

# Struktur Dasar Package

Kita telah menginstal *tools* yang diperlukan dan membuat program sederhana yang menampilkan 'Hello, World!' di bagian Pendahuluan. Kini kita akan mempelajari struktur dasar dari program Java. Setiap *class* di program Java diletakkan dalam sebuah *package*, yakni mekanisme penempatan/penamaan *class* agar lebih terstruktur atau modular.

Mekanisme *package* sangat diperlukan. Salah satunya untuk membedakan *class* dengan nama yang sama. Misalnya institusi Apache membuat *class* `StringUtils`. Begitu juga dengan institusi Spring yang membuat *class* `StringUtils`. Lalu bagaimana kita membedakan keduanya jika kita ingin menggunakan kedua *library* tersebut secara bersamaan? Karena pada praktiknya hampir setiap aplikasi Java dibuat dengan *library* dari eksternal. Yup, coding Java sama dengan bermain di dunia *open-source*.

## Langkah penamaan *package*:

- Umumnya menggunakan nama domain institusi (dengan penulisan di balik) pembuat program tersebut.
- Sertakan nama program/aplikasi. Ikuti dengan nama modul-modulnya.
- Beri pemisah dengan tanda titik.

Anda tidak perlu bingung, berikut contohnya:

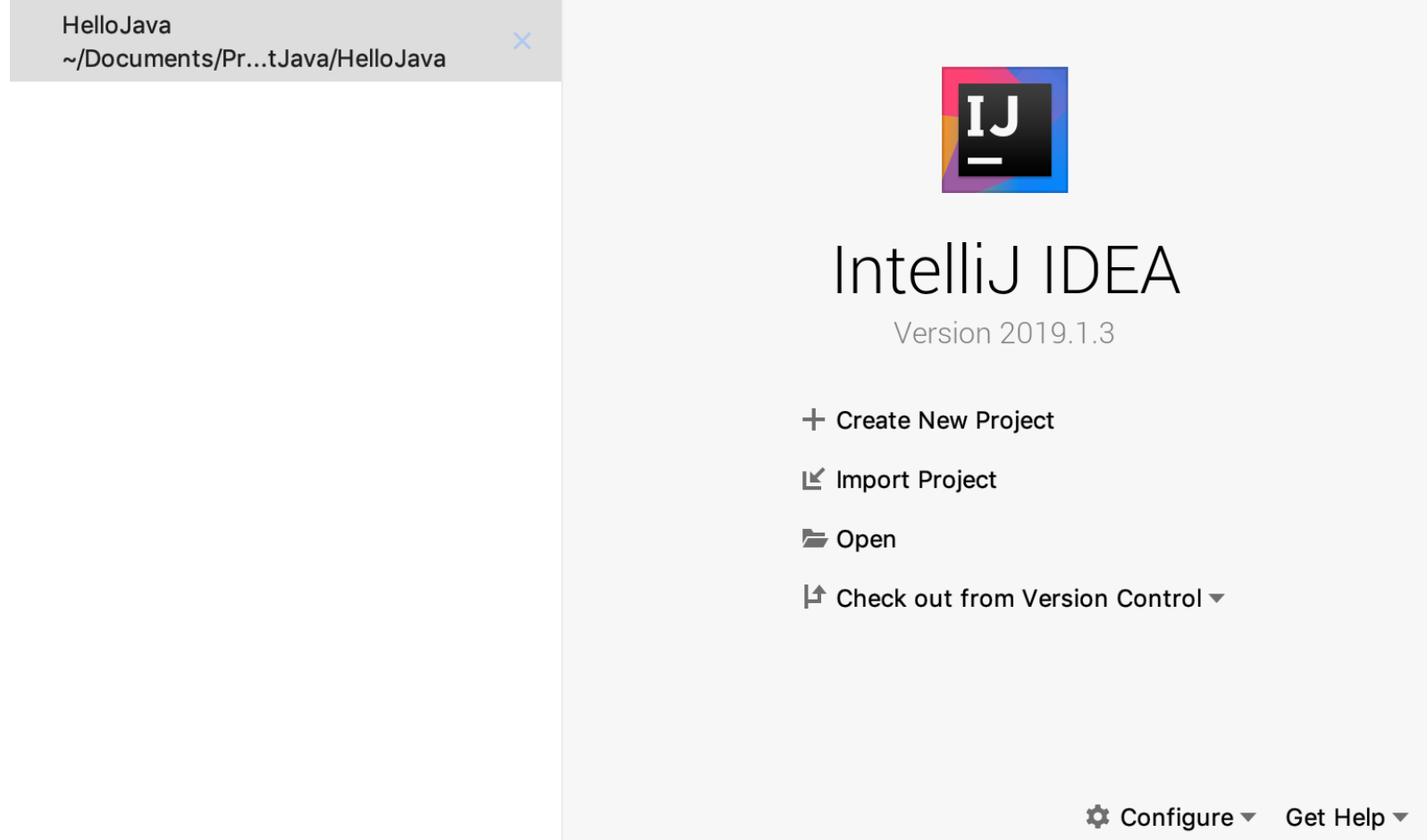
Nama domain institusi	dicoding.com
Nama aplikasi	Java Fundamental
Nama modul	Basic
Maka nama <i>package</i> dapat kita tulis seperti ini	<code>com.dicoding.javafundamental.basic</code>

Perhatikan tabel di atas, nama *package* harus menggunakan huruf kecil tanpa spasi. *Package* akan sinkron dengan *directory* dari berkas (*file*) `.java` (*source-code*) dan juga hasil *compile*-nya yaitu file `.class`.

## Codelab Struktur Dasar

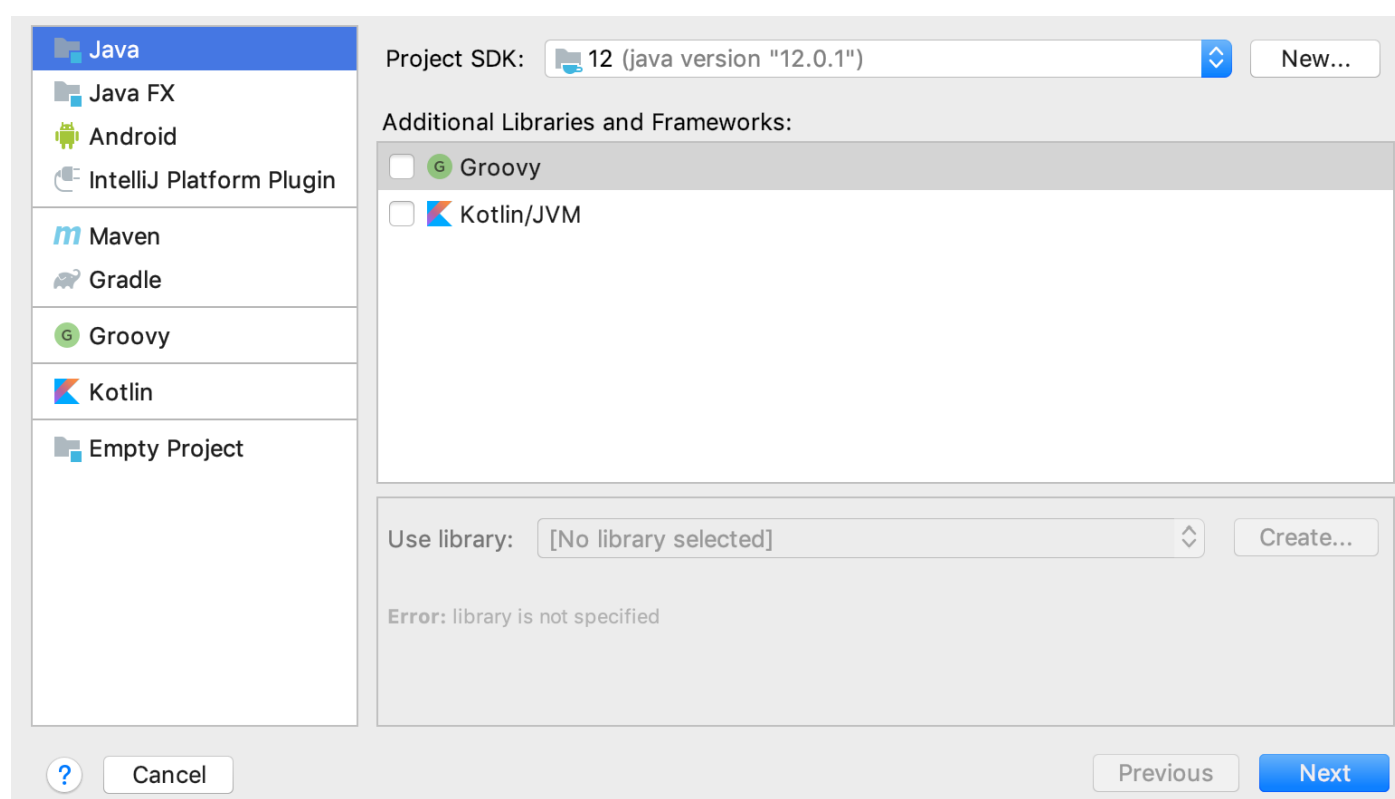
Di bagian pendahuluan Anda sudah membuat program `HelloJava` dengan sebuah *class* `Main.Java`, namun jika dilihat kembali Anda tidak akan melihat informasi *package* di sana. Dalam kasus ini program `HelloJava` menggunakan *default package*, di mana hal tersebut adalah cara yang sangat tidak disarankan. Oleh karena itu mari kita mulai koding dengan membuat kelas `Main` dalam *package*.

1. Buka aplikasi `IntelliJ`.

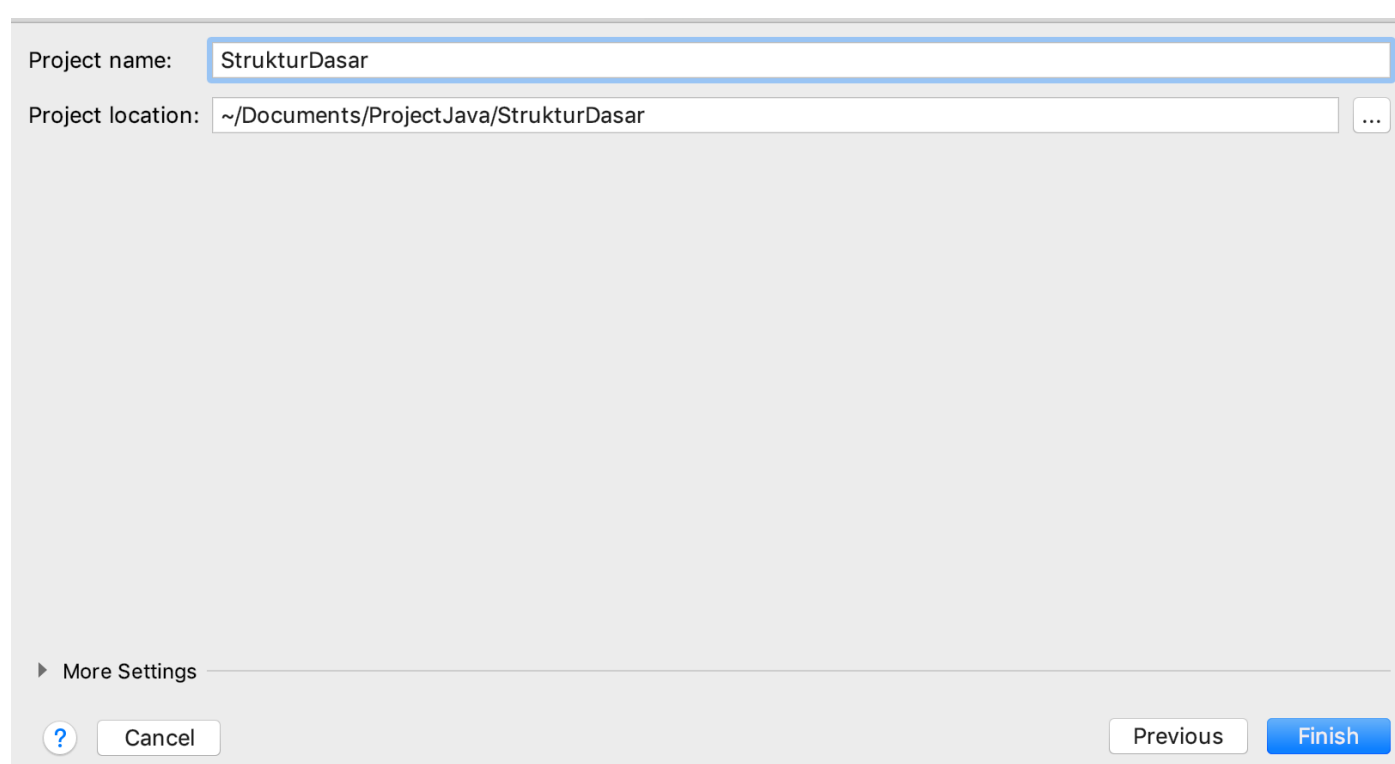


2. Setelah mengetahui beberapa penjelasan tersebut, sekarang kita pilih Create New Project.

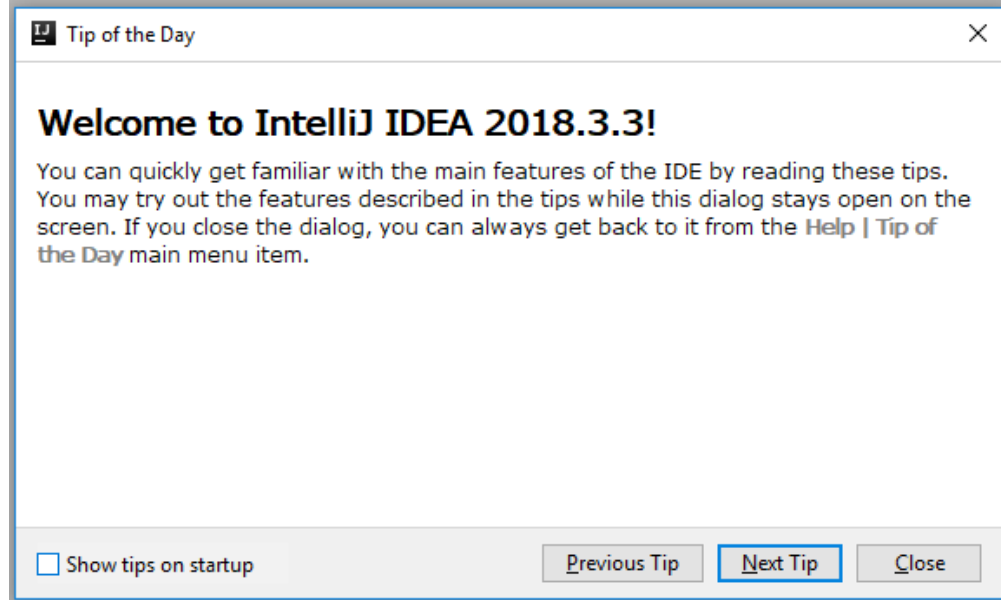
3. Pastikan pilih Java di panel kiri lalu klik tombol Next.



4. Kemudian, masukkanlah Project name dengan **StrukturDasar**. Untuk lokasi proyeknya, Anda bisa memilih lokasi yang diinginkan. Kemudian Klik tombol Finish.

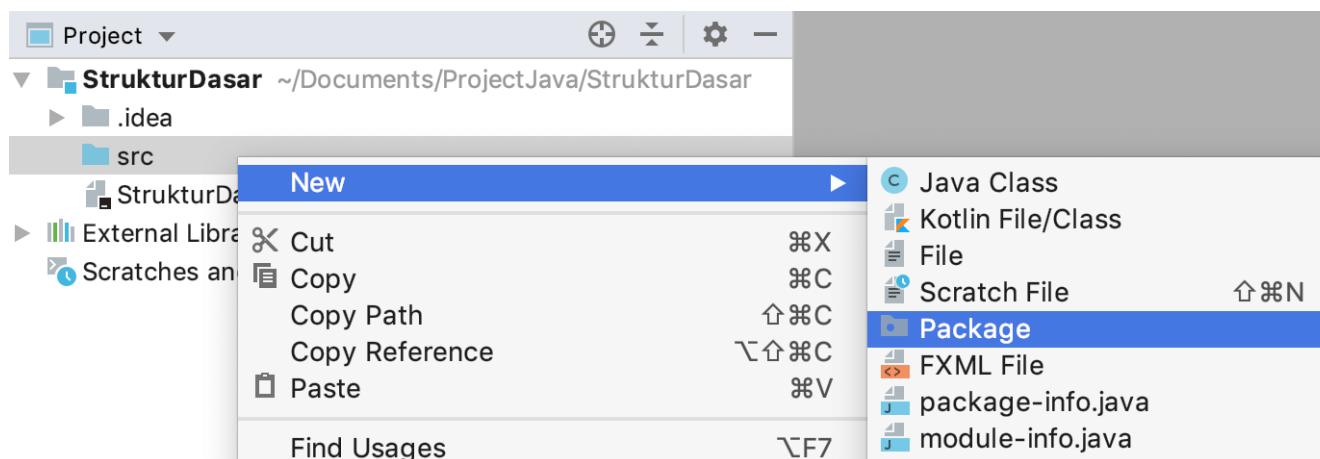


Catatan: Jika ada *popup-window Tip of the Day*, silakan lihat untuk membantu Anda beradaptasi dengan aplikasi **IntelliJ**.

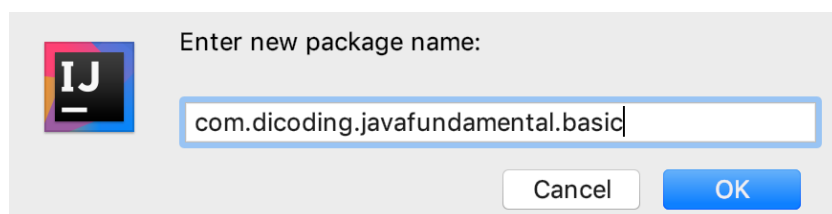


Tutuplah jendela *Tip of the Day* untuk melanjutkan ke langkah berikutnya.

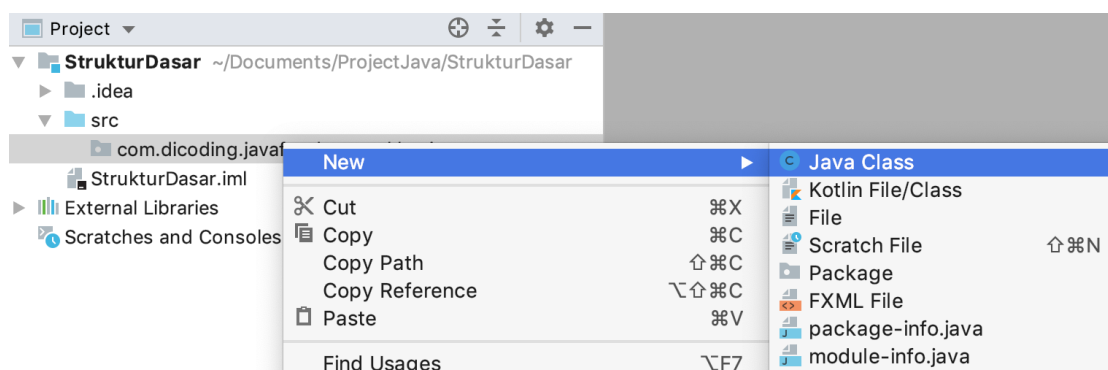
5. Buat *package* baru dengan cara sebagai berikut. Di panel proyek sebelah kiri, klik kanan directory `src`. Kemudian pilih New - Package.



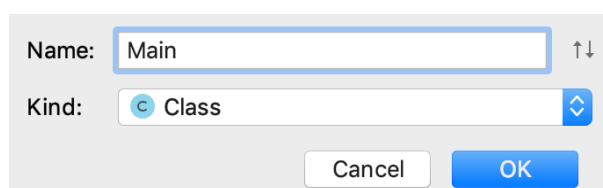
Kemudian masukkan package name-nya dengan `com.dicoding.javafundamental.basic`.



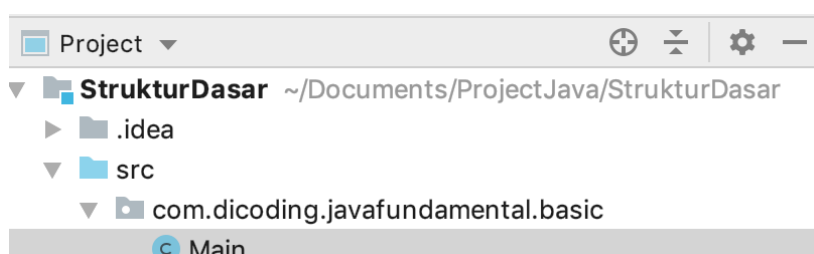
6. Selanjutnya buat *class* baru di dalam *package* yang baru saja Anda buat dengan cara, klik kanan di *package* tersebut dan pilihlah New - Java Class.



Kemudian masukan nama *class* tersebut dengan `Main` dan klik OK.



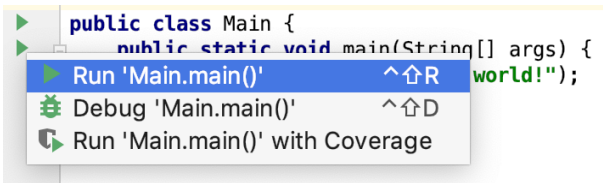
7. Perhatikan posisi dari *class* `Main` yang Anda buat sebelumnya. *Class* tersebut akan berada di dalam *directory* yang sama dengan nama *package*-nya.



8. Tambahkan kode sebagai berikut di *class* `Main`.


```
1. package com.dicoding.javafundamental.basic;
2.
3. public class Main {
4.     public static void main(String[] args) {
5.         System.out.println("Hello world!");
6.     }
7. }
```

9. Jika Anda perhatikan terdapat kode `package` yang merupakan lokasi dari kelas `Main`. Kelas `Main` berada di dalam package `com.dicoding.javafundamental.basic`.

10. Lalu jalankanlah *class* `Main.java` dengan klik salah satu tombol di sebelah kiri panel Editor. Lalu pilih Run 'Main.main()'.  


11. Hasilnya akan muncul di panel  di bawah:

Hello world!

12. 

Apa yang sudah kita lakukan sebenarnya sama saja dengan bagian Pendahuluan. Bedanya, Anda meletakkan *class* `Main` di dalam *package* `com.dicoding.javafundamental.basic`. Ayo kita lanjut!

Ketika setiap *class* diletakkan di dalam sebuah *package*, bagaimana cara memanggil *class* tertentu di antara *class* lain yang berbeda *package*-nya? Caranya adalah dengan memanggil nama *package* secara lengkap, kemudian diikuti nama *class*-nya. Hal ini biasa disebut dengan istilah *Fully Qualified Name*. Misalnya, *fully qualified name* dari *class* `Main` yang baru saja kita buat adalah:

```
1. com.dicoding.javafundamental.basic.Main
```

## Import

*Import* digunakan untuk menyederhanakan pemanggilan *class* yang berbeda *package*. Alhasil, Anda tak perlu menyebutkan *fully qualified name* dari *class* yang ingin digunakan. Dari pengertian ini juga tersirat kita tak perlu juga menyebutkan *fully qualified name* jika dalam *package* yang sama.

Khusus untuk *class* dari *package* `java.lang`, ia tidak perlu dipanggil menggunakan *fully qualified name*. Artinya tidak perlu menggunakan *import* saat memanggilnya, misalnya class `java.lang.System` yang kita gunakan untuk *print* “Hello world!”. Perhatikan bahwa kode di bawah ini:

1. `System.out.println("Hello world!");`

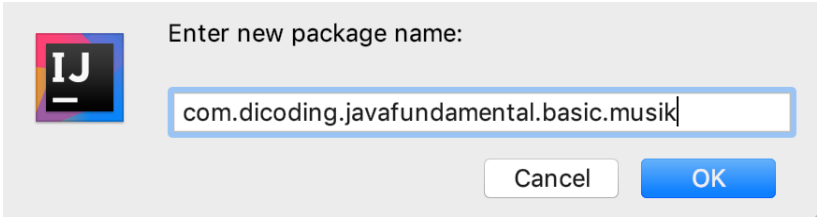
Pada kode di atas, kita hanya memanggil *class* dengan nama *class* System tanpa nama *package*-nya.

Bagaimana kalau kita *import* beberapa *class* dari *package* yang sama? Kita bisa menggunakan *wildcard* (simbol \*) untuk menggantikan nama *class*-nya. Artinya, Java akan otomatis mengenali seluruh class dari *package* tersebut.

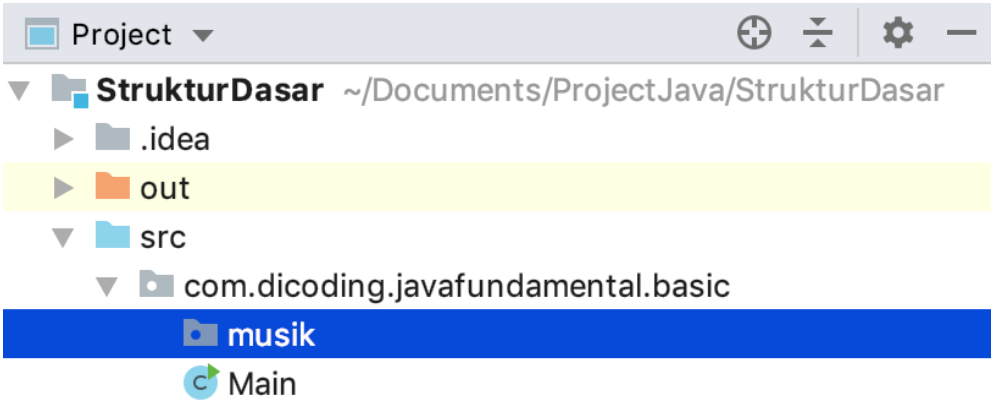
## Codelab Import

OK untuk lebih jelasnya, mari kita langsung koding saja.

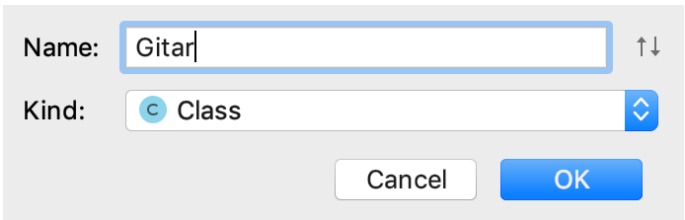
1. Buka kembali project StrukturDasar.
2. Buatlah kembali *package* dengan cara klik kanan directory src, kemudian pilih New - Package. Masukkanlah nama *package*-nya dengan `com.dicoding.javafundamental.basic.musik`.



Maka tampilan package saat ini menjadi seperti ini:



3. Selanjutnya buatlah *class* `Gitar` di dalam *package* `musik`.



Tambahkan kode berikut di class `Gitar`:

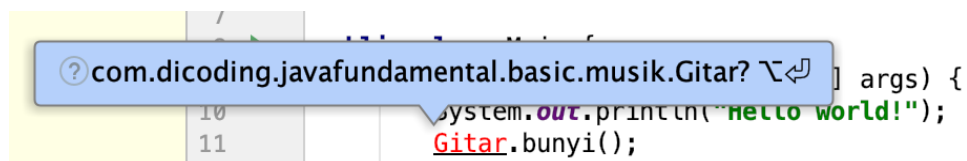
```
1. public class Gitar {
2.     public static void bunyi(){
3.         System.out.println("jrenggg..");
4.     }
5. }
```

4. Buka kembali *class* **Main**, tambahkan kode seperti di bawah ini:

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello world!");

        Gitar.bunyi();
    }
}
```

5. Perhatikan IntelliJ akan memberi tanda eror berwarna merah karena *class* **Gitar** tidak dikenali. Untuk itu kita perlu *import class* **Gitar** agar dikenali. Pindahkan kursor ke baris kode yang ditandai dengan *error* tersebut, lalu tekan alt + Enter(Windows) atau option + return(MacOS).

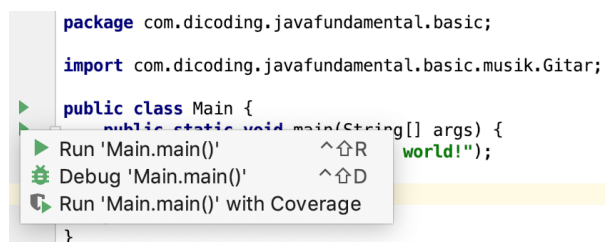


```
10     System.out.println("Hello world!");
11     Gitar.bunyi();
```

Otomatis IntelliJ akan menambahkan baris kode seperti berikut:

```
1. import com.dicoding.javafundamental.basic.musik.Gitar;
```

6. Lalu jalankanlah *class* **Main** dengan klik tombol run pastikan tidak ada eror.



```
package com.dicoding.javafundamental.basic;
import com.dicoding.javafundamental.basic.musik.Gitar;

public class Main {
    public static void main(String[] args) {
        System.out.println("Hello world!");
        Gitar.bunyi();
    }
}
```

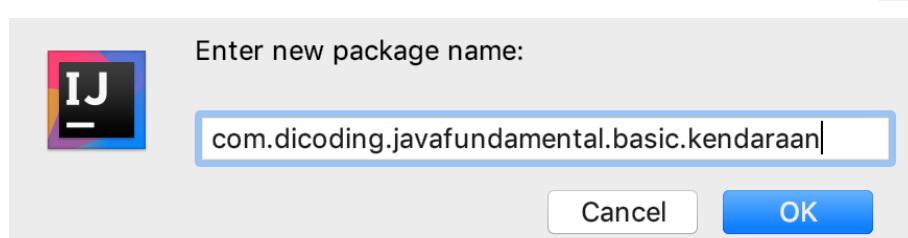
Hasilnya akan jadi seperti ini:

```
Hello world!
jrenggg..
```



```
Run: Main
/Library/Java/JavaVirtualMachines/jdk-12.0.1.jdk/Contents/Home/bin/java "-javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=58357:/Applications/IntelliJ IDEA CE.app/Contents/bin" -Dfile.encoding=UTF-8 -classpath /Users/gilangramadhan/Documents/ProjectJava/StrukturDasar/out/production/StrukturDasar com.dicoding.javafundamental.basic.Main
Hello world!
jrenggg..
Process finished with exit code 0
```

7. Selanjutnya buatlah kembali *package* baru dengan cara klik kanan directory src, kemudian pilih New - Package. Masukkanlah nama *package*-nya dengan: **com.dicoding.javafundamental.basic.kendaraan**.

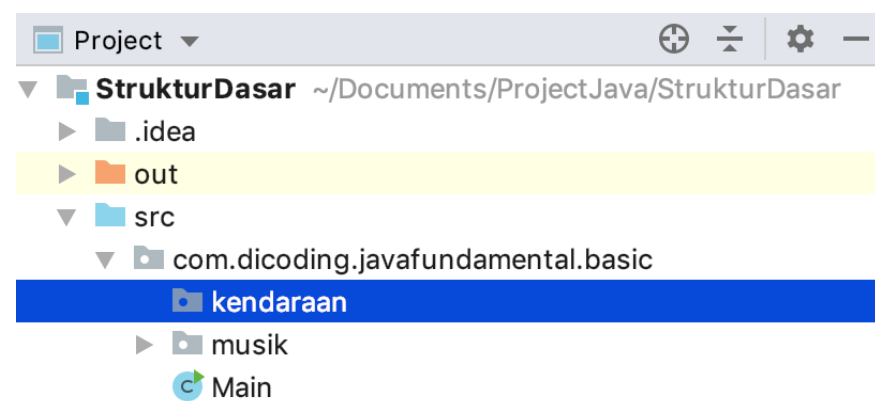


Enter new package name:

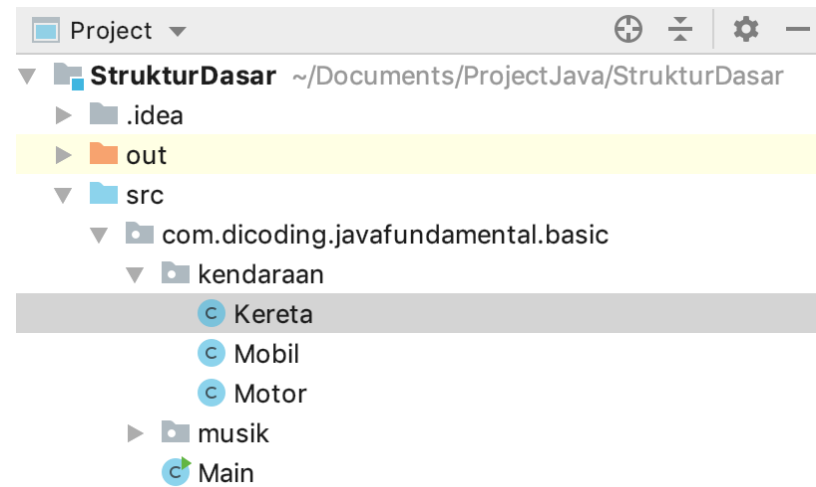
```
com.dicoding.javafundamental.basic.kendaraan
```

Cancel OK

Maka struktur package kali ini akan menjadi seperti ini:



8. Selanjutnya buat *class* baru dengan nama **Mobil**, **Motor**, dan **Kereta** di dalam *package* tersebut.



Tambahkan baris kode berikut di masing-masing *class* tersebut.

Pertama buka *class* **Mobil** dan ubahlah kode di dalamnya menjadi seperti iniberikut:

```
1. public class Mobil {
2.     public static void jumlahBan(){
3.         System.out.println("Ban mobil 4");
4.     }
5. }
```

Selanjutnya bukalah *class* **Motor** dan ubahlah kode di dalamnya:

```
1. public class Motor {
2.     public static void jumlahBan() {
3.         System.out.println("Ban motor 2");
4.     }
5. }
```

Terakhir bukalah *class* **Kereta** dan ubah juga kode di dalamnya menjadi seperti ini:

```
1. public class Kereta {
2.     public static void jumlahBan() {
3.         System.out.println("Ban kereta banyak!");
4.     }
5. }
```

9. Bukalah *class* **Main** lalu ketik (jangan *copy-paste*) kode berikut.

```
1. Mobil.jumlahBan();
```



10. Ketika kode di atas mulai diketik sebagian, maka **IntelliJ** akan memberi saran kode (dikenal dengan istilah *auto-complete*) seperti gambar di bawah. Tekan Enter agar **IntelliJ** melengkapi baris kode.



Dengan bantuan *auto complete*, Anda tidak perlu melakukan *import* secara manual.

11. Lengkapi *class* **Main** dengan kode sebagai berikut :

1. **Motor**.jumlahBan();
2. **Kereta**.jumlahBan();

12. Baris *import* dapat Anda sederhanakan yang awalnya seperti ini:

1. **import** com.dicoding.javafundamental.basic.kendaraan.**Kereta**;
2. **import** com.dicoding.javafundamental.basic.kendaraan.**Mobil**;
3. **import** com.dicoding.javafundamental.basic.kendaraan.**Motor**;
4. **import** com.dicoding.javafundamental.basic.musik.**Gitar**;

Menjadi:

1. **import** com.dicoding.javafundamental.basic.kendaraan.\*;
2. **import** com.dicoding.javafundamental.basic.musik.**Gitar**;


Maka kelas **Main** menjadi seperti ini:

1. **package** com.dicoding.javafundamental.basic;
- 2.
3. **import** com.dicoding.javafundamental.basic.kendaraan.\*;
4. **import** com.dicoding.javafundamental.basic.musik.**Gitar**;
- 5.
6. **public class** **Main** {
7.     **public static void** main(**String**[] args) {
8.         **System.out.println**("Hello world!");
- 9.
10.         **Gitar.bunyi**();
- 11.
12.         **Mobil.jumlahBan**();
13.         **Motor.jumlahBan**();
14.         **Kereta.jumlahBan**();
15.     }



Catatan:

Penggunaan *wildcard* (\*) kadang tidak disarankan jika *class* di dalam *package* tersebut terlalu banyak. Sebabnya, ia memperlama waktu untuk pencarian *class* tersebut. Untuk itu kita serahkan ke **IntelliJ** mengatur *import* yang optimal. Pilih menu Code | Optimize Imports. Baris *import* akan kembali ke sebelumnya tanpa menggunakan *wildcard* (\*).

13. Lalu jalankan class Main dengan klik tombol  dan pastikan tidak ada *error* di dalamnya.

```
package com.dicoding.javafundamental.basic;

import com.dicoding.javafundamental.basic.kendaraan.*;
import com.dicoding.javafundamental.basic.musik.Gitar;

public class Main {
    public static void main(String[] args) {
        System.out.println("Hello world!");

        Mobil.jumlahBan();
        Motor.jumlahBan();
        Kereta.jumlahBan();
    }
}
```

14. Maka hasilnya jadi seperti ini:

Hello word!!

jrenggg..

Ban mobil 4

Ban motor 2

Ban kereta banyak!



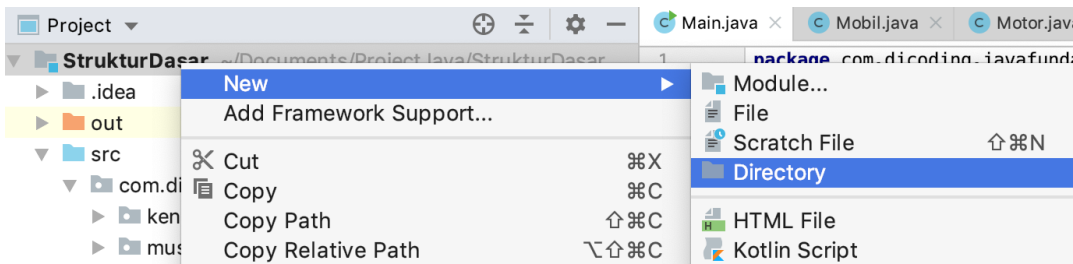
# Classpath

*Classpath* adalah mekanisme di Java untuk menemukan *class* lain. Biasanya *class* lain tersebut berasal dari *library* yang berbeda atau bahkan JDK itu sendiri (kita sudah memakai *class* **System** ). Jika Java tidak bisa menemukan *class* yang kita panggil melalui kode kita, akan terjadi *error* **ClassNotFoundException** atau **NoClassDefFoundError**. *Error* ini biasa terjadi bagi pemula Java karena biasanya belum paham konsep *classpath* atau ada kesalahan pengaturan *classpath*.

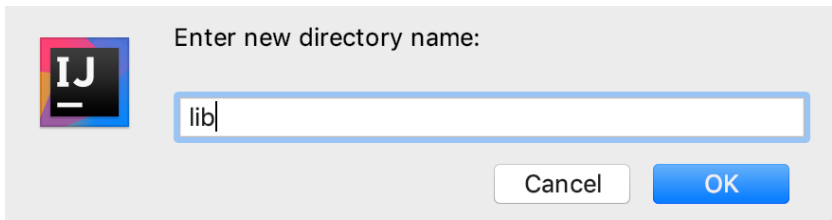
Sekarang tenang ya kalau ketemu *error* serupa. Kita kan sudah tahu bahwa terduga adalah problem di *classpath*. Jadi paham juga deh, dari mana kita mesti melakukan *troubleshooting*.

# Codelab Classpath

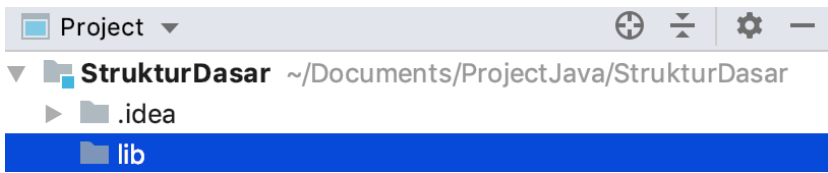
1. Buka kembali proyek **StrukturDasar**. Buat directory baru dengan cara klik kanan di proyek **StrukturDasar** seperti gambar di bawah:



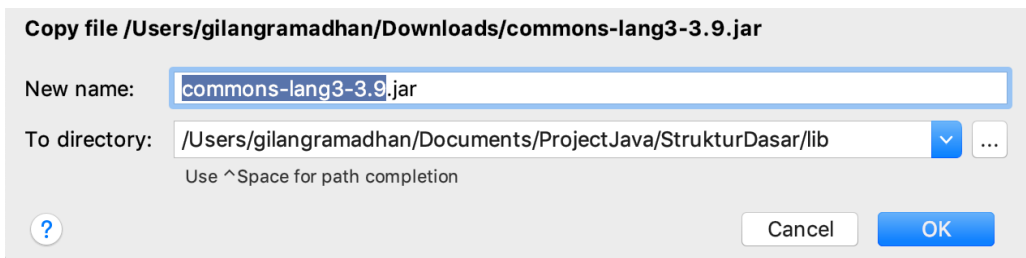
Selanjutnya masukkan directory name dengan **lib**.



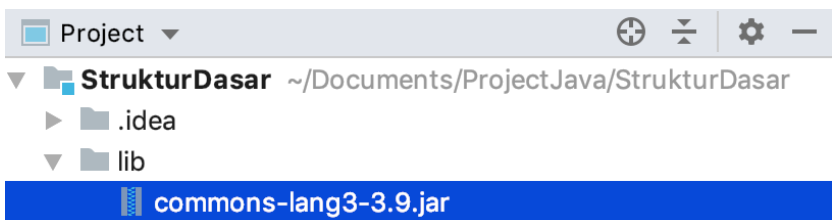
Maka akan jadi seperti ini:



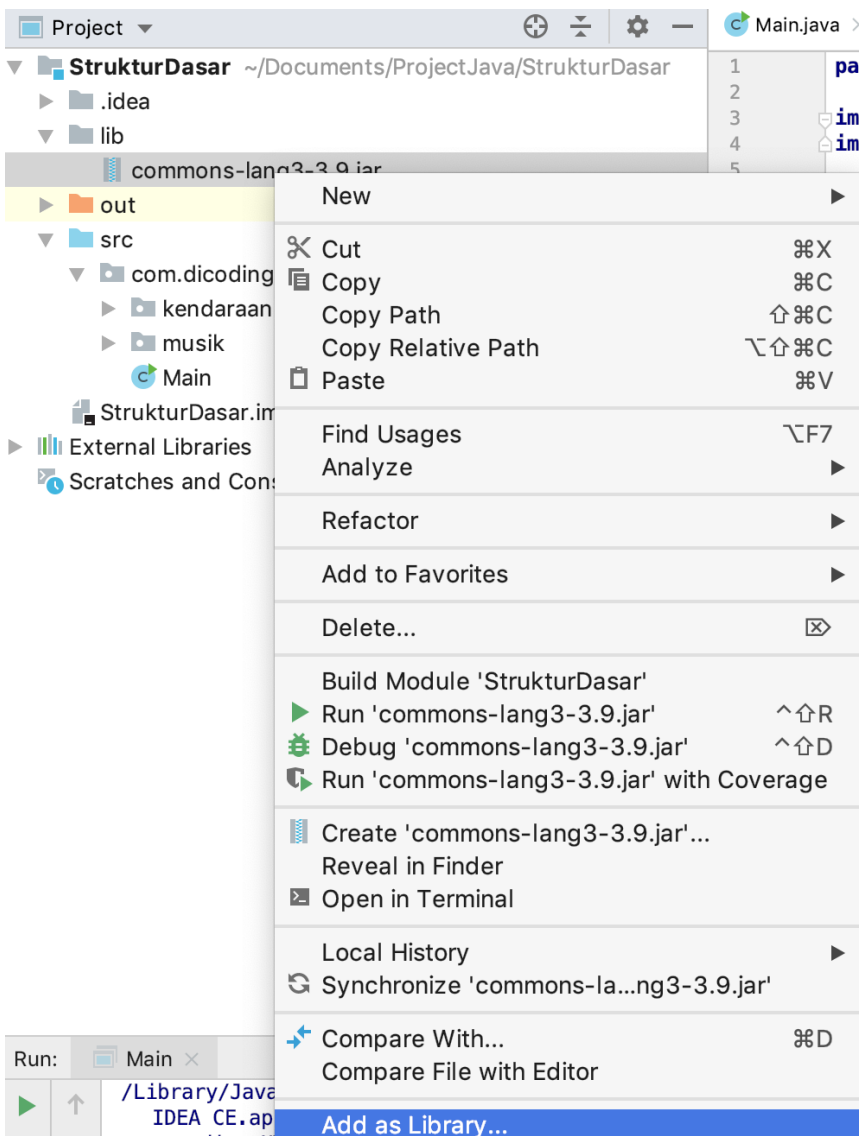
2. Unduh *library apachecommon-lang* dari tautan berikut:
- <https://repo1.maven.org/maven2/org/apache/commons/commons-lang3/3.9/commons-lang3-3.9.jar>
3. Salin berkas **.jar** tersebut ke directory lib.



Maka akan jadi seperti ini:



4. Klik kanan di **commons-lang3-3.9.jar** untuk menjadikan file **jar** menjadi library, pilihlah Add as Library....



Setelah itu biarkan namanya menjadi default dan tekan OK.

Name:

commons-lang3-3.9

Level:

Project Library

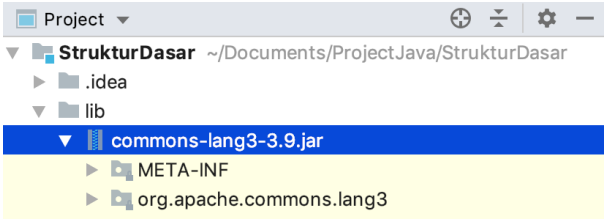
Add to module:

StrukturData

Cancel

OK

Maka hasilnya akan jadi seperti ini:



Perhatikan sekarang ada tanda panah di kiri sehingga kita bisa *browse* isi berkas **jar** tersebut. Jika tanda tersebut tidak ada, maka proses penambahan library yang Anda lakukan gagal.

5. Sekarang berkas **jar** sudah menjadi *library* di *project* **StrukturDasar**.

6. Tambahkan baris kode berikut di *class* **Main**. Sebaiknya ketik (koding) saja supaya *get a grip* tentang fitur *auto-completel*IntelliJ ini.

1. `Date` today = `new` `Date`();

2. `System.out.println`("Hari ini = " + today);

3. `Date` tomorrow = `DateUtils`.addDays(today, 1);

4. `System.out.println`("Besok = " + tomorrow);

Sehingga kelas Main.java menjadi seperti ini:

1. `package` com.dicoding.javafundamental.basic;

2.

3. `import` com.dicoding.javafundamental.basic.kendaraan.\*;

4. `import` com.dicoding.javafundamental.basic.musik.Gitar;

5. `import` org.apache.commons.lang3.time.DateUtils;

6.

7. `import` java.util.Date;

8.

9. `public class` Main {

10.     `public static void` main(`String`[] args) {

11.         `System.out.println`("Hello world!");

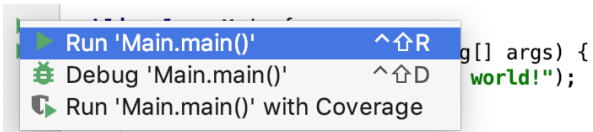
12.         Gitar.bunyi();

13.

14.         Mobil.jumlahBan();

15.         Motor.jumlahBan();

7. Jalankan *class* **Main** dengan klik tombol , pastikan tidak ada *error*.



Maka hasilnya akan menjadi seperti ini:

Hello word!!

jrenggg..

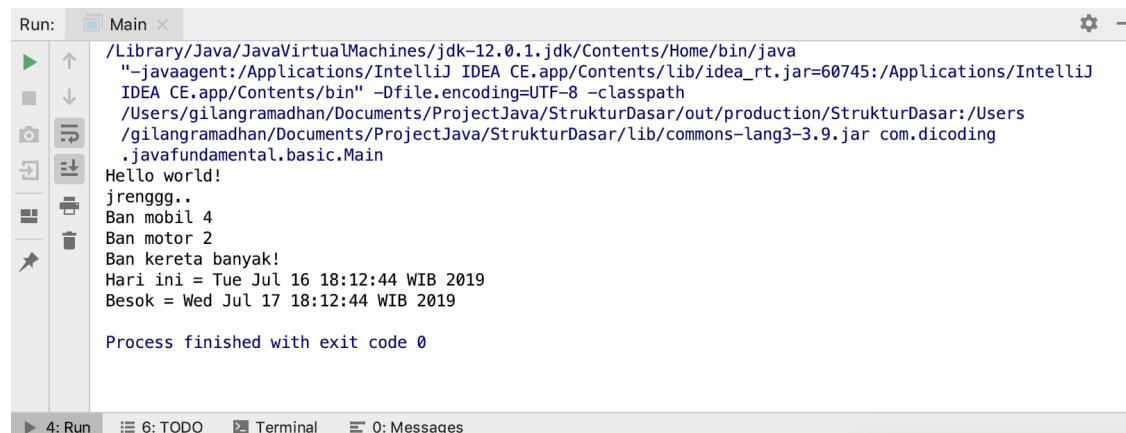
Ban mobil 4

Ban motor 2

Ban kereta banyak!

Hari ini = Tue Jul 16 18:12:44 WIB 2019

Besok = Wed Jul 17 18:12:44 WIB 2019



Praktik yang kita lakukan barusan adalah menambahkan *library* (file **jar**) secara manual ke proyek. Cara ini sebenarnya kurang efektif jika dilakukan untuk proyek besar. Bayangkan jika kita menggunakan *library* A yang bergantung (*dependencies*) ke *library* B, C, D, lalu *library* B bergantung ke *library* X dan Y. Cukup bikin repot kan? Solusinya, gunakan *tools* seperti maven atau gradle. Pembahasan maven dan gradle di luar cakupan dari *class* ini. Saran kami, bacalah referensi yang ada di bawah karena kedua *tools* tersebut sudah menjadi standar dalam *development* Java.

- <http://www.baeldung.com/java-classnotfoundexception-and-noclassdeffoundererror>
- <https://maven.apache.org>
- <https://gradle.org>
- <https://spring.io/guides/gs/maven>
- <https://spring.io/guides/gs/gradle>

## Pemaketan

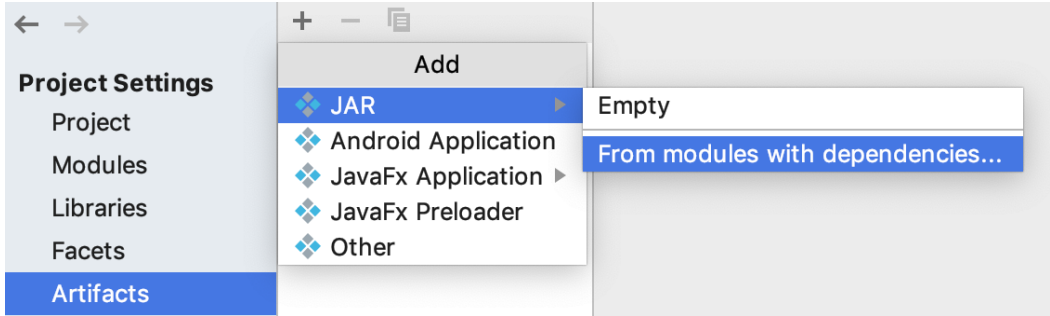
Setiap kode Java yang ditulis akan dikompilasi menjadi *bytecode* (file dengan *extension* **.class**) lalu pada akhirnya akan dipaketkan untuk didistribusikan. Bentuk pemaketan yang sering digunakan adalah bentuk JAR (Java ARchive), ada juga bentuk lain misal WAR (Web ARchive) dan EAR (Enterprise ARchive). Dari nama pemaketan ini bisa ditebak sebenarnya hanya berkas *archive* atau berkas zip. Isinya bisa diintip menggunakan winzip, winrar atau aplikasi sejenis.

Di dalam hasil pemaketan tersebut ada berkas metadata yang menjelaskan isi berkas JAR. *File* tersebut adalah **manifest.mf** yang diletakkan di *directory* META-INF.

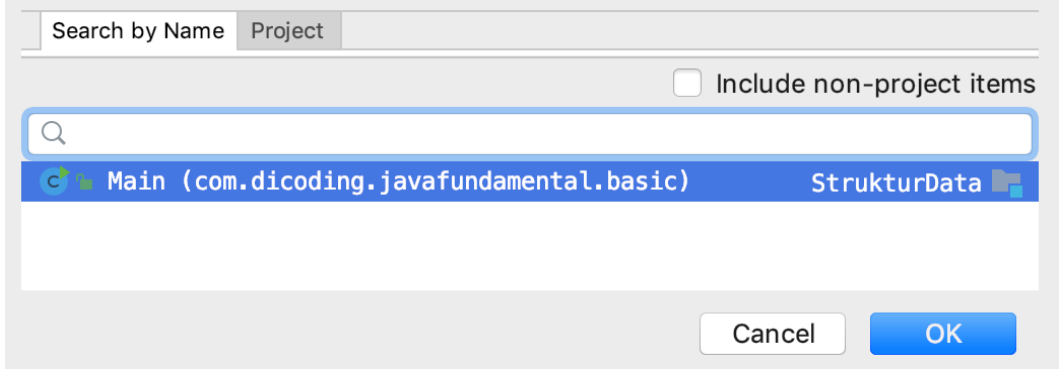
## Codelab Pemaketan

Mari kita coba paketkan project StrukturDasar dalam bentuk JAR.

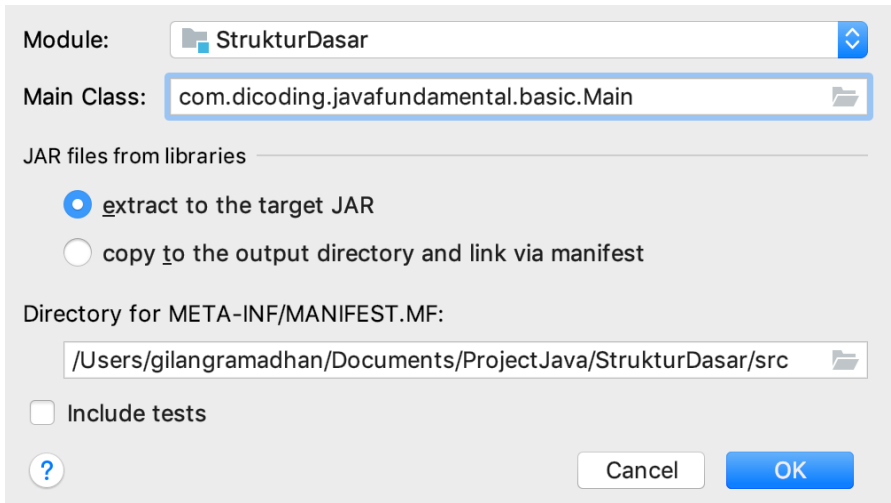
1. Buka kembali IntelliJ. File | Project Structure | Artifacts. Klik tombol + lalu pilih JAR | From modules with dependencies.



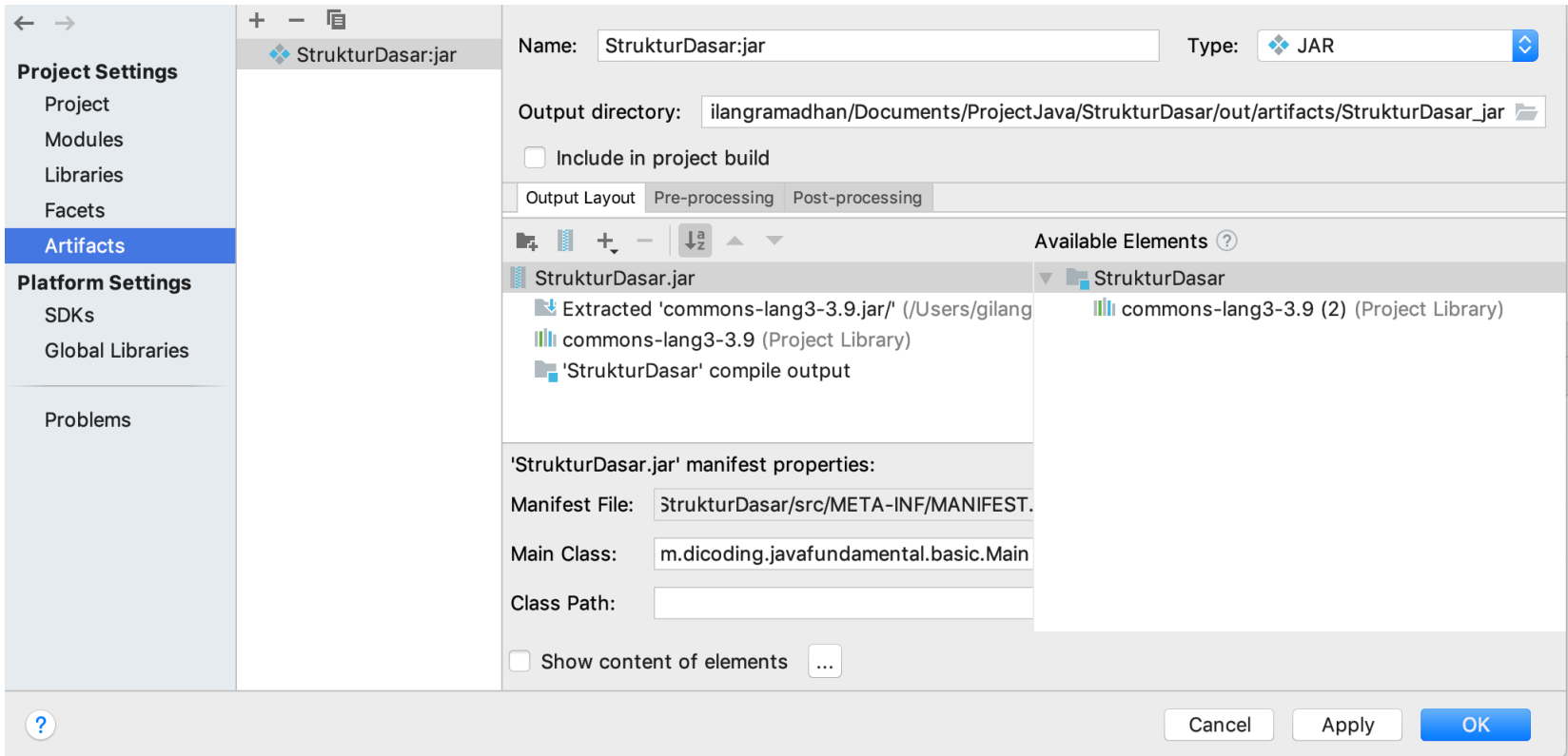
Pilih Main Class, klik tombol atau lalu ketik **Main** di popup. Kemudia klik OK untuk melanjutkan.



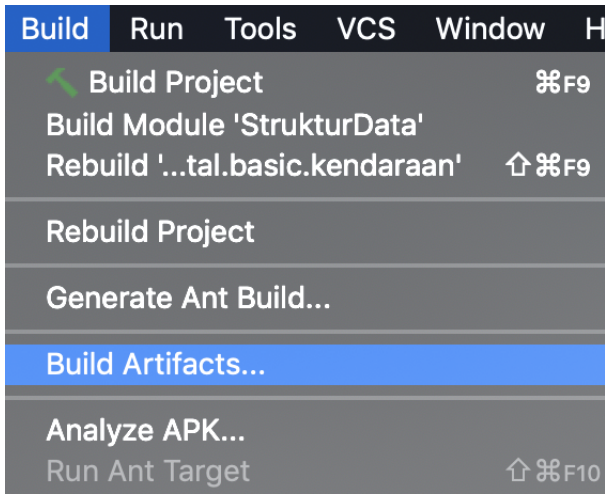
Maka akan jadi seperti gambar di bawah. Klik OK untuk melanjutkan.



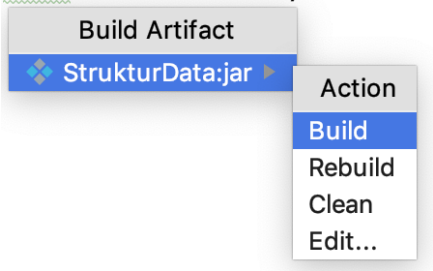
Hasil akhirnya akan menjadi seperti gambar di bawah. Klik OK untuk melanjutkan.



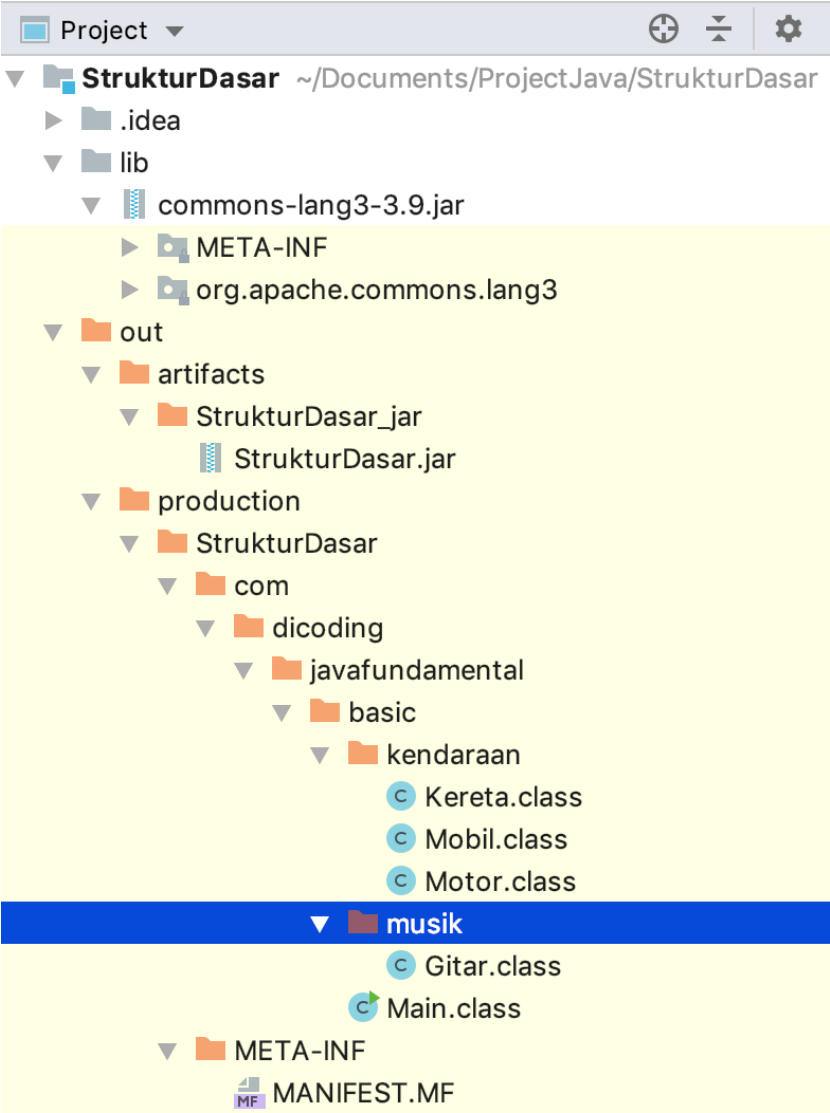
2. Selanjutnya pilihlah menu Build dan pilih Build Artifacts.



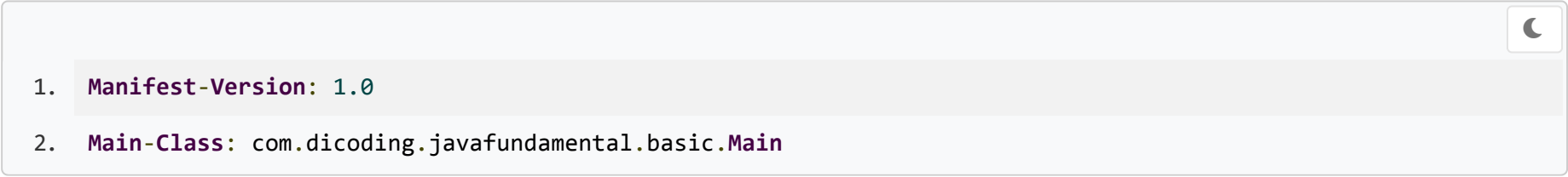
Pilihlah Build untuk memulai membangun Artifacts.



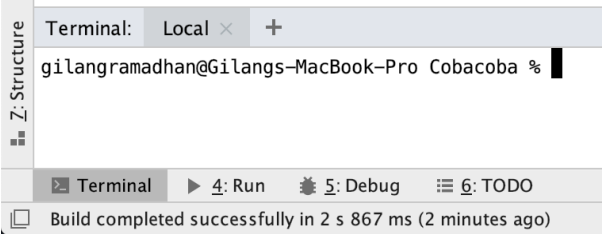
Perhatikan, akan terbentuk **StrukturDasar.jar** di *directory artifacts* yang isinya sama persis seperti *directory production*.



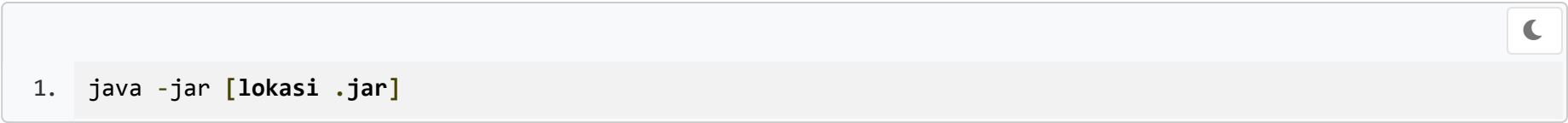
3. Perhatikan dan buka berkas **MANIFEST.MF** yang dibuat otomatis oleh proses Build.



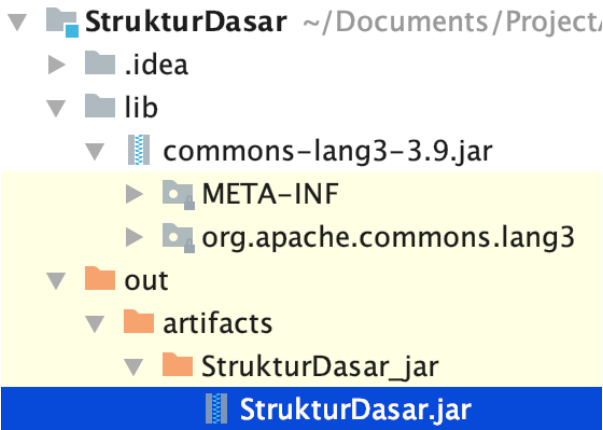
4. Buka panel Terminal di bagian bawah IDE (sesuaikan dengan versi IDE Anda):



Kemudian ketikkan command atau perintah berikut dan lihat hasilnya.



Untuk lokasi .jar, Anda dapat melihat letak berkas .jar yang sudah Anda generate. Contohnya seperti ini:





Maka dapat disimpulkan lokasinya adalah out/artifacts/StrukturDasar\_jar/StrukturDasar.jar.

Maka hasilnya akan jadi seperti ini:

```
Hello word!!
jrenggg..
Ban mobil 4
Ban motor 2
Ban kereta banyak!
Hari ini = Tue Jul 16 18:12:44 WIB 2019
Besok = Wed Jul 17 18:12:44 WIB 2019
```

```
Terminal: Local x +
Gilangs-MacBook-Pro:StrukturDasar gilangramadhan$ java -jar out/artifacts/StrukturDasar_jar/StrukturDasar.jar
Hello world!
jrenggg..
Ban mobil 4
Ban motor 2
Ban kereta banyak!
Hari ini = Tue Jul 16 18:15:27 WIB 2019
Besok = Wed Jul 17 18:15:27 WIB 2019
Gilangs-MacBook-Pro:StrukturDasar gilangramadhan$
```

Selesai sudah materi struktur dasar. Sampai ketemu di materi selanjutnya, *happy coding!*

Jika Anda mengalami kesulitan, silakan unduh *source code*-nya di [Source Code Struktur Dasar](#).

[← Sebelumnya](#)

[Selanjutnya →](#)