dan sesudahnya.

  Contoh sederhananya adalah sebagai berikut :

```
1 fun main(args: Array<String>) {
2   val list = listOf(1, 2, 3, 4, 5)
3   println(list.filter { it >= 4 }
4 }
```

  Inti dari kode diatas sebetulnya ada pada baris ke-3, pada bagian textttit ¿= 4, yang menghasilkan hanya isi yang nilainya lebih dari atau sama dengan 4. Hasil dari kode di atas adalah sebagai berikut :

```
1 [4, 5]
```

  Untuk lambda sederhana tidak perlu melakukan definisi parameter, cukup menggunakan kata kunci `it`. Namun pada lambda yang memiliki parameter, harus selalu menyertakan parameter dan isinya. Seperti contoh kode berikut untuk lambda yang menyertakan parameternya :

```
1 fun main(args: Array<String>) {
2   val list = listOf(1, 2, 3, 4, 5)
3   println(list.map { data -> data * 2 } )
4 }
```

  Inti dari kode di atas sebetulnya ada pada baris ke-3, pada bagian `data -> data * 2`, inilah bentuk lambda di Kotlin. Seluruh isi pada variabel `list` akan dikalikan dengan 2 dengan perintah tersebut, sehingga hasil keluarannya adalah seperti berikut :

```
1 [2, 4, 6, 8, 10]
```

- Format *Header* Kelas

  Deklarasi kelas dengan sedikit parameter dapat dituliskan dalam satu baris kode seperti berikut :

```
1 class Mahasiswa(nim: String, nama: String)
```

  Namun deklarasi kelas yang lebih banyak harus tersusun lebih rapih, sehingga setiap parameter atau argumen pembentuk kelas berada pada baris terpisah. Kemudian tutup kurung untuk parameter / argumen harus berada pada baris baru.

Jika menggunakan pewarisan, kontruktor super-kelas harus ditempatkan pada baris yang sama dengan tutup kurung. Berikut adalah contoh kodenya :

```
class Pejabat (
    nip: String,
    nama: String,
    alamat: String,
    hp: String,
    gaji: BigInteger
) : Pegawai(nip, nama) {
    ...
}
```

- Unit

  Bila sebuah fungsi mengembalikan nilai bertipe `Unit`, maka tidak perlu dituliskan, seperti contoh berikut :

```
fun fungsiKu() {
    ...
}
```

  Unit sendiri di Kotlin seperti tipe `void` di Java, yaitu tipe dengan hanya satu nilai saja.

- Fungsi vs. Properti

  Dalam beberapa hal, fungsi tanpa sebuah parameter mungkin terlihat sama dengan properti. Walau secara tertulis mirip, namun ada beberapa pertimbangan gaya untuk memilih salah satunya.

  Jadikanlah itu sebuah properti dan bukan fungsi bila :

  – tidak melamparkan eksepsi
  – kompleksitas rendah
  – perhitungan yang terjadi tidak berat
  – mengembalikan nilai yang sama setelah beberapa kali dipanggil

Demikianlah gaya penulisan kode pada Kotlin, walaupun bila tidak mengikuti aturan ini aplikasi tetap bisa di*compile* dan berjalan sebagaimana mestinya, tetapi ini adalah aturan baku untuk memudahkan sesama pemrogram bahasa Kotlin lebih cepat dan mudah dalam memahami kode yang ditulis oleh orang lain.

BAB 2

DASAR-DASAR

2.1 Tipe Data

Sebagaimana kebanyakan bahasa pemrograman, mengenali tipe data adalah hal yang penting untuk diketaui, karena perbedaan tipe data dapat menyebabkan perbedaan operasi yang dapat dilakukan kepadanya.

Tipe data secara garis besar dapat dikelompokan menjadi : angka, karakter, *boolean*, dan larik.

2.1.1 Angka

2.1.2 Karakter

2.1.3 *Boolean*

2.1.4 Larik

2.2 Paket

2.3 Mengatur Alur

BAB 3

KELAS DAN OBJEK

BAB 4

FUNGSI DAN LAMDA

BAB 5

JAVA INTEROPERABILITAS

BAB 6

PERKAKAS

6.1 Menggunakan Gradle

6.2 Menggunakan Maven

BAB 7

CONTOH KASUS APLIKASI CHAT