Java Developer Fundamentals

Training Preparation

Java Development Prerequisites

- Windows OS or Mac OS Laptop
- Java Development Kit - Java > 7
- Android Studio (Current 3.3.1)
- Android Phone & Usb Cable (Data Cable)
- SDK Manager API Level > 23

Install Java SDK

1.1. Memasang Java Development Kit

1. Buka jendela terminal di komputer.

2. Ketikkan java -version

Jika sudah terinstall maka akan muncul output

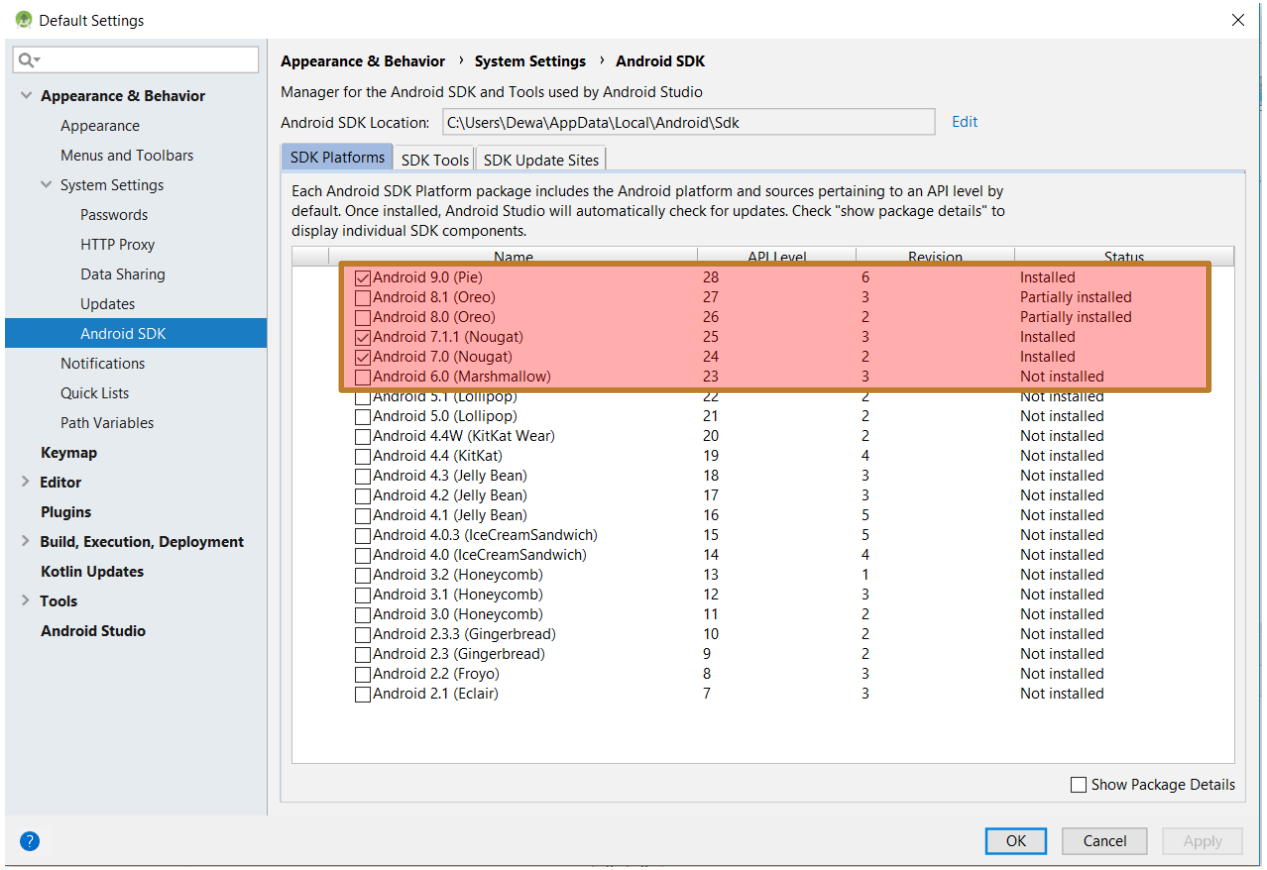
Java(™) SE Runtime Environment (build1.X.0_05-b13)

Untuk mengunduh Java Standard Edition () Development Kit (JDK):

https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

SDK Manager Config

- Pada Android Studio pilih menu tab Tools → SDK Manager
- Pilih Menu Android SDK, dan pastikan pada Tab SDK Platform sudah tercentang SDK API Level sesuai dengan handphone Android yang dimiliki
- Lalu Apply, maka Android Studio akan menginstall komponen-komponen yang dibutuhkan



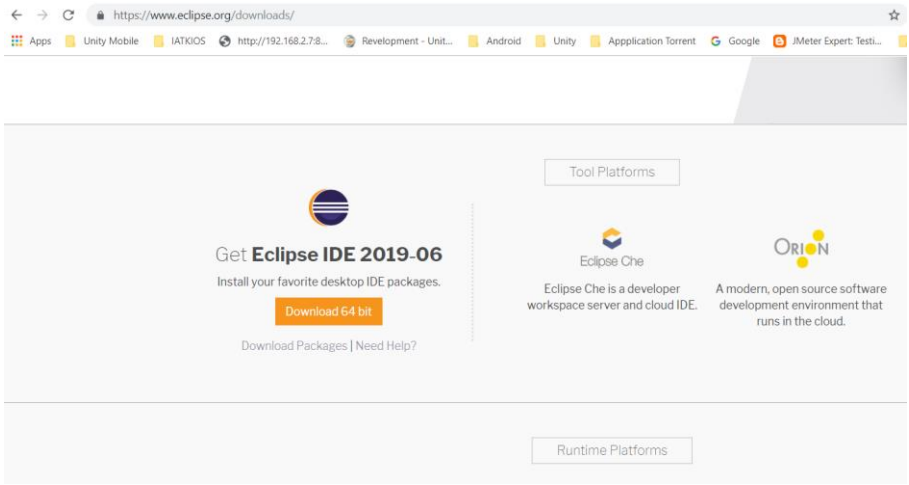
Install Eclipse Studio

1. Menjalankan Eclipse

Setelah tahapan instalasi JDK selesai, sekarang saatnya untuk menjalankan aplikasi Eclipse. Bagi yang belum download, bisa mengunjungi situs resmi Eclipse disini: https://www.eclipse.org/downloads/

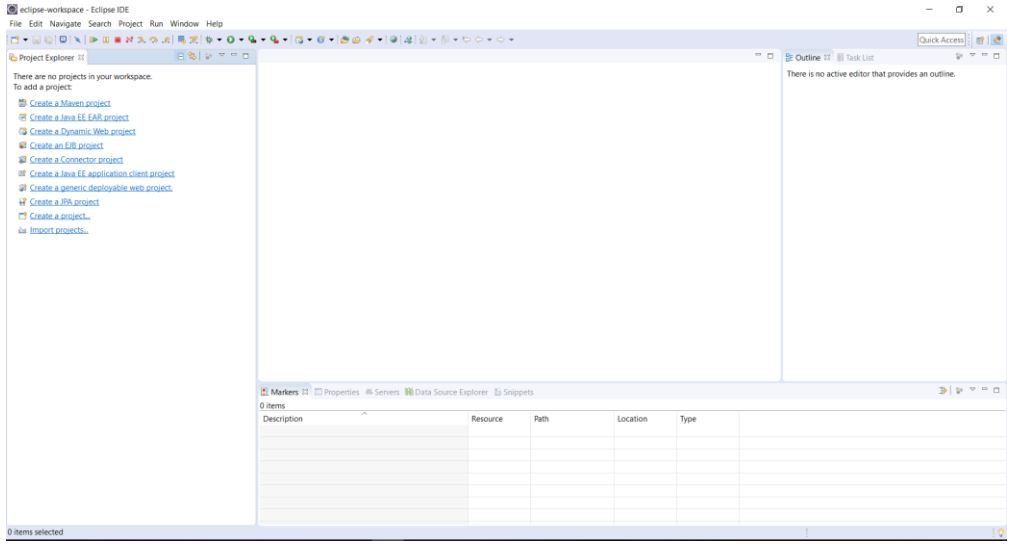
2. Pilih yang sesuai dengan Sistem Operasi dan Arsitektur prosesor Anda. Pada tutorial ini saya menggunakan Windows 10 64 bit dan Eclipse 2019-06 Enterprise (53 MB).

Setelah anda download (Bagi yang belum punya) ekstrak file tersebut kesebuah folder terserah anda dengan aplikasi compreser misalkan WinRar. Lalu jalankan dengan membuka file Eclipse.exe



Install Eclipse Studio

Pada saat pertama kali menjalankan aplikasi Eclipse, maka anda akan melihat tampilan seperti gambar dibawah ini. Pilih lokasi untuk menyimpan workspace dan beri tanda centang pada “Use this as the default and do not ask again” agar tampilan tersebut tidak muncul lagi.



Terdapat beberapa menu di halaman awal ini yaitu:

- ✓Overview yang berisi sekilas tentang fitur-fitur pada Eclipse
- ✓Tutorials yang berisi petunjuk cara menggunakan Eclipse
- ✓Samples yang berisi contoh source code program yang sudah jadi.
- ✓What’s New yang isinya apa saja fitur baru pada versi ini.

Setelah ini anda dapat langsung menggunakan Eclipse dan membuat program-program Java seperti bias

Preparation Completed

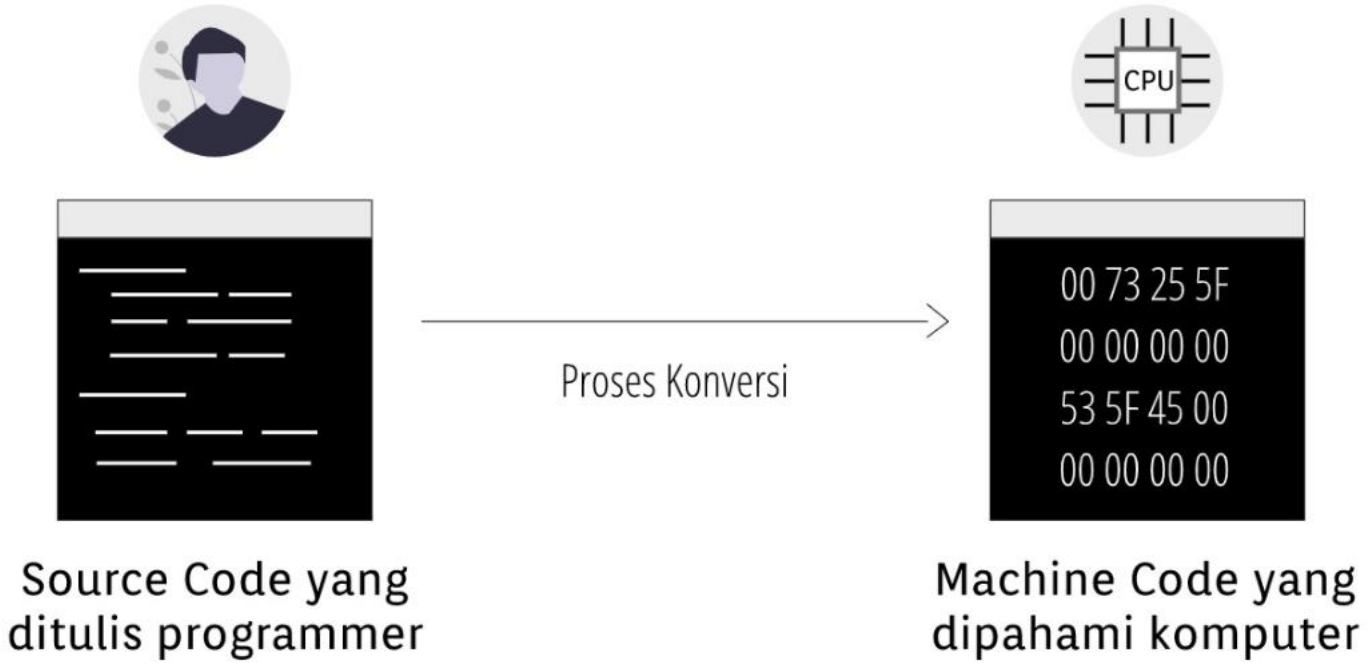
See you in next Session

Perbedaan Compiler dan Interpreter

Perbedaan Compiler dan Interpreter – Berkomunikasi dengan komputer tidak dapat menggunakan bahasa manusia, melainkan menggunakan bahasa pemrograman. Instruksi yang kita berikan akan diproses oleh *CPU (Central Processing Unit)* yang ada pada komputer. Sebenarnya bahasa pemrograman atau *source code* akan melalui proses sebelum bisa dipahami oleh komputer. Karena pada dasarnya komputer hanya memahami bahasa mesin.

Bahasa mesin yang dimengerti oleh komputer identik dengan angka 0 dan 1 serta tidak bisa dipahami oleh manusia. Bisa saja Anda berusaha untuk belajar bahasa mesin tapi sangat tidak menyarankan hal tersebut karena akan menghabiskan banyak waktu. Bahasa mesin akan berbeda-beda tergantung pada prosesor yang digunakan. Bisa dibayangkan seandainya kita berhasil memahami bahasa mesin di prosesor A kemudian beralih ke prosesor B atau bahkan versi yang lain dari Prosesor A maka bahasanya akan berbeda.

Perbedaan Compiler dan Interpreter



Lalu bagaimana bisa kode yang kita buat bisa dengan berbagai bahasa pemrograman bisa dipahami oleh komputer ?

Perbedaan Compiler dan Interpreter

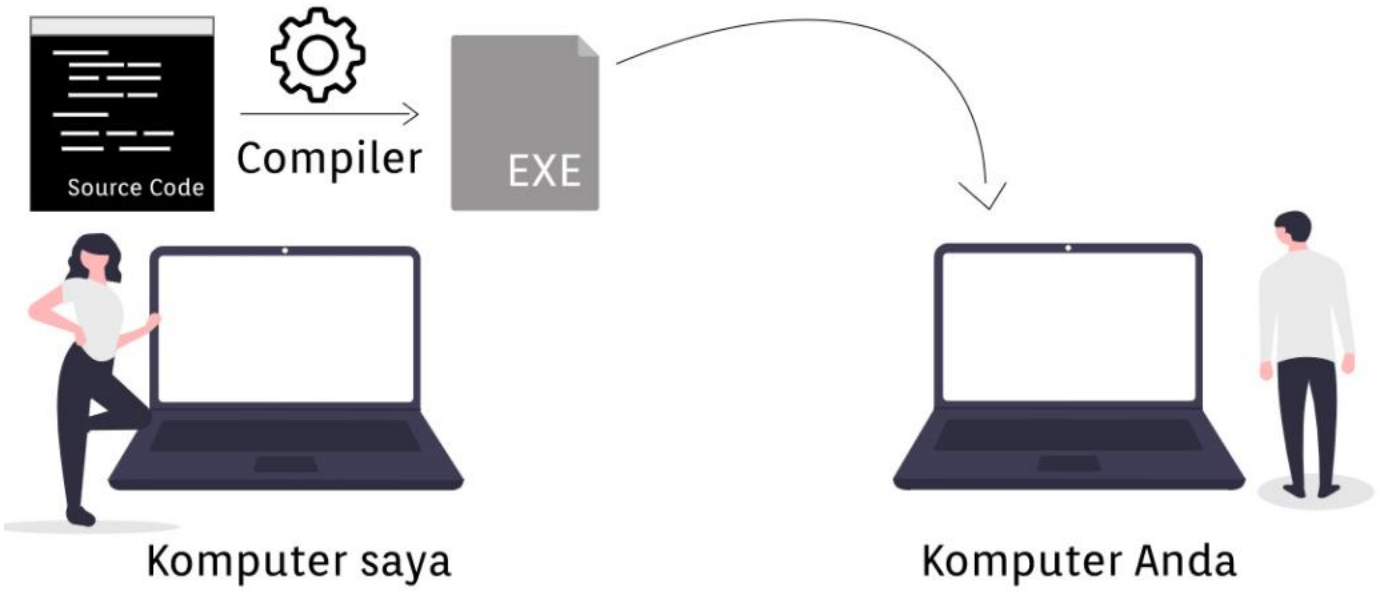
Ketika seorang *programmer* menuliskan kode program atau *source code* kemudian menjalankannya, maka akan terjadi proses konversi. Proses konversi tersebut dibedakan menjadi dua yaitu Compile dan Interpret. Untuk mengubah *source code* menjadi kode mesin kita memerlukan *compiler* dan *interpreter*. Kita akan membahas perbedaan *compiler* dan *interpreter* di bawah ini.

Pengertian Compiler

Bayangkan Anda membuat sebuah program sederhana di komputer milik Anda sendiri. Kemudian Anda juga ingin menjalankan program tersebut di komputer teman Anda. Metode pertama kita menggunakan compile.

Jika kita membahas tentang metode compile, maka kita akan berteman dengan *compiler*. *Compiler* adalah sebuah program yang bertugas untuk mengonversi *source code* yang kita buat menjadi bahasa mesin. Apabila terdapat penulisan kode yang salah, maka compiler akan mengirimkan pesan eror kepada kita dan harus diperbaiki. Jika tidak maka akan menghasilkan berkas *executable*, contohnya seperti .exe.

Perbedaan Compiler dan Interpreter



Setelah proses *compile* di komputer Anda selesai, kemudian Anda memberikan hasilnya berupa berkas .exe kepada teman Anda. Apa yang terjadi di komputer teman Anda? Program dapat dijalankan di komputer teman Anda tetapi ia tidak mengetahui *source code* atau bahasa pemrograman apa yang Anda gunakan. Sehingga *source code*-nya masih berada di komputer Anda.

Perbedaan Compiler dan Interpreter

Arsitektur compiler modern biasanya bukan lagi merupakan program tunggal namun merupakan rangkaian komunikasi antar program dengan tugas spesifik masing-masing. Program-program tersebut beserta tugasnya secara umum terdiri dari:

- **Compiler** itu sendiri, yang menerima kode sumber dan menghasilkan bahasa tingkat rendah *(assembly)*.
- **Assembler**, yang menerima keluaran *compiler* dan menghasilkan berkas objek dalam bahasa mesin.
- **Linker**, yang menerima berkas objek keluaran *assembler* untuk kemudian digabungkan dengan pustaka-pustaka yang diperlukan dan menghasilkan program yang dapat dieksekusi *(executable)*.

Compiler yang menggunakan arsitektur ini misalnya GCC, Clang dan FreeBASIC.

Beberapa *compiler* tidak menggunakan arsitektur di atas secara gamblang, dikarenakan komunikasi antar program jauh lebih lambat dibandingkan jika komunikasi dilakukan secara internal di dalam satu program. Sehingga *compiler* tersebut mengintegrasikan *assembler* dan *linker* di dalam *compiler*. *Compiler* yang menggunakan arsitektur ini salah satunya adalah Free Pascal.

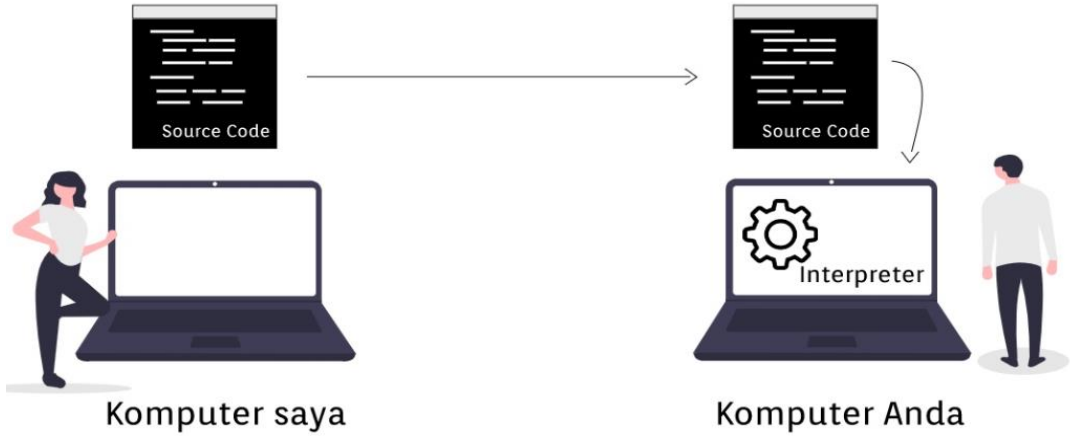
Perbedaan Compiler dan Interpreter

Pengertian Interpreter

Dalam ilmu komputer, *interpreter* (penerjemah) adalah perangkat lunak yang berfungsi melakukan eksekusi sejumlah instruksi yang ditulis dalam suatu bahasa pemrograman tanpa terlebih dahulu menyusunnya menjadi program bahasa mesin. Interpreter umumnya menggunakan salah satu strategi berikut untuk menjalankan program:

- Mengeksekusi kode sumber secara langsung.
- Menerjemahkannya ke dalam serangkaian *portable-code* atau *precompiled-code* kemudian mengeksekusinya.
- Mengeksekusi kode yang telah dikompilasi sebelumnya oleh *compiler* yang merupakan bagian dari sistem penerjemahan.

Perbedaan Compiler dan Interpreter



Pada dasarnya metode *interpret* sama dengan *compile* yaitu mengonversi bahasa pemrograman supaya bisa dipahami oleh mesin dengan bantuan interpreter. Perbedaannya adalah ketika kita menggunakan *compiler*, kode sumber akan dikonversi menjadi *machine code* (membuat berkas *executable*) sebelum program tersebut dijalankan.
Sedangkan *interpreter* mengonversi *source code* menjadi machine code secara langsung ketika program dijalankan.

Salah satu bahasa pemrograman yang dapat diinterpretasikan adalah JavaScript. Bayangkan Anda membuat program sederhana menggunakan JavaScript. Kemudian Anda membagi *source code* tersebut ke teman Anda. Untuk menjalankan JavaScript tersebut, teman Anda setidaknya bisa menggunakan web browser untuk menjalankannya. Web browser sudah terdapat interpreter di dalamnya sehingga berkas JavaScript tersebut bisa diinterpretasikan secara langsung

Perbedaan Compiler dan Interpreter

Masih bingung antara perbedaan *compiler* dan *interpreter*? Simak perbedaannya pada tabel di bawah ini.

Kategori	Compiler	Interpreter
Penggunaan	Source code telah dikonversi menjadi machine code. Sehingga waktu eksekusi program akan lebih singkat.	Lebih mudah digunakan untuk pemula yang baru belajar.
Hasil keluaran	Menghasilkan program luaran atau berkas executable. Contohnya seperti .exe yang dapat dijalankan secara independen	Tidak menghasilkan program luaran atau berkas executable. Jika ingin menjalankan program, maka harus melibatkan source code secara langsung selama proses eksekusi.
Efektifitas	Hasil kompilasi dari source code akan berjalan lebih cepat.	Berjalan lebih lambat ketika dieksekusi.
Platform	Spesifik ke platform tertentu, misal hasil kompilasi berupa berkas.exe tidak dapat dijalankan di Mac. Begitu pula sebaliknya.	Cross platform. Bisa dijalankan di banyak platform asalkan memiliki interpreter yang sesuai.
Bahasa Pemrograman	Bahasa pemrograman yang memerlukan compiler seperti C, C++, C#, Swift, Java	Bahasa pemrograman yang memerlukan interpreter seperti JavaScript, Python, PHP, Ruby.

Java Virtual Machine

Java Virtual Machine (JVM) adalah mesin yang menyediakan lingkungan *runtime* untuk menjalankan kode dalam aplikasi Java.

Mesin ini mengubah *bytecode* Java menjadi bahasa mesin. JVM sendiri merupakan bagian dari *Java Run Environment* (JRE).

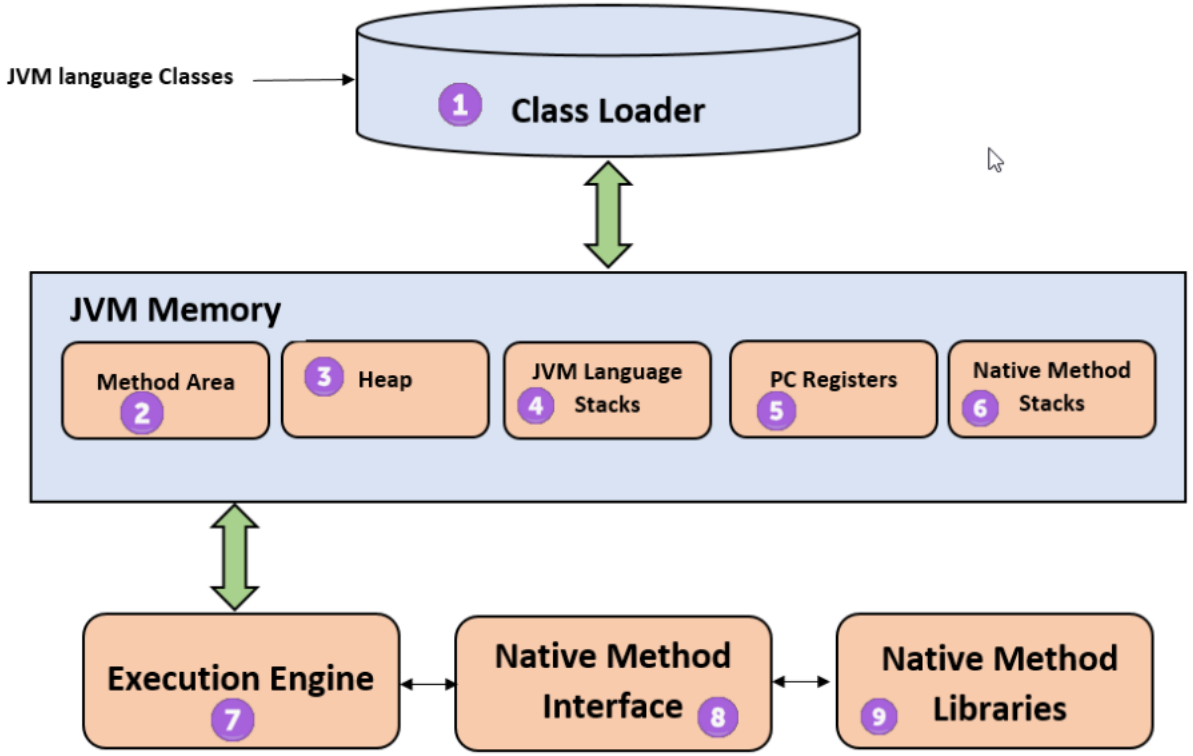
Dalam bahasa pemrograman lain, *compiler* menghasilkan kode mesin untuk sistem tertentu.
Namun, *compiler* Java menghasilkan kode untuk mesin virtual yaitu Java Virtual Machine.

Menurut Info World, JVM memiliki dua fungsi utama. Pertama, untuk memungkinkan program Java berjalan pada perangkat atau sistem operasi apa pun.

Fungsi ini dikenal sebagai prinsip WORA yang telah dijelaskan di atas.

Kemudian yang kedua adalah untuk mengelola dan mengoptimalkan memori program.

Arsitektur dan Cara Kerja JVM



Arsitektur dan Cara Kerja JVM

1. *Class loader,* *class loader* secara umum bertanggung jawab untuk tiga tugas berikut.

a. Loading

Class loader membaca file .class, menghasilkan data biner yang sesuai dan menyimpannya di area metode. Untuk setiap file .class, JVM menyimpan informasi berikut di area metode.

Nama kelas yang dimuat sepenuhnya dan kelas induk langsungnya.

Apakah file .class terkait dengan Class atau *Interface* atau *Enum.*

Modifier, variabel, informasi metode dan sebagainya.

Setelah memuat file .class, JVM membuat objek berjenis *Class* untuk mewakili file ini di *memory heap*.

Objek ini adalah tipe *Class* yang telah ditentukan sebelumnya dalam paket java.lang.

Objek *Class* ini dapat digunakan oleh pemrogram untuk mendapatkan informasi tingkat kelas seperti nama kelas, nama induk, metode, informasi variabel, dan lainnya.

Arsitektur dan Cara Kerja JVM

b. Linking

Tugas *linking* pada JVM adalah melakukan verifikasi, persiapan, dan sekali waktu melakukan resolusi.

Verifikasi dilakukan memastikan kebenaran file .class yaitu memeriksa apakah file ini diformat dengan benar dan dihasilkan oleh kompiler yang valid atau tidak. Jika verifikasi gagal, kamu akan mendapatkan pengecualian run-time java.lang.VerifyError.

Persiapan, JVM akan mengalokasikan memori untuk variabel kelas dan menginisialisasi memori ke nilai default.

Resolusi, yaitu proses mengganti referensi simbolik dari tipe dengan referensi langsung. Ini dilakukan dengan mencari ke area metode untuk menemukan entitas yang direferensikan.

c. Initialization

Dalam fase ini, semua variabel statis ditetapkan dengan nilainya yang ditentukan dalam kode dan blok statis (jika ada).

Ini dijalankan dari atas ke bawah dalam sebuah kelas dan berdasarkan hierarki kelas.

Arsitektur dan Cara Kerja JVM

2. JVM Memory

a. Method area

Di area ini, semua informasi tingkat kelas seperti nama kelas, nama kelas induk langsung, metode, dan informasi variabel, dan sebagainya isimpan, termasuk variabel statis.

Hanya ada satu area metode per JVM, dan itu adalah sumber daya bersama.

b. Heap area

Informasi dari semua objek disimpan di *heap area*. Meski demikian, hanya ada satu *heap area* per JVM. Ini juga merupakan bagian dari sumber daya bersama.

Arsitektur dan Cara Kerja JVM

c. Stack area

Untuk setiap *thread*, JVM membuat satu tumpukan *run-time* yang disimpan di *stack area*. Setiap blok tumpukan ini disebut catatan aktivasi/bingkai tumpukan yang menyimpan panggilan metode.

Semua variabel lokal dari metode itu disimpan dalam bingkai yang sesuai. Setelah *thread* dihentikan, tumpukan *run-time*-nya akan dihancurkan oleh JVM. Sayangnya, ini bukan bagian dari sumber daya bersama.

d. PC registers

PC registers menyimpan alamat instruksi eksekusi saat ini dari sebuah *thread*. Tentunya, setiap utas memiliki *PC registers* terpisah.

e. Native method stacks

Native stacks dibuat untuk setiap *thread*. *Stacks* ini menyimpan informasi mengenai *native method.*

Arsitektur dan Cara Kerja JVM

3. *Execution engine*

Execution engine menjalankan .class (*bytecode*). Mesin ini membaca *bytecode* baris demi baris, menggunakan data dan informasi yang ada di berbagai area memori dan menjalankan instruksi. Ini dapat diklasifikasikan dalam tiga bagian.

- *Interpreter*
- *Just-in-time compiler*
- *Garbage collector*

Arsitektur dan Cara Kerja JVM

4. Java Native Interface (JNI)

JNI adalah antarmuka yang berinteraksi dengan Native Method Libraries dan menyediakan perpustakaan asli (C, C ++) yang diperlukan untuk eksekusi.

Ini memungkinkan JVM memanggil library C atau C++ dan dipanggil oleh library C atau C ++ yang mungkin khusus untuk perangkat keras.

Tanpa Java Virtual Machine, program yang kamu buat tidak akan berjalan dengan lancar terlepas dari sistem operasi yang digunakan.

Apakah itu JRE?

Java Run-time Environment (JRE) adalah bagian dari Java Development Kit (JDK). Merupakan distribusi perangkat lunak yang dapat di download JRE secara gratis yang memiliki Java Class Library, alat khusus, dan JVM yang berdiri sendiri.

Ini adalah lingkungan paling umum yang tersedia di perangkat untuk menjalankan program java. Kode sumber Java dikompilasi dan diubah menjadi bytecode Java. Jika Anda ingin menjalankan bytecode ini di platform apa pun, Anda harus download JRE.

JRE memuat kelas, memverifikasi akses ke memori, dan mengambil sumber daya sistem. JRE bertindak sebagai lapisan di atas sistem operasi.

Apakah itu JRE?

Ini juga termasuk:

- Teknologi yang digunakan untuk penyebaran seperti Java Web Start.
- Toolkit untuk antarmuka pengguna seperti Java 2D.
- Perpustakaan integrasi seperti Java Database Connectivity (JDBC) dan Java Naming and Directory Interface (JNDI).
- Perpustakaan seperti Lang dan util.
- Pustaka dasar lainnya seperti Java Management Extensions (JMX), Java Native Interface (JNI) dan Java for XML Processing (JAX-WS).

Apakah itu JRE?

Kita dapat melihat perangkat lunak sebagai rangkaian lapisan yang berada di atas perangkat keras sistem. Setiap lapisan menyediakan layanan yang akan digunakan (dan diperlukan) oleh lapisan di atasnya. Java Runtime Environment adalah lapisan perangkat lunak yang berjalan di atas sistem operasi komputer, menyediakan layanan tambahan khusus untuk Java.

JRE memperhalus keragaman sistem operasi, memastikan bahwa program Java dapat berjalan di hampir semua OS tanpa modifikasi. Ini juga menyediakan layanan bernilai tambah. Manajemen memori otomatis adalah salah satu layanan JRE yang paling penting, memastikan bahwa pemrogram tidak perlu mengontrol alokasi dan realokasi memori secara manual.

Singkatnya, JRE adalah semacam meta-OS untuk program Java. Ini adalah contoh klasik *abstraksi*, mengabstraksi sistem operasi yang mendasarinya menjadi platform yang konsisten untuk menjalankan aplikasi Java.

Bagaimana JRE Bekerja Dengan JVM?

Java Virtual Machine adalah sistem perangkat lunak yang berjalan bertanggung jawab untuk melaksanakan program-program Java hidup. JRE adalah sistem pada disk yang mengambil kode Java Anda, menggabungkannya dengan pustaka yang diperlukan, dan memulai JVM untuk menjalankannya.

JRE berisi pustaka dan perangkat lunak yang perlu dijalankan oleh program Java Anda. Sebagai contoh, pemuat kelas Java adalah bagian dari Lingkungan Waktu Proses Java.

Perangkat lunak penting ini memuat kode Java yang dikompilasi ke dalam memori dan menghubungkan kode tersebut ke pustaka kelas Java yang sesuai.

Bagaimana JRE Bekerja Dengan JVM?

JRE memiliki instance JVM dengannya, kelas perpustakaan dan alat pengembangan. Untuk memahami cara kerja JRE, mari kita lihat contoh program “Hello World” sederhana.

```
import util. *

public static void main (String [] args) {

system.out.println (“ Halo dunia”);

}
```

Setelah Anda menulis program ini, Anda harus menyimpannya dengan ekstensi .java. Kompilasi program Anda. Output dari compiler Java adalah kode byte yang tidak bergantung pada platform.

Setelah kompilasi, kompilator menghasilkan file .class yang memiliki bytecode. Bytecode adalah platform independen dan berjalan di perangkat apa pun yang memiliki JRE. Dari sini, pekerjaan JRE dimulai. Untuk menjalankan program Java apa pun, Anda memerlukan JRE.

Bagaimana JRE Bekerja Dengan JVM?

Langkah-langkah berikut berlangsung saat runtime:

Class Loader
Pada langkah ini, class loader memuat berbagai class yang penting untuk menjalankan program. Pemuat kelas secara dinamis memuat kelas-kelas di Java Virtual Machine.
Saat JVM dimulai, tiga pemuat kelas digunakan:

- Loader kelas bootstrap
- Loader kelas ekstensi
- Loader kelas sistem

Pemverifikasi kode byte Pemverifikasi kode
byte dapat dianggap sebagai penjaga gerbang. Ini memverifikasi bytecode sehingga kode tersebut tidak membuat gangguan apa pun untuk interpreter. Kode diperbolehkan untuk diinterpretasikan hanya ketika melewati tes dari pemverifikasi Bytecode yang memeriksa format dan memeriksa kode ilegal.

Bagaimana JRE Bekerja Dengan JVM?

Interpreter
Setelah kelas dimuat dan kode diverifikasi, interpreter membaca kode assembly baris demi baris dan melakukan dua fungsi berikut:

- Jalankan Kode Byte
- Lakukan panggilan yang sesuai ke perangkat keras yang mendasarinya

Dengan cara ini, program berjalan di JRE.

Apa Itu SDK (Software Development Kit)?

SDK adalah seperangkat alat dan program perangkat lunak yang digunakan pengembang untuk membangun aplikasi untuk platform tertentu.

Berdasarkan pengertian SDK di atas, SDK atau Devkit berfungsi dengan cara yang hampir sama dengan API dan menawarkan serangkaian alat, pustaka, dokumentasi yang relevan, contoh kode, proses dan atau manual yang memungkinkan pengembang membuat aplikasi perangkat lunak pada platform tertentu

Mengacu pada pengertian SDK di atas, karena SDK seluler Anda dimaksudkan untuk digunakan di luar organisasi Anda, karena dimaksudkan untuk digunakan di luar perusahaan.

Apa Itu SDK (Software Development Kit)?

Nilai itu tergantung pada SDK Anda yang memiliki karakteristik berikut:

1. Mudah digunakan oleh pengembang lain
2. Dokumentasi lengkap untuk menjelaskan cara kerja kode Anda
3. Fungsionalitas yang cukup untuk menambah nilai pada aplikasi lain
4. Tidak memiliki efek negatif pada CPU, baterai atau konsumsi data perangkat seluler
5. Bermain dengan baik dengan SDK lainnya.

Idealnya, SDK harus bekerja dengan elegan, tetapi ketika waktu adalah esensi, pasti harus cukup baik selama Anda melakukan pekerjaan Anda.

Apa Itu SDK (Software Development Kit)?

Nilai itu tergantung pada SDK Anda yang memiliki karakteristik berikut:

1. Mudah digunakan oleh pengembang lain
2. Dokumentasi lengkap untuk menjelaskan cara kerja kode Anda
3. Fungsionalitas yang cukup untuk menambah nilai pada aplikasi lain
4. Tidak memiliki efek negatif pada CPU, baterai atau konsumsi data perangkat seluler
5. Bermain dengan baik dengan SDK lainnya.

Idealnya, SDK harus bekerja dengan elegan, tetapi ketika waktu adalah esensi, pasti harus cukup baik selama Anda melakukan pekerjaan Anda.

Tujuan SDK (Software Development Kit)?

Sebagaimana penjelasan Kami mengenai pengertian SDK di atas, SDK memberi para pengembang seperangkat alat yang diperlukan untuk memberikan antarmuka akhir yang jelas dan standar kepada pengguna akhir. Selain itu, pengembang tidak harus menemukan kembali roda ketika datang ke fungsi aplikasi standar seperti lokasi, penyimpanan data, geofencing, otorisasi pengguna, dll. Mereka memiliki berbagai fitur canggih, penanganan kesalahan, kinerja yang konsisten dan penggunaan kembali kode yang menghemat banyak upaya pengembang perangkat lunak.

Manfaat SDK (Software Development Kit)?

Berdasarkan pengertian SDK (Software Development Kit) di atas, adapun beberapa manfaat SDK (Software Development Kit) adalah sebagai berikut:

1. Integrasi Lebih Cepat

Jika Anda mencoba untuk menutup beberapa kesepakatan, semua kesepakatan ini akan dipercepat oleh SDK seluler. Devkit Anda membantu Anda mempersingkat siklus penjualan dengan menyederhanakan integrasi ke dalam tumpukan teknologi yang ada.

2. Pengembangan Efisien

Jika Anda menganggap bahwa aplikasi media Android bahkan menggunakan lebih banyak SDK dari sekitar 18,2 penyedia pihak ketiga ketika itu adalah sebuah game, Anda akan segera melihat bahwa tidak ada pengembang perangkat lunak yang memiliki waktu untuk kode setiap alat tunggal.

Manfaat SDK (Software Development Kit)?

3. Peningkatan Reach (Jangkauan)

Jika produk Anda berharga dan SDK yang disertakan memungkinkan tingkat interoperabilitas yang besar, Anda meningkatkan kemungkinan alat lain dapat dan akan diintegrasikan ke dalam produk Anda yang akan meningkatkan kesadaran merek Anda dan tentu saja meningkatkan jangkauan Anda.

4. Kontrol Merek

Untuk mengurangi risiko, Anda dapat menggunakan SDK untuk lebih mengontrol elemen antarmuka pengguna yang muncul di aplikasi lain. Dengan cara ini, Anda dapat menentukan tidak hanya bagaimana produk diintegrasikan ke dalam aplikasi lain, tetapi juga bagaimana tampilannya. Sementara itu, harus dipastikan bahwa fungsi yang paling penting dilindungi dari gangguan dan dapat merusak keramahan pengguna.

Java Developer Fundamentals

Java Fundamental

Java Fundamental

1. Kerangka Program Java
2. Variabel
3. Cast dan Promosi Variabel
4. Operator
5. Penggunaan If, Else If, Else
6. Penggunaan While dan Do While
7. Penggunaan For
8. Deklarasi Array Java
9. Inisialisasi Array Java
10. String Java
11. Kelas dan Obyek Java