PACKAGE, INTERFACE, DAN EXCEPTION HANDLING

OBJEKTIF :

1. Mahasiswa Mampu Memahami *Package*, *Interface*, dan *Exception Handling* pada Java.
2. Mahasiswa Mampu Menggunakan *Software* IntelliJ IDEA dalam Pembuatan Program Terkait *Package*, *Interface*, dan *Exception Handling* dengan Bahasa Pemrograman Java.

11.1 DEKLARASI DAN IMPLEMENTASI PACKAGE

Java menyediakan sebuah bentuk untuk pengelompokan kelas-kelas dan *interface* yang berkaitan, bentuk tersebut adalah *Package*. *Package* merupakan direktori yang menyimpan *class* dan *interface* yang memiliki kemiripan fungsi. Dengan kata lain, *package* adalah sebuah folder yang berisi sekumpulan program Java. Sintaks untuk pendeklarasian *package* adalah sebagai berikut:

```
package namaPackage;
```

Jika sebuah *class* dihilangkan kalimat *package* nya, maka *class* tersebut akan diletakkan pada *package default* tanpa nama. Namun, untuk aplikasi yang besar dan kompleks, hanya menggunakan *package default* saja tidak cukup. Nama *package* mengikuti nama domain dari sebuah vendor yang mengeluarkan program tersebut. Pada contoh di bawah ini, com.integratedlaboratory; merupakan nama domain dari iLab.

```
package com.integratedlaboratory.program;
```

Membuat Package

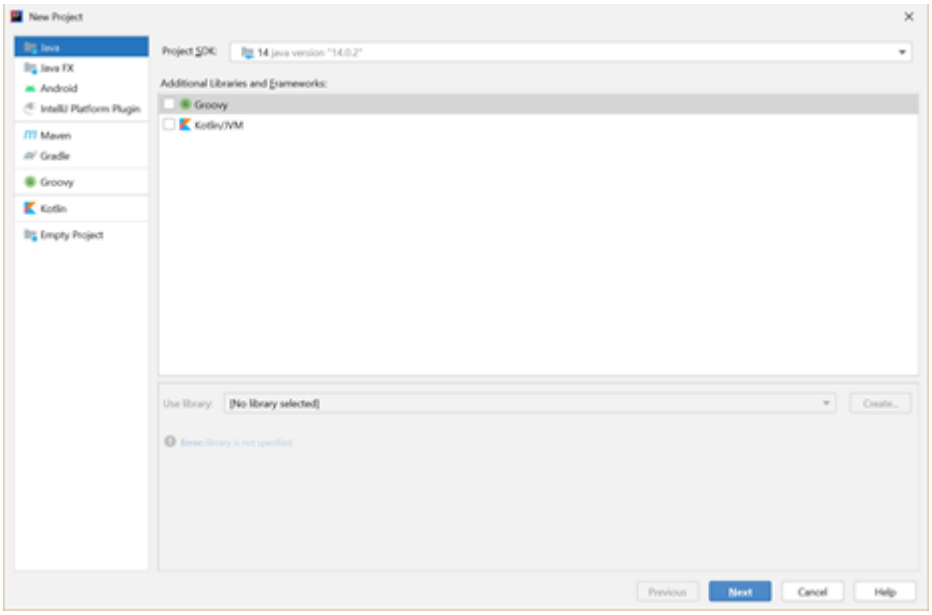
Pernyataan *package* harus diletakkan pada baris pertama *file* sumber. Pernyataan *package* hanya ada satu pada setiap *file* sumber dan berlaku untuk seluruh jenis *file*. Pada contoh berikut ini, merupakan sebuah *class* yang terdapat pada *package* dengan domain `com.integratedlaboratory.mypackage;`.

```
package com.integratedlaboratory.mypackage;

public class Sample {
   public void demo() {
      System.out.println("Ini adalah contoh method dalam class");
   }
   public static void main(String args[]) {
      System.out.println("Hello World!");
   }
}
```

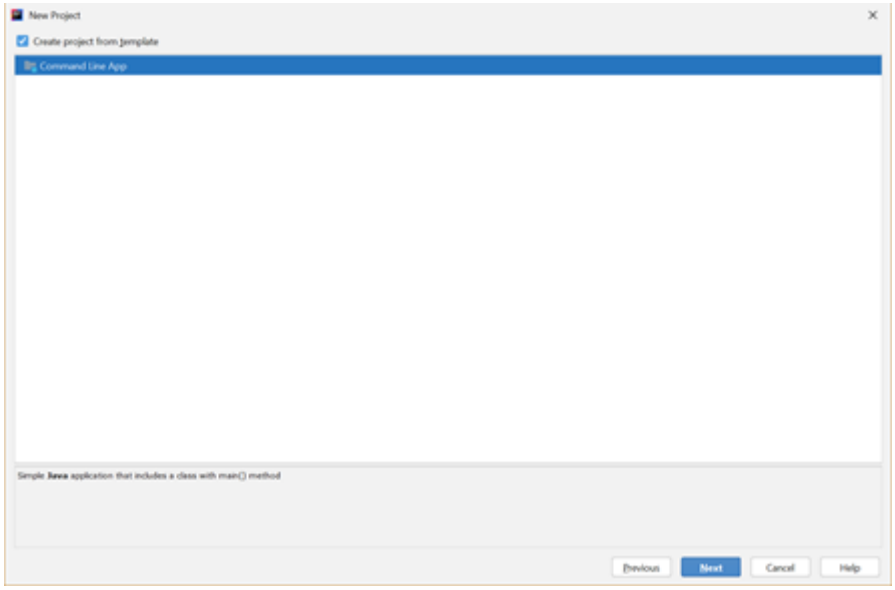
Implementasi *Package* pada IntelliJ IDEA

Berikut ini merupakan contoh implementasi *package* pada IntelliJ IDEA. Pada IntelliJ IDEA, untuk membuat sebuah program, diharuskan membuat *project* terlebih dahulu. Buka IntelliJ IDEA lalu akan muncul tampilan berikut, lalu pilih “Create New Project”.

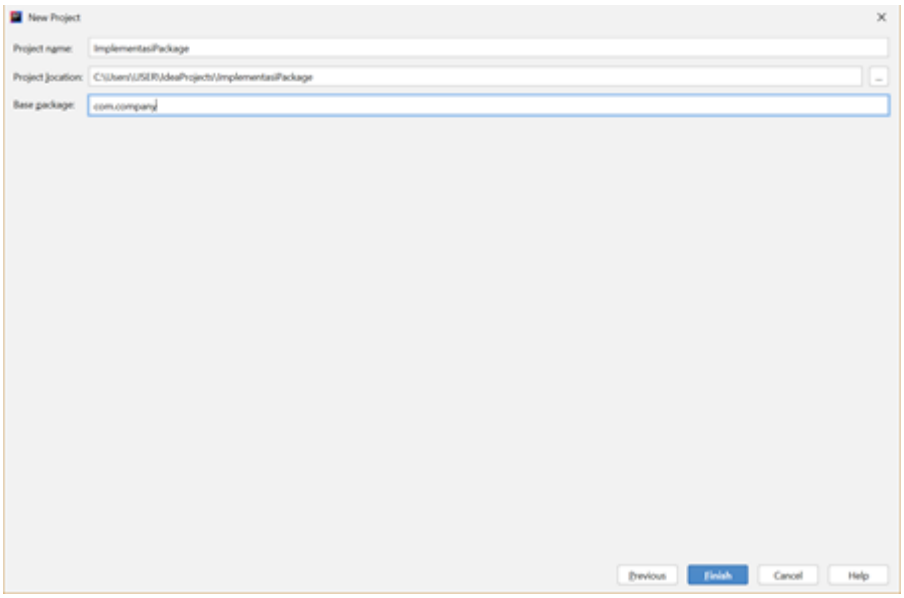
Selanjutnya, pastikan bahwa sudah terdapat Java SDK dan memilih *project* Java.



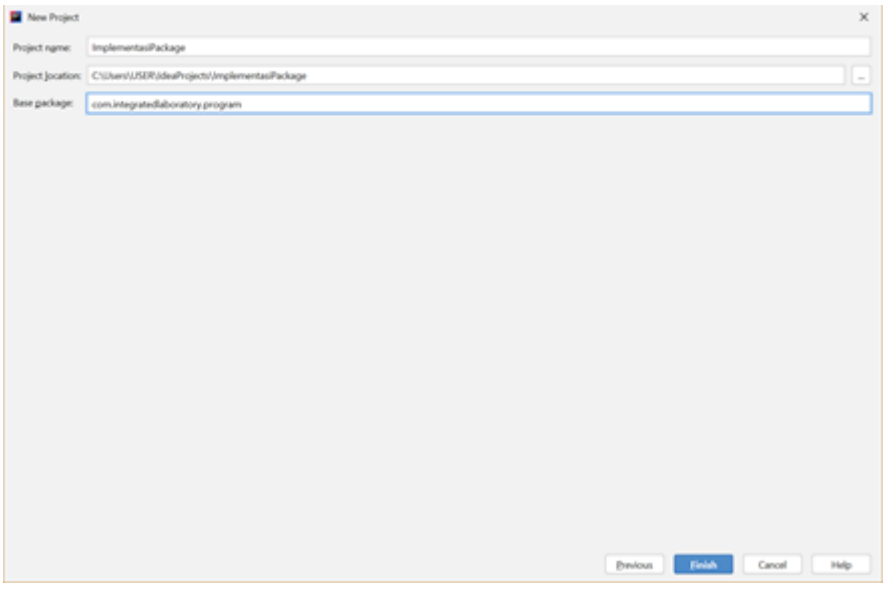
Pada IntelliJ IDEA, jika membuat *project* dengan mengaktifkan *template project*, maka akan otomatis membuat *file* utama untuk *project* tersebut.



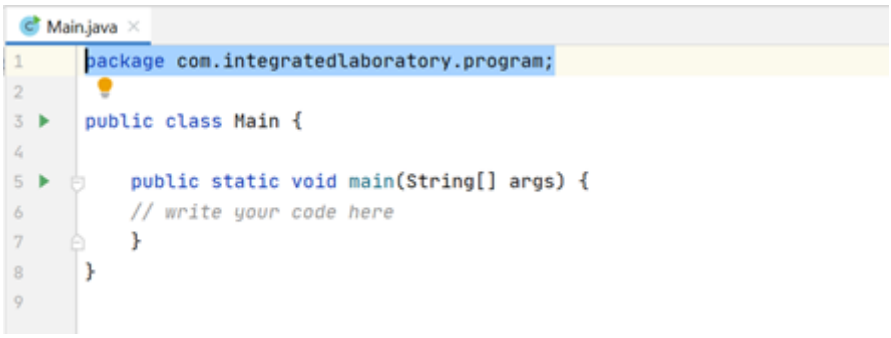
Tahap selanjutnya adalah tahap dimana konfigurasi *project* dilakukan. Seperti pemberian nama *project*, direktori penyimpanan, dan nama *package* diberikan. IntelliJ IDEA memberikan *default package* com.company, maka seluruh *file* yang terdapat pada program tersebut akan memiliki *package* com.company.



Untuk memberikan domain *package* sesuai vendor, maka ubah domain *package* yang terdapat pada kolom “Base package”.



Setelah membuat *project* dengan mengisikan domain sesuai vendor, maka setiap *file* pada *project* tersebut akan otomatis memiliki *package* dengan domain yang diinginkan.

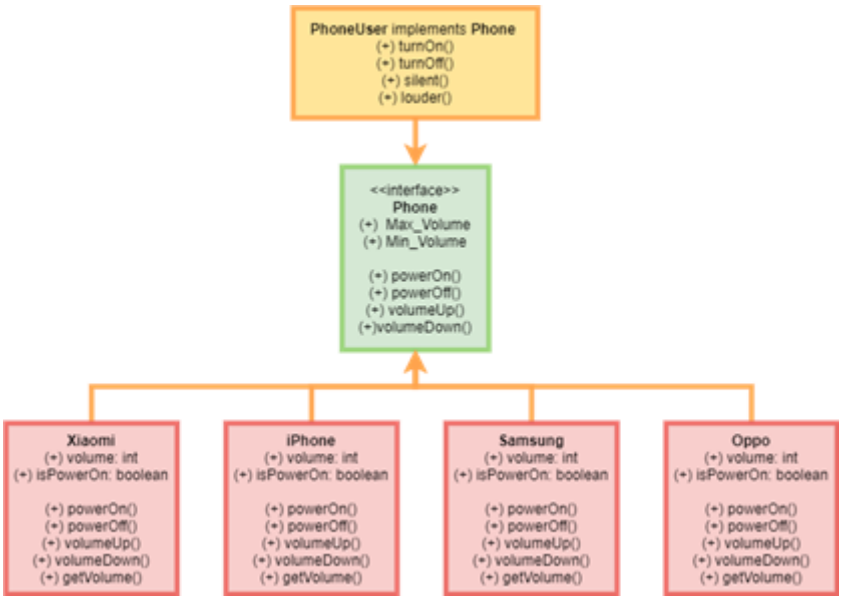


11.2 DEKLARASI DAN IMPLEMENTASI INTERFACE

Pada Java, *Interface* merupakan sebuah prototipe untuk *class* yang berfungsi untuk meninjau rancangan logika atau *method* spesifik dari program yang akan diolah. *Interface* merupakan penghubung antar objek yang serupa dengan menurunkan sebuah kelas. Sebuah *interface* berisi konstanta dan *method*. *Method* yang terdapat pada *interface* berupa *public* dan *abstract*. Mendeklarasikan *interface*, serupa dengan mendeklarasikan *class*. *Class* menggambarkan atribut dan perilaku suatu objek, sedangkan *interface* berisi perilaku yang dari *class* yang diimplementasikannya. Sintaks untuk menciptakan *interface* adalah sebagai berikut:

```
public interface nama_interface {
   // isi dengan method static atau final maupun abstract
}
```

Berikut ini merupakan ilustrasi dari sebuah *interface*:



Sebuah *handphone* memiliki beberapa fitur yang pasti ada, seperti menyalakan, mematikan, mengatur besar kecilnya suara *handphone*, dan sebagainya. Namun, berbeda merk *handphone* implementasi nya pun berbeda-beda. Pada gambar di atas, **PhoneUser** yang merupakan pengguna *handphone* memiliki *handphone* dengan fitur turnOn(), turnOff(), silent(), dan louder(). **Phone** sebagai *interface* menghubungkan pengguna tersebut ke beberapa merk *handphone* yang memiliki fitur yang terdapat pada **PhoneUser**.

Implementasi *Interface* pada IntelliJ IDEA

Setelah membuat *project* pada IntelliJ IDEA, maka kita akan membuat *file interface* baru dengan nama Phone.

```
package com.integratedlaboratory.program;

public interface Phone {
    public void setNama(String namaPemilik);
    public String getNama();
    public String getStatus();
}
```

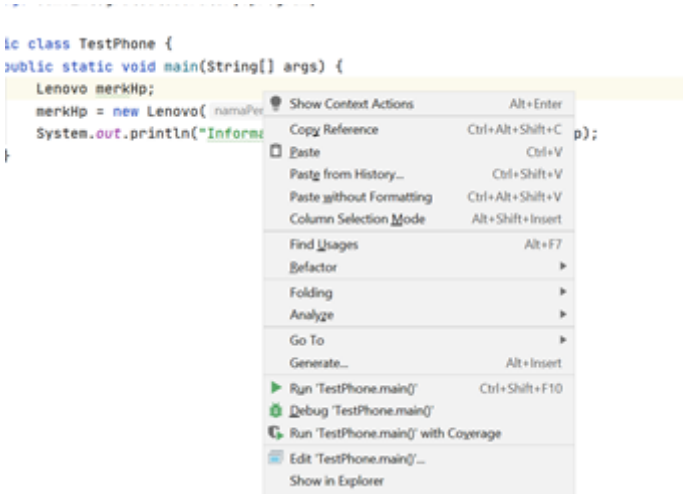
Setelah itu buat *class* baru akan mengimplementasikan *interface* Phone dengan nama Lenovo.

```
package com.integratedlaboratory.program;

public class Lenovo implements Phone{
    String nama;
    String status = "Menyala";

    public Lenovo(String namaPemilik){
        nama = namaPemilik;
    }

    @Override
    public void setNama(String namaPemilik) {
        nama = namaPemilik;
    }

    @Override
    public String getNama() {
        return nama;
    }

    @Override
    public String getStatus() {
        return status;
    }

    public String toString(){
        return
                "Nama : "+nama+" \n" + "Status : "+status;
    }
}
```

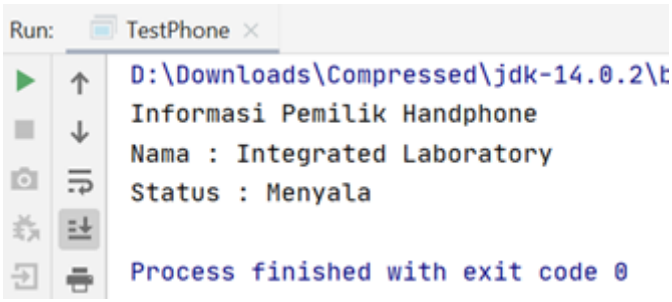
Terakhir, membuat *class* utama dengan nama TestPhone.java.

```
package com.integratedlaboratory.program;

public class TestPhone {
    public static void main(String[] args) {
        Lenovo merkHp;
        merkHp = new Lenovo("Integrated Laboratory");
        System.out.println("Informasi Pemilik Handphone" + "\n" + merkHp);
    }
}
```

Lalu jalankan program pada Main.java dengan tekan Ctrl+Shift+F10 atau dengan melakukan klik kanan pada Main.java seperti berikut:



Hasil program:



11.3 BUG DAN EXCEPTION

Kesalahan merupakan bagian normal dari pemrograman. Pada Java, kesalahan terbagi ke dalam dua jenis, yaitu *bug* dan *exception*. *Bug* merupakan kesalahan yang terjadi akibat kelemahan dari perancangan atau implementasi. *Exception* merupakan kesalahan yang dapat ditangkap serta ditangani yang disebabkan oleh kondisi sistem atau lingkungan seperti memori habis atau nama *file* yang tidak sah dan kondisi *abnormal* yang muncul saat kode sedang dijalankan yang terjadi ketika *run-time*.

Kesalahan yang terjadi karena *exception*, akan ditentukan apakah kesalahan ini dapat berubah menjadi *bug*. *Exception* merupakan kondisi pengecualian dan sesuatu yang di luar biasanya. *Exception* akan menangkap kesalahan yang terjadi serta cara menanganinya. Pada Java, *Exception* merupakan *subclass* dari java.lang.Throwable. Kelas Throwable akan mengirimkan *string* berupa pesan yang akan mendeskripsikan kesalahan yang disebabkan oleh *exception*. Java menangani

exception dengan melibatkan kata kunci *try*, *catch*, *throw* dan *throws*. Berikut ini merupakan contoh dari penangkapan *exception*:

```
public class CobaException {
    public static void main(String args[]) {
        int num[] = {1, 2, 3, 4};
        System.out.println(num[5]);
    }
}
```

Pada kode di atas, *array* dideklarasikan dengan ukuran 4 dan mencoba mengakses elemen ke 5 *array* sehingga menyebabkan sebuah `ArrayIndexOutOfBoundsExceptionexception` terjadi. *Output* yang tampil sebagai berikut:

11.4 BLOK TRY DAN CATCH

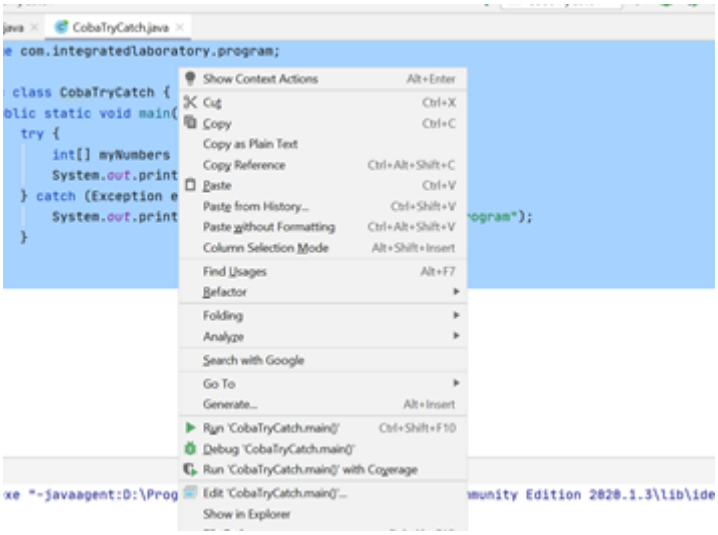
Untuk menanggapi *exception*, digunakan sebuah blok yaitu *try* dan *catch*. Blok *try* dan *catch* berpasangan. Blok *try* digunakan untuk menyimpan detail *exception* ke dalam sebuah *stack* dan akan dilanjutkan oleh blok *catch*. Blok *catch* akan melakukan penanganan *exception*. Sintaks umum dari blok *try* dan *catch*:

```
try {
  //Pemanggilan metode yang dihasilkan exception
}
catch(Exception e) {
  //Penanganan terhadap exception
}
```

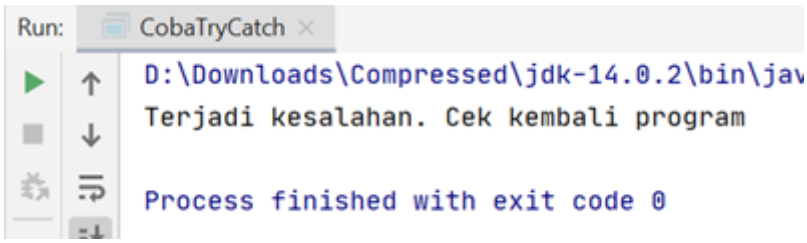
Perhatikan contoh berikut:

```
package com.integratedlaboratory.program;

public class CobaTryCatch {
    public static void main(String[ ] args) {
        try {
            int[] myNumbers = {1, 2, 3};
            System.out.println(myNumbers[10]);
        } catch (Exception e) {
            System.out.println("Terjadi kesalahan. Cek kembali program");
        }
    }
}
```

Jalankan program dengan tekan Ctrl+Shift+F10 atau dengan mengklik kanan pada program seperti berikut dan pilih Run:



Hasil program:



Pada contoh di atas, pemanggilan System.out.println() pada blok *try* tidak dieksekusi. Begitu *exception* dilempar, kendali program ditransfer keluar blok *try* dan ditangani oleh blok *catch*.

11.5 THROW DAN THROWS

Throw merupakan pelemparan kesalahan atau *bug* yang *exception*nya telah ditentukan sebelumnya (*predefined*) dan secara *manual* oleh *programmer* (*user-defined*). Sintaks bentuk umum *throw*:

```
throw ThrowableInstance;
```

`ThrowableInstance` merupakan sebuah objek dengan tipe `Throwable` atau *subclass* dari `Throwable`. Terdapat dua cara untuk mendeklarasikan objek `Throwable`, yaitu:

1. Menggunakan parameter pada blok *catch*.

```
package com.integratedlaboratory.program;

public class ContohExceptionThrow {
        public static void main(String[] args) {
            try {
                throw new Exception ("Kesalahan Terjadi");
            }catch (Exception e){
                System.out.println(e);
            }
        }
}
```

2. Menciptakan salah satu dengan menggunakan operator `new()`.

```
public class CobaException {
        public static void main(String[] args) {
            System.out.println("Hello");
            NullPointerException nullPointer = new NullPointerException();
            throw nullPointer;
        }
}
```

Throws merupakan pendeklarasian *exception* yang biasa digunakan pada saat penggunaan blok *try* dan *catch*. *Throws* digunakan dalam sebuah method atau kelas yang memiliki kemungkinan menghasilkan suatu kesalahan sehingga perlu ditangkap kesalahannya. Sintaks bentuk umum *throws*:

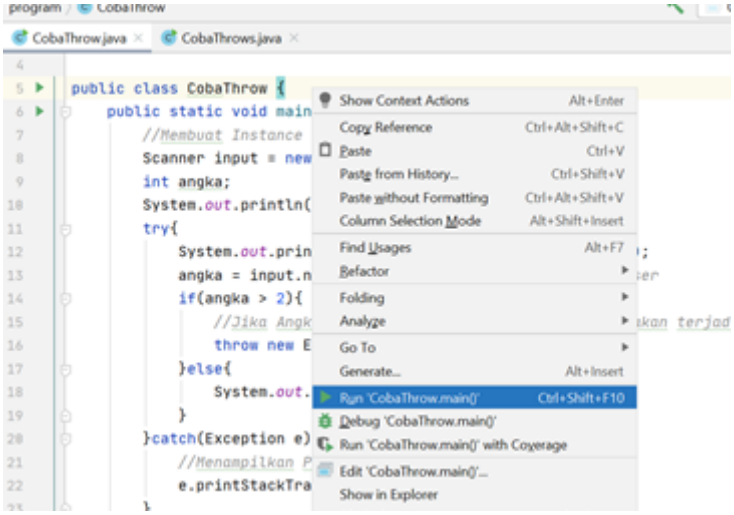
```
(modifier) nama_method() throws list_exception {
       //badan method
}
```

`list_exception` adalah daftar *exception* yang dipisahkan dengan koma, yang berisi daftar *exception* yang dapat dilempar oleh method.

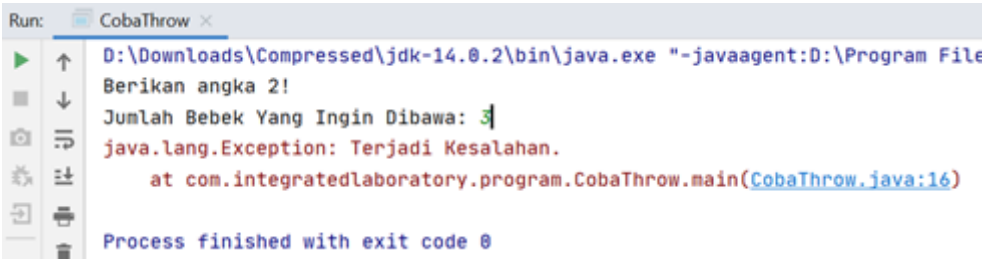
Berikut ini merupakan contoh program dari penggunaan *throw* dan *throws*:

```
package com.integratedlaboratory.program;

import java.util.Scanner;

public class CobaThrow {
    public static void main(String[] args){
        Scanner input = new Scanner(System.in);
        int angka;
        System.out.println("Berikan angka 2!");
        try{
            System.out.print("Jumlah Bebek Yang Ingin Dibawa: ");
            angka = input.nextInt();
            if(angka > 2){
                throw new Exception("Terjadi Kesalahan.");
            }else{
                System.out.println("Perfect");
            }
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

Jalankan program dengan tekan Ctrl+Shift+F10 atau dengan mengklik kanan pada program seperti berikut dan pilih Run:



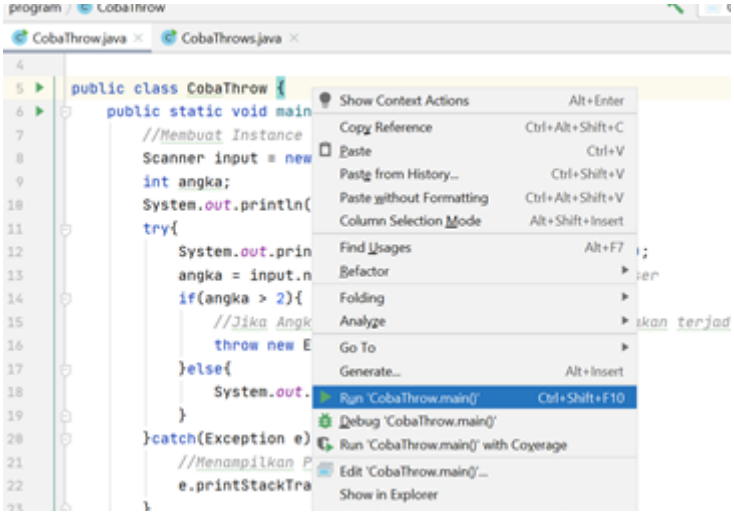
Hasil program:



Terlihat bahwa pada program di atas *exception* telah ditentukan sebelumnya. Selanjutnya adalah program dengan *throws*.

```
package com.integratedlaboratory.program;

class KelasAsing{
    static void strange() throws ClassNotFoundException{
        System.out.println("Terjadi Kesalahan.");
        throw new ClassNotFoundException("Kesalahan sudah ditangkap.");
    }
}

public class CobaThrows {
    public static void main(String[] args){
        try{
            KelasAsing.strange();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

Jalankan program dengan tekan Ctrl+Shift+F10 atau dengan mengklik kanan pada program seperti berikut dan pilih Run:



Hasil program:

REFERENSI:

[1] Hariyanto, Bambang. 2017. *Esensi-Esensi Bahasa Pemrograman Java Revisi Kelima*. Bandung: Informatika.

[2] Muhardian, Ahmad. 2018. "Belajar Java: Memahami Struktur dan Aturan Penulisan Sintaks Java", https://www.petanikode.com/java-sintaks/. (1 Agustus 2020)

[3] URL: https://www.tutorialspoint.com/java/java_packages.htm, diakses pada 1 Agustus 2020.

[4] Muhardian, Ahmad. 2019. "Tutorial Java OOP: Memahami Interface di Java (dan Contohnya)", https://www.petanikode.com/java-oop-interface/, diakses pada 1 Agustus 2020.

[5] URL: https://www.tutorialspoint.com/java/java_interfaces.htm, diakses pada 1 Agustus 2020.

[6] URL: https://www.tutorialspoint.com/java/java_exceptions.htm, diakses pada 3 Agustus 2020.

[7] URL: https://www.w3schools.com/java/java_try_catch.asp, diakses pada 3 Agustus 2020.

[8] URL: https://www.tutorialspoint.com/throw-and-throws-in-java, diakses pada 3 Agustus 2020.

[9] Athoillah, Wildan M. 2017. "Tutorial Exception Handling (throw & throws) pada Java", https://www.wildantechnoart.net/2017/11/exception-handling-throw-throws-java.html, diakses pada 3 Agustus 2020.