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 :

```
Hai, selamat datang
```

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

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 :

```
val a: Int = 2
// atau
val c = 2
// atau
val d: Int
d = 5
```

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

```
var e = 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 :

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

Untuk kode komentar multi-bari adalah sebagai berikut :

```
/*  ini komentar
    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 :

```
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

BAB 2

DASAR-DASAR

2.1 Tipe Data

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