



Crash Course



Dr. Bambang Purnomosidi D. P.

PT Wabi Teknologi Indonesia
2018



Materi ini diproduksi oleh **PT Wabi Teknologi Indonesia** dan siapapun juga diijinkan untuk menggunakan materi ini secara bebas sesuai dengan lisensi *CC BY-NC-ND 4.0 Internasional*:

1. Harus memberikan atribusi ke PT Wabi Teknologi Indonesia
2. Tidak boleh digunakan untuk keperluan komersial.
3. Tidak boleh ada produk derivatif atau turunan

Lisensi lengkap dari CC BY-NC-ND 4.0 International bisa dilihat di <https://creativecommons.org/licenses/by-nc-nd/4.0/>). Untuk penggunaan selain ketentuan tersebut, silahkan menghubungi:

PT Wabi Teknologi Indonesia

Jl. Raya Janti Karang Jambe no 143

Yogyakarta 55198

Indonesia

Phone: 0274 486664 ext 3431

WhatsApp/Telegram (9:00-16:00): +62 813-3593-7700

General inquiries: info@kamiwabi.id

Engineering inquiries: engineering@kamiwabi.id

Training inquiries: education@kamiwabi.id

01. Apa Kotlin Itu?

Kotlin adalah spesifikasi bahasa pemrograman serta peranti kompilator (*compiler tools*) yang dibuat oleh JetBrains (perusahaan berbasis di St. Petersburg - Rusia, pembuat IDE IntelliJ IDEA). Untuk penyebutan selanjutnya, Kotlin akan mengacu pada spesifikasi bahasa pemrograman serta *compiler* yang mengimplementasikan spesifikasi tersebut.

Kotlin termasuk dalam kategori *statically-typed*, yaitu mensyaratkan berbagai konstruksi bahasa pemrograman tersebut untuk mempunyai tipe yang pasti sehingga perubahan tipe saat menjalankan program tidak bisa dilakukan. Kepastian tipe ini juga membuat hasil kompilasi lebih kecil dan lebih cepat karena tidak perlu ada mekanisme untuk pengaturan tipe dinamis.

Kotlin memerlukan JDK, bisa dikompilasi ke *byte code* Java Virtual Machine dan bisa dijalankan di JRE. Selain itu Kotlin juga bisa dikompilasi ke JavaScript. Kompilasi ke *native code* juga tersedia tetapi masih belum stabil dan bukan merupakan versi resmi yang didukung oleh Kotlin.

02. Aplikasi Apa yang Cocok Dikembangkan Menggunakan Kotlin?

Kotlin digunakan untuk berbagai macam aplikasi. Penggunaan utama dari Kotlin adalah sebagai peranti pengembangan untuk Android. Meskipun demikian, Kotlin juga kuat di sisi server / *backend* serta aplikasi-aplikasi infrastruktur (teknologi blockchain Corda dibuat menggunakan Kotlin). Kotlin tidak bisa / tidak sesuai digunakan untuk pemrograman aras rendah (*low-level programming*) atau pemrograman untuk Hardware.

03. Persyaratan Sistem

Kotlin berjalan di atas platform Java (JDK harus tersedia), dengan demikian, semua sistem operasi yang mendukung Java bisa digunakan. Jika mengkompilasi dari *source code*, maka JDK6, JDK7, JDK8, JDK9 harus tersedia (lihat repo Kotlin untuk cara kompilasi: <https://github.com/JetBrains/kotlin>). Materi di *crash course* ini menggunakan *command line compiler* yang bisa diperoleh di <https://github.com/JetBrains/kotlin/releases/latest>. Keluaran dari hasil kompilasi bisa ditargetkan untuk JDK6 maupun JDK8.

04. Mulai Menggunakan Kotlin

Untuk mulai menggunakan Kotlin, pastikan Java telah terinstall dengan baik (harus Java Development Kit, Java Runtime Environment saja tidak cukup. Ketikkan berikut ini untuk memeriksa:

```
» java -version
java version "1.8.0_181"
Java(TM) SE Runtime Environment (build 1.8.0_181-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.181-b13, mixed mode)
```



```
bpdp at archer1 in ~
» javac -version
javac 1.8.0_181
bpdp at archer1 in ~
»
```

Untuk instalasi Kotlin, hanya perlu menggunakan mengekstrak file yang diperoleh dari repo Kotlin dan kemudian mengatur variabel lingkungan (*environemnt variable*). Misal ekstraksi dilakukan di direktori berikut:

```
» pwd
/opt/software/kotlin-dev-tools/kotlin-compiler-1.2.70
bpdp at archer1 in /o/s/k/kotlin-compiler-1.2.70
» ls -la
total 24
drwxr-xr-x 5 bdpd bdpd 4096 Sep 14 16:13 ./
drwxr-xr-x 5 bdpd bdpd 4096 Sep 15 20:03 ../
drwxr-xr-x 3 bdpd bdpd 4096 Sep 14 16:13 bin/
-rw-r--r-- 1 bdpd bdpd  17 Sep 10 17:35 build.txt
drwxr-xr-x 2 bdpd bdpd 4096 Sep 14 16:13 lib/
drwxr-xr-x 3 bdpd bdpd 4096 Sep 14 16:13 license/
bpdp at archer1 in /o/s/k/kotlin-compiler-1.2.70
»
```

Variabel lingkungan yang diatur adalah sebagai berikut:

Jika menggunakan shell **Fish**:

```
set -x PATH $PATH /opt/software/kotlin-dev-tools/kotlinc/bin
```

Jika menggunakan shell **Bash**:

```
export PATH $PATH:/opt/software/kotlin-dev-tools/kotlinc/bin
```

Untuk memeriksa apakah Kotlin telah terinstall:

```
» kotlinc -version
info: kotlinc-jvm 1.2.70 (JRE 1.8.0_144-jdk_2017_08_24_20_46-b00)
»
```



Kotlin menyediakan fasilitas REPL (Read-Eval-Print-Loop) untuk mencoba *source code* pendek:

```
» kotlinc-jvm
Welcome to Kotlin version 1.2.70 (JRE
1.8.0_144-jdk_2017_08_24_20_46-b00)
Type :help for help, :quit for quit
>>> 42*10-23/4
415
>>> 42*10-23.00/4
414.25
>>> println("Selamat belajar Kotlin")
Selamat belajar Kotlin
>>>
»
```

05. Kompilasi Kotlin

Pada awalnya Kotlin dirancang sebagai bahasa yang diimplementasikan di atas JVM sehingga memungkinkan untuk menjangkau berbagai platform serta menggunakan berbagai pustaka-pustaka Java yang sudah dibangun sebelumnya. Secara default, Kotlin akan menghasilkan *bytecode* (.class) yang bisa dijalankan oleh JRE, tetapi perkembangan berikutnya memungkinkan Kotlin untuk menargetkan hasil kompilasi ke JavaScript serta *native code*.

Contoh *source code* Kotlin yang akan diterjemahkan ke dalam berbagai platform:

```
» cat hello.kt
fun main(args : Array<String>) {
    val scope = "world"
    println("Hello, $scope!")
}
»
```

JVM

```
» kotlinc hello.kt
» ls -la
...
-rw-r--r--  1 bdpd bdpd      85 May 18  2017 hello.kt
-rw-r--r--  1 bdpd bdpd    1236 Sep 16  08:06 HelloKt.class
...
```



```
» kotlin HelloKt
Hello, world!
»
```

JavaScript

```
» kotlinc-js hello.kt -output hello.js
» ls -la
...
-rw-r--r--  1 bdpd bdpd    511 Sep 16 08:08 hello.js
-rw-r--r--  1 bdpd bdpd    85 May 18 2017 hello.kt
```

Untuk menjalankan hasil, diperlukan file **kotlin.js** yang bisa diperoleh menggunakan *npm* dari Node.js (bisa diperoleh di <https://nodejs.org>):

```
» npm install kotlin
```

Setelah itu salin file di *node_modules/kotlin/kotlin.js* ke direktori apa saja kemudian buat file HTML (hello.html):

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Console Output</title>
  </head>
  <body>

    <script type="text/javascript" src="lib/kotlin.js"></script>
    <script type="text/javascript" src="hello.js"></script>
  </body>
</html>
```

Panggil file HTML tersebut melalui browser (file:///lokasi/ke/file/hello.html). Browser tidak akan menampilkan apapun tetapi menampilkan tulisan di *console log* seperti pada *Developer Tools* di browser *Chromium* berikut:





Native Code

Untuk mengkompilasi ke *native code* dari sistem operasi, diperlukan *compiler* khusus yang bisa diperoleh di <https://github.com/JetBrains/kotlin-native/releases>. Instalasi hanya memerlukan ekstraksi file serta konfigurasi variabel lingkungan (PATH). Saat mengkompilasi, Kotlin akan mengambil *dependencies* LLVM, sysroot, dan lain-lain. Berikut ini adalah gambaran dari proses:

```
» kotlinc-native hello.kt
Downloading native dependencies (LLVM, sysroot etc). This is a
one-time action performed only on the first run of the compiler.
Downloading dependency:
https://download.jetbrains.com/kotlin/native/clang-llvm-6.0.1-linux-x8
6-64.tar.gz (509.0 MiB/509.0 MiB). Done.
Extracting dependency:
/home/bpdp/.konan/cache/clang-llvm-6.0.1-linux-x86-64.tar.gz into
/home/bpdp/.konan/dependencies
Downloading dependency:
https://download.jetbrains.com/kotlin/native/target-gcc-toolchain-3-li
nux-x86-64.tar.gz (58.4 MiB/58.4 MiB). Done.
Extracting dependency:
/home/bpdp/.konan/cache/target-gcc-toolchain-3-linux-x86-64.tar.gz
into /home/bpdp/.konan/dependencies
Downloading dependency:
https://download.jetbrains.com/kotlin/native/libffi-3.2.1-2-linux-x86-
64.tar.gz (55.1 kiB/55.1 kiB). Done.
Extracting dependency:
/home/bpdp/.konan/cache/libffi-3.2.1-2-linux-x86-64.tar.gz into
/home/bpdp/.konan/dependencies
bpdp at archer1 in ~/k/s/kotlin
»
» ls -la
```



```
...
-rw-r--r-- 1 bdp bdp 85 May 18 2017 hello.kt
-rwxr-xr-x 1 bdp bdp 456056 Sep 15 20:05 program.kexe*
» file program.kexe
program.kexe: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for
GNU/Linux 2.6.16,
BuildID[sha1]=814622b994377cc5764884f5686292550a54c7b4, not stripped
» ./program.kexe
Hello, world!
»
```

06. Kotlin dan Gradle

Saat aplikasi yang sudah kita buat semakin kompleks, maka biasanya kita akan memerlukan berbagai pustaka dan dengan kondisi kompilasi yang mungkin berbeda-beda (misalnya untuk test, menghasilkan dokumentasi, dan lain-lain). Untuk kasus seperti ini, digunakan *build tool*. Di dunia Java, *build tool* yang sampai saat ini banyak digunakan adalah Gradle (<https://gradle.org>). Selain itu, untuk Kotlin bisa juga menggunakan Apache Ant, Apache Maven, Bazel, atau yang dibuat khusus untuk Kotlin seperti Kobalt. Untuk memulai proyek menggunakan Gradle:

```
» gradle init
Starting a Gradle Daemon (subsequent builds will be faster)

BUILD SUCCESSFUL in 18s
2 actionable tasks: 2 executed
» ls -la
total 36
drwxr-xr-x 4 bdp bdp 4096 Sep 16 08:49 ./
drwxr-xr-x 4 bdp bdp 4096 Sep 16 08:48 ../
-rw-r--r-- 1 bdp bdp 201 Sep 16 08:49 build.gradle
drwxr-xr-x 4 bdp bdp 4096 Sep 16 08:49 .gradle/
drwxr-xr-x 3 bdp bdp 4096 Sep 16 08:49 gradle/
-rwxr-xr-x 1 bdp bdp 5296 Sep 16 08:49 gradlew*
-rw-r--r-- 1 bdp bdp 2260 Sep 16 08:49 gradlew.bat
-rw-r--r-- 1 bdp bdp 356 Sep 16 08:49 settings.gradle
»
```

Proses *build* akan memerlukan beberapa file. File utama yang diperlukan adalah **build.gradle**. Untuk kompilasi dan menjalankan *source code* Kotlin, isikan berikut ini pada file build.gradle:



```
» cat build.gradle
buildscript {
    ext.kotlin_version = '1.2.70'

    repositories {
        mavenCentral()
    }

    dependencies {
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
    }
}

plugins {
    id "org.jetbrains.kotlin.jvm" version "1.2.70"
}

apply plugin: 'application'

repositories {
    mavenCentral()
}

dependencies {
    compile "org.jetbrains.kotlin:kotlin-stdlib"
    compile "org.jetbrains.kotlin:kotlin-stdlib-jdk8"
}

compileKotlin {
    kotlinOptions.suppressWarnings = true
}

mainClassName = 'HelloKt'
»
```

Untuk mengkompilasi, gunakan task *build*, untuk menjalankan, gunakan task *run*.

```
» gradle build
```

```
BUILD SUCCESSFUL in 15s
6 actionable tasks: 6 executed
» gradle run
```

```
> Task :run
Hello, world!
```

```
BUILD SUCCESSFUL in 1s
2 actionable tasks: 1 executed, 1 up-to-date
»
```



07. Sintaksis Dasar Kotlin

Pada bab ini, akan dibahas beberapa sintaksis dasar dari Kotlin.

Tipe

Data yang tersimpan dan digunakan di Kotlin mempunyai tipe. Kotlin merupakan bahasa pemrograman yang bersifat *statically-typed*, artinya beberapa konstrukti bahasa yang terkait dengan data harus mempunyai tipe tertentu dan tidak bisa diganti tipenya. Kotlin mendukung beberapa tipe berikut untuk angka:

1. Double: 64 bit ($\pm 1.79769313486231570E+308$)
2. Float : 32 bit ($\pm 3.40282347E+38F$)
3. Long : 64 bit (9,223,372,036,854,775,808 sampai 9,223,372,036,854,775,807)
4. Int : 32 bit (-2,147,483,648 sampai 2,147,483, 647)
5. Short : 16 bit (-32,768 sampai 32,767)
6. Byte : 8 bit (-128 sampai 127)

Selain itu ada juga tipe:

1. String : Diapit “ “
2. Char : Diapit tanda petik tunggal, bisa berisi escape character ('\\...') atau bisa juga menggunakan 'u...' untuk karakter *unicode*.
3. Boolean : true / false

Variabel

Variabel merupakan suatu nama dari lokasi *memory* komputer yang digunakan untuk menyimpan suatu data. Data ini disimpan untuk keperluan pengolahan. Kotlin mempunyai 2 jenis deklarasi variabel:

1. Deklarasi menggunakan **val** (immutable)
2. Deklarasi menggunakan **var** (mutable)

```
fun main(args : Array<String>) {
```

```
    val immu1 = 10
    val immu2 = "Wabi"
    val immu3: Int = 200
    val immu4: Int
```



```

    immu4 = 250

    // what if we change the value?
    // immu4 = 100

    println(immu1)
    println(immu2)
    println(immu3)
    println(immu4)

    var mu1 = 10
    var mu2: String = "Angka 200"
    var mu3: Int = 200
    var mu4: Int

    mu4 = 250

    println(mu1)
    println(mu2)
    println(mu3)
    println(mu4)

    // what if we change the value?
    mu4 = 300

    println(mu4)
    // what if we change the type?
    // mu4 = "Angka 300"
}

```

Konstanta

Konstanta merupakan penetapan nilai yang tidak bisa diubah. Berbeda dengan *immutable* **val**, konstanta tidak bisa dideklarasikan di level *local* dan hanya digunakan untuk nilai yang diketahui pada saat *compile time*.

```

const val startDay: String = "Monday"

fun main(args : Array<String>) {

```



```
// what if we put const decalaration here:
// const val startDay: String = "Monday"

println("Hello, $startDay!")

}
```

Komentar

Komentar digunakan untuk menandai bagian dari *source code* yang tidak dikompilasi dan / atau dijalankan. Ada 2 cara untuk memberikan komentar:

```
// komentar 1 baris

/*
    Komentar lebih
    dari 1 baris
*/
```

Operator

Operator digunakan untuk melakukan operasi terhadap nilai-nilai (*operand*). Kotlin mempunyai banyak operator, beberapa yang umum digunakan antara lain:

1. +, -, *, /, % untuk operasi matematika
2. = untuk operator penugasan
3. +=, -=, *=, /=, %= untuk operator penugasan sesuai dengan operator matematika di depan.
4. ++, -- untuk operator penambahan dan pengurangan.
5. &&, ||, ! untuk operator logika **and**, **or**, dan **not**
6. ==, != operator untuk memeriksa kesamaan
7. ===, !== untuk memeriksa kesamaan, sama jika mereferensikan pada obyek yang sama (*referential equality* operator).
8. <, >, <=, >= operator perbandingan nilai.

Daftar lengkap dari operator bisa dilihat di <https://kotlinlang.org/docs/reference/keyword-reference.html>.

Pengendali Alur Program

If ... else if ... else



```
fun main(args: Array<String>) {

    var angka = 2
    val res = if (angka > 0)
        "positif"
    else if (angka < 0)
        "negatif"
    else
        "nol"
    println("angka $angka adalah angka $res")

    angka = -2
    var res1: String
    if (angka > 0)
        res1 = "positif"
    else if (angka < 0)
        res1 = "negatif"
    else
        res1 = "nol"
    println("angka $angka adalah angka $res1")

}
```

ranges

Untuk mengakses komponen dalam jangkauan tertentu, gunakan **in** dan titik 2 kali.

```
fun main(args : Array<String>) {

    val a = 15
    val b = 20
    if (a in 1..b) {
        println("angka $a ada di range 1 sampai $b")
    }

}
```

when

when digunakan untuk mengevaluasi suatu nilai tunggal.



```
fun main(args : Array<String>) {

    val a = 200

    when (a) {
        15 -> println("a = 15")
        in 20..30 -> println("berada dalam range 20 - 30")
        16, 18 -> println("16 atau 18")
        !in 100..1000 -> println("tidak berada di antara 100 - 1000")
        else -> {
            println("Tidak masuk semua")
            println("Ini menggunakan lebih satu statement, jadi harus dengan
block")
        }
    }
}
```

Loop for

Ada beberapa penggunaan loop **for**:

```
fun main(args: Array<String>) {

    println("First")

    val listOfItems = listOf(1, "two", 3, "four")

    for (a in listOfItems) {
        println(a)
    }

    println("Second")

    for (b in 1..10) {
        println(b)
    }

    println("Third")

    for (c in 10 downTo 0 step 2) {
        println(c)
    }
}
```



```

    }

    println("Fourth")

    loop@ for (d in 10 downTo 0) {
        println(d)
        if ((d % 2) == 0)
            break@loop
    }

    println("Fifth")

    for (e in 10 downTo 0) {
        if ((e % 2) == 0)
            continue
        println(e)
    }
}

```

Loop while

Kotlin menyediakan sintaksis **while** serta **do ... while**

```

fun main(args: Array<String>) {

    var x: Int = 10

    while (x > 0) {
        println(x)
        x--
    }

    var y: Int = 20

    do {
        println(y)
        y--
    } while (y > 0)

}

```



Function

Function di Kotlin didefinisikan menggunakan kata kunci **fun**. Berikut ini adalah contoh deklarasi function:

```
fun main(args : Array<String>) {  
  
    fun kaliEmpat(x: Int): Int {  
        return 4 * x  
    }  
  
    println(kaliEmpat(20))  
  
}
```

Package dan Import

Seperti halnya Java, Kotlin menyediakan **package** untuk mengatur *source code* ke dalam berbagai paket untuk menghindari *name collision* atau tabrakan nama. Oleh karena itu, nama *package* biasanya menggunakan nama domain supaya terhindar dari kemungkinan *name* yang sama. Pada contoh program di bawah ini, terdapat 2 (dua) file:

1. myLib.kt
2. package.kt

File **myLib.kt** berisi function dan akan digunakan di file utama (**package.kt**).

Isi file myLib.kt

```
package id.kamiwabi.lib  
  
fun kaliEmpat(x: Int): Int {  
    return 4 * x  
}
```

Isi file package.kt:

```
import id.kamiwabi.lib.kaliEmpat  
  
fun main(args : Array<String>) {  
  
    println(kaliEmpat(20))  
  
}
```




```
}
```

Proses kompilasi:

```
kotlinc myLib.kt
kotlinc -cp . package.kt
```

Hasil:

```
» tree id
id
├── kamiwabi
│   └── lib
│       └── MyLibKt.class
```

2 directories, 1 file

```
» ls -la
total 28
drwxr-xr-x  4 bdpd bdpd 4096 Sep 16 21:39 ./
drwxr-xr-x 13 bdpd bdpd 4096 Sep 16 18:57 ../
drwxr-xr-x  3 bdpd bdpd 4096 Sep 16 21:38 id/
drwxr-xr-x  2 bdpd bdpd 4096 Sep 16 21:19 META-INF/
-rw-r--r--  1 bdpd bdpd   71 Sep 16 21:38 myLib.kt
-rw-r--r--  1 bdpd bdpd   96 Sep 16 21:37 package.kt
-rw-r--r--  1 bdpd bdpd  992 Sep 16 21:39 PackageKt.class
»
```

Setelah itu, untuk menjalankan:

```
» kotlin PackageKt
80
»
```

Untuk penggunaan pustaka Java / Kotlin yang ada di file .jar, sebaiknya gunakan Gradle untuk mengelola proyek karena masalah *dependencies*.

Penanganan Terhadap Eksepsi

Secara umum, Kotlin menyediakan fasilitas untuk menangani kondisi jika terjadi sesuatu hal di luar alur semestinya (sering disebut sebagai *exception*).



```
fun main(args: Array<String>) {

    try {
        val v:String = "PT Wabi Teknologi Indonesia";
        v.toInt();
    } catch(e:Exception) {
        e.printStackTrace();
    } finally {
        println("An exception happened");
    }

}
```

08. Object-Oriented Programming

Tentang OOP

OOP merupakan paradigma pemrograman yang meniru pola pikir manusia. Dalam OOP, kemampuan mengabstraksi merupakan kemampuan yang sangat penting karena programmer harus membuat *blueprint* dari berbagai macam obyek yang ada dan kemudian mengimplementasikan *blueprint* tersebut ke dalam suatu kelas. Sebagai contoh, untuk membuat kelas Mobil, seorang programmer harus melihat sedemikian banyak mobil kemudian mengabstraksi mobil menjadi ciri-ciri serta perilaku yang sama dan kemudian memasukkan ciri-ciri serta perilaku tersebut ke dalam suatu kelas.

Implementasi OOP di Kotlin

OOP diimplementasikan di Kotlin menggunakan **class** dan kemudian membuat *instance* dari kelas tersebut.

```
class Company {

    private var name: String = "Wabi"

    fun printName() {
        println(name)
    }

    fun getName(): String {
        return name
    }
}
```



```

    }
}

fun main(args: Array<String>) {
    val myCompany = Company()
    myCompany.printName()
    var myCompanyName = myCompany.getName()
    println(myCompanyName)
}

```

```

fun main(args: Array<String>) {

    val myCar = Car("First Car", 4)

    println("Car name = ${myCar.brand}")
    println("Seats = ${myCar.seats}")

}

// constructor
class Car(val brand: String, var seats: Int) {
}

```

Constructor

Constructor merupakan bagian yang digunakan untuk menginisialisasi variabel yang diperlukan dalam suatu *class* serta mengerjakan logika saat obyek diinisialisasi. Constructor di Kotlin dilakukan dengan menempatkan parameter dari *class* secara langsung di deklarasi *class* serta menggunakan **init** untuk logika pemrograman saat obyek diinisialisasi.

```

fun main(args: Array<String>) {

    val myCar = Car("First Car", 4)

    println("Car name = ${myCar.brand}")
    println("Seats = ${myCar.seats}")
    println("Type = " + myCar.carType)

}

```



```

class Car(val brand: String, var seats: Int) {

    var carType: String

    init {

        if (seats == 2)
            carType = "Sport"
        else if (seats > 4)
            carType = "Kendaraan umum"
        else
            carType = "Unknown car"

    }

}

```

Interface

Interface digunakan untuk mendeklarasikan *abstract class*.

```

interface CarInterface {

    val seats: Int
        get() = 4

    // dengan implementasi
    fun printSeats() {
        println(seats)
    }

}

class CarImp : CarInterface {
    override val seats: Int = 2
}

fun main(args : Array<String>) {

    var myCar = CarImp()
}

```



```
println(myCar.seats)

}
```

Inheritance

Inheritance pada OOP digunakan untuk pewarisan *class* ke *subclass*. Untuk keperluan ini, class induk harus diberi kata kunci **open** sebagai penanda bahwa *class* tersebut masih bisa di-subclass. Default dari setiap pembuatan *class* adalah *final*, artinya tidak bisa di-subclass.

```
// penggunaan "open" supaya masih bisa dilakukan inheritance
open class Car(seats: Int, brand: String) {
    init {
        println(brand)
        println("Number of seats: " + seats)
    }
}

class Truck(wheels: Int, seats: Int, brand: String): Car(seats, brand)
{

    init {
        println(brand)
        println(seats)
        println(wheels)
    }

}

fun main(args: Array<String>) {
    val truck1 = Truck(6, 2, "Isuzu")
    println(truck1)
}
```

09. Generics

Generics merupakan salah satu *programming style* dari *generic programming* yang dipelopori oleh ML sekitar tahun 1973. Generics digunakan pada *statically-typed* programming language untuk memungkinkan adanya konstruksi bahasa pemrograman (dalam konteks Kotlin adalah konstruksi yang terkait dengan **class** serta **fun**) yang “menunda” pendefinisian tipe



sampai saat digunakan. Hal ini disebabkan karena saat programmer ingin membuat **class** atau **fun** mempunyai tipe yang fleksibel sementara sebagai bahasa pemrograman yang *statically-typed*, Kotlin harus menetapkan tipe saat kompilasi. Untuk menetapkan parameter sebagai *generics*, Kotlin menggunakan `<>`.

```
fun main(args: Array<String>) {

    var aString = GenClass("Ini menggunakan parameter String")
    var bInt: GenClass<Int> = GenClass(3431)

    println(aString.theArg)
    println(bInt.theArg)

}

class GenClass<T>(arg: T) {

    var theArg = arg

}
```

10. Functional Programming

Functional Programming (FP) merupakan paradigma pemrograman yang menjadikan fungsi (*function*) sebagai *first class citizens* yaitu dapat disimpan ke dalam variabel dan struktur data, digunakan sebagai argumen, serta di-*return* sebagai hasil dari *higher-order function*. FP juga memperlakukan fungsi sebagai ekspresi matematis. FP juga mendorong *immutability* serta konstruksi lain terkait *function* (*anonymous function*, *lambda*, dan lain-lain).

Lambda Expressions

Lambda Expression adalah *function literal* yang tidak dideklarasikan tetapi dilewatkan secara langsung sebagai suatu ekspresi. Kegunaan utamanya untuk fungsi yang sifatnya singkat dan padat.

```
fun main(args: Array<String>) {

    var lambdaOne :(String)->Unit = {s:String -> println(s)}
```



```
lambdaOne("Argumen 1")

val sum = { x: Int, y: Int -> x + y }

println(sum(2,3))

val tanpaArg : () -> Unit = { println("Kosong")}
tanpaArg()

}
```

Higher Order Function

HOF merupakan fungsi yang setidaknya menggunakan fungsi lain sebagai argumen atau menghasilkan fungsi sebagai keluaran fungsi.

```
fun main(args: Array<String>) {

    var printIt: (String) -> Unit = { println(it) }
    hof("para peserta training Kotlin", printIt)

}

fun hof(str: String, exp: (String) -> Unit) {
    print("Selamat datang di Wabi ")
    exp(str)
}
```

11. Struktur Data

Array

Array sering juga disebut sebagai variabel berindeks. Array biasanya digunakan saat programmer mendefinisikan data untuk satu entitas dan komponennya lebih dari satu.

```
fun main(args : Array<String>) {
```



```

// otomatis Any jika tidak ada deklarasi tipe
val kabupaten = arrayOf("Sleman", "Kotamadya", "Kulon Progo", "Gunung
Kidul", "Bantul")

println(kabupaten[1])

// Jika yakin, sertakan type
var a = arrayOf<String>("nol", "satu", "dua")

println(a[2])
println("ambil isi array = " + a.get(2))

// campuran
var b = arrayOf<Any>("nol", 1, 2, "tiga", true)

println(b[4])
println("Jumlah array b = " + b.size)
for (i in b) {
    println(i)
}
println("Ganti isi array")
b.set(1, "satu")
println(b[1])

val firstMatrix = arrayOf(intArrayOf(2, 3, 4), intArrayOf(5, 2, 3))
val secondMatrix = arrayOf(intArrayOf(-4, 5, 3), intArrayOf(5, 6,
3))

println(firstMatrix[0][0])
println(secondMatrix[0][0])

}

```

Collections: List, Set, Map

Collections di Kotlin membedakan antara *immutable* (read only, tidak bisa diubah) dengan *mutable* (bisa dimanipulasi).

List berisi daftar elemen, mirip dengan array tetapi mempunyai jumlah elemen yang lebih fleksibel.

```

fun main(args : Array<String>) {

```




```

val buah = mutableListOf("Jeruk", "Alpukat", "Mangga", "Edamame")

buah.add("Nangka")
for (a in buah) {
    println(a)
}

val buah2 = listOf("Manggis", "Durian", "Manggis")

// try this:
//buah2.add("Duku")
for (i in buah2) {
    println(i)
}

}

```

Set berisi daftar elemen, tetapi tidak boleh ada yang ganda.

```

fun main(args : Array<String>) {

    val buah = mutableSetOf("Jeruk", "Alpukat", "Mangga", "Edamame")

    buah.add("Jeruk")
    for (a in buah) {
        println(a)
    }

    println("=====")

    buah.add("Pisang")
    for (x in buah) {
        println(x)
    }

}

```

Map mirip seperti *dictionary* (kamus):

```

fun main(args : Array<String>) {

```



```

    val kodePos = mutableMapOf("55198" to "Bantul", "55571" to "Griya
Purwa Asri")

    println(kodePos["55198"])

    kodePos["55198"] = "Bantul Kota"

    println(kodePos["55198"])
    kodePos["55572"] = "Wedomartani"
    println(kodePos["55572"])
    kodePos.remove("55572")

}

```

12. Data Class

Data class merupakan class yang digunakan untuk menyimpan dan mengelola data. Jika programmer membuat data class, secara otomatis Kotlin akan membuat berbagai member yang berguna bagi data:

1. equals()/hashCode()
2. toString()
3. componentN()
4. copy()

```

data class Pasien(val nama: String, val alamat: String, var usia: Int)

```

```

fun main(args : Array<String>) {

```

```

    val pasien1: Pasien = Pasien("Bp. Pasien Satu", "Alamat pasien 1",
38)

```

```

    println(pasien1.nama)
    println(pasien1.alamat)
    println(pasien1.usia)
    pasien1.usia = 37
    println(pasien1.usia)
    println("=====")
    print(pasien1.toString())

```

```

}

```



