BINAR

# Java Exception Handling

**Silver** - Chapter 3 - Topic 1

BINAR

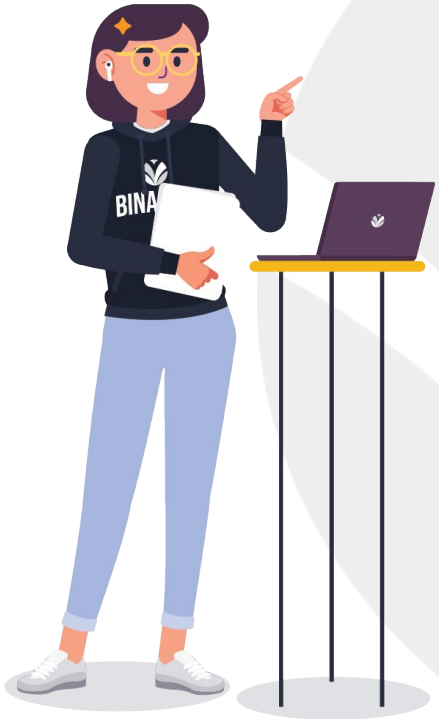
# Selamat datang di Chapter 3 Topic 1 online course Back End java dari Binar Academy!

## Selamat datang di chapter baru!

Wah.. kamu udah memasuki chapter tiga aja, nih.

Setelah mempelajari penerapan algoritma programming dan OOP, chapter 3 akan mengajak kamu untuk **memodifikasi OOP dan Style Functional dengan menerapkan Unit Testing secara tepat**.

Eitss, sebelum kesitu, pada topik pertama ini, kita bahas **Java Exception Handling** dulu, ya. Let's go!



## Dari sesi ini, kita bakal bahas hal-hal berikut:

- Konsep Error pada Java
- Pengantar Exceptions
- Penanganan Error dengan Try-catch-block
- Bentuk Try-catch-finally pada Try-catch-block
- Keyword Throw

**Error** adalah situasi dimana kamu menemukan ada kesalahan pada sistem.

Pada Java kita juga berkemungkinan besar menghadapi error.

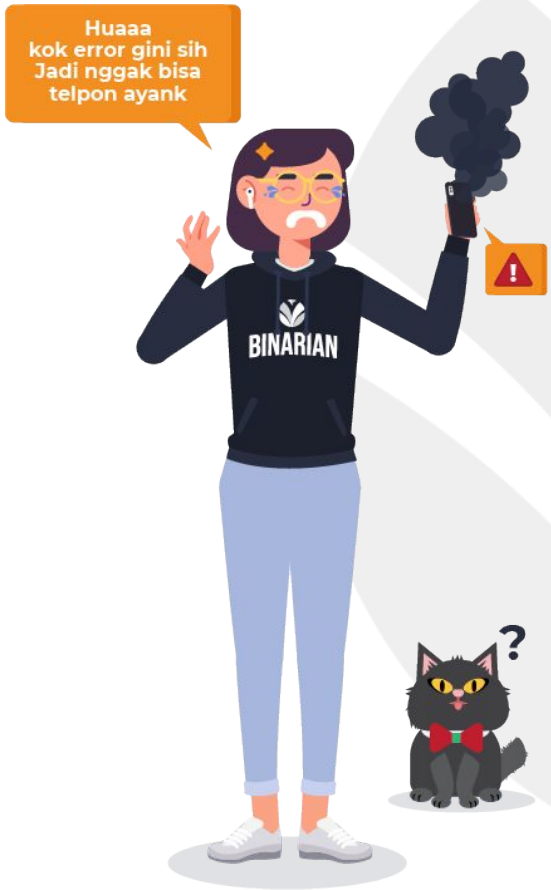
Pertanyaannya, apakah konsep error yang biasa kita temui akan sama dengan konsep di Java? Cari tahu, yuk!

## Error itu apa sih?

Sebelum bahas tentang konsepnya, sekarang kita ambil contoh dulu.

Coba deh, Smartphone kamu pernah mengalami kondisi hang, nggak? Iya, yang tiba-tiba nggak bisa disentuh atau bahkan nggak bisa menyala sama sekali?

Kejadian tersebut terjadi karena smartphone kamu ada di kondisi error atau kesalahan dari sistem yang nggak sesuai sama ekspektasi.



BINAR

## Jadi...

Dari contoh tadi kita bisa menyimpulkan, bahwa error adalah kesalahan pada program yang **membuat program berjalan nggak sesuai ekspektasi atau malah bikin aplikasi jadi nggak berjalan sama sekali**.

Masalahnya, Error menjadi salah satu kondisi yang cukup bikin developer pusing.

Bahkan error bisa terjadi cuma karena developer lupa untuk memberikan semicolon (;) atau ada salah satu huruf yang nggak sesuai sama nama class aja.

Haduhh~

Kalau kamu ingat pada Chapter 1, kita udah bahas tentang dua jenis error:

- **runtime error** yaitu error yang terjadi ketika aplikasi berjalan, dan.
- **compilation error** yaitu error yang terjadi pas aplikasi di-compile.

Karena kita udah sempet mention error di Chapter 1 dan mempelajari OOP di chapter 2, akhirnya pada chapter ini kita bakal menggabungkan keduanya dengan membahas lebih dalam lagi tentang error.

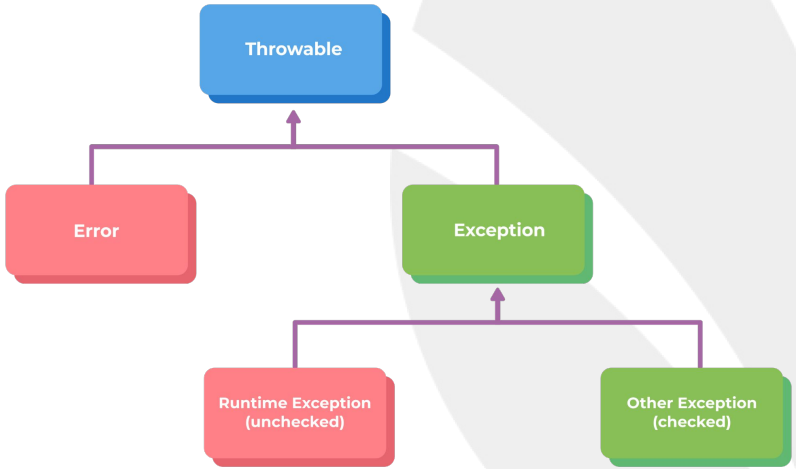
BINAR

## Berikut adalah bagan dari Error pada Java!

Error di dalam Java merupakan sebuah **subclass dari Class Throwable.** Throwable sendiri merupakan **superclass dari dari exceptions dan errors.**

Object throwable dibentuk ketika terjadi sesuatu yang **unexpected atau yang nggak terduga**.

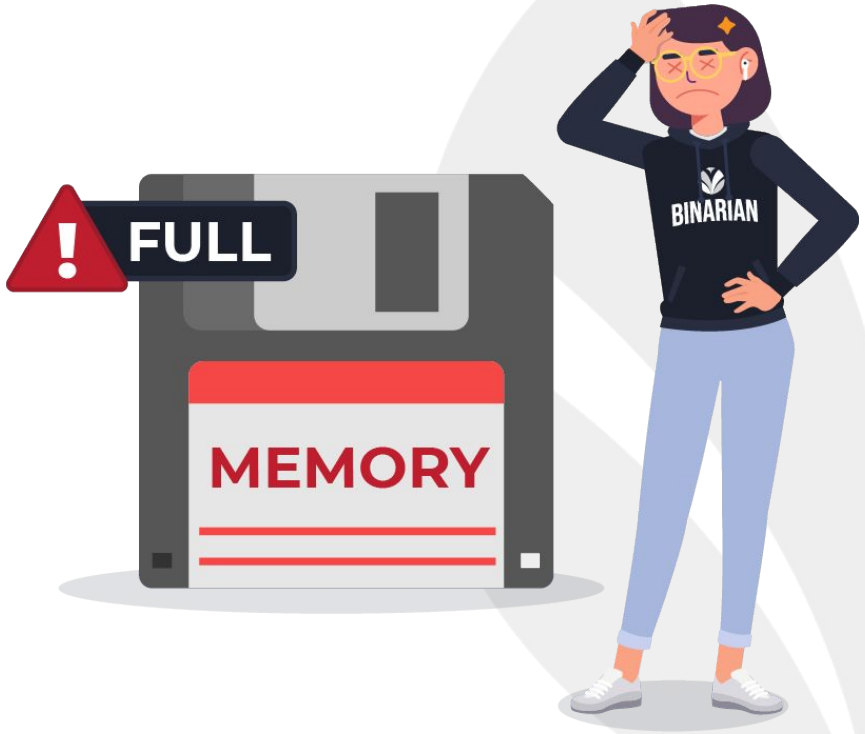
Walaupun unexpected, developers bisa tahu letak error-nya di mana dan di bagian apa, karena ada bantuan dari exception handling ini.



# Java Exception Handling

Beberapa error yang cukup dikenal pada Java yaitu:

- **StackOverFlowError**, error yang disebabkan oleh recursion yang terjadi tanpa henti.
- **OutOfMemoryError,** error yang disebabkan oleh penggunaan memory yang terlalu banyak dan Garbage Collectors nggak sanggup untuk menanganinya.



Kalau kamu ingin tahu penjelasan lebih lanjut mengenai macam-macam error pada Java, bisa di cek pada link di bawah ini ya!

**Java error**

Konsep error kayaknya udah mulai kebayang nih, sekarang kita masuk ke pembahasan tentang **Exceptions.**

Exceptions ini mirip kayak error, tapi tetep ada bedanya.

## Exception itu sebenarnya apa sih?

Exception adalah **kondisi unexpected yang terjadi pada program di Java.** Tapi, nggak sampe menyebabkan berhentinya suatu program alias masih bisa di-recovery.

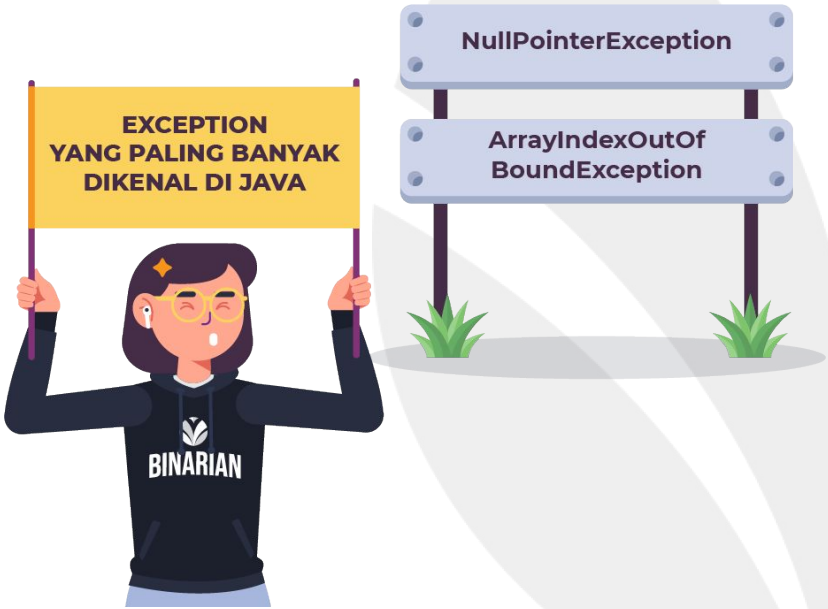
Ada masalah tapi ada solusi juga, cukup menarik, kan? Eh yang bikin makin menarik lagi, kita bisa membuat custom exception pada program kita.

Kita bisa bikin custom exception untuk program kita dengan cara meng-extend class exception yang udah ada.

Hal ini kadang dibutuhkan untuk membuat exception yang berkaitan dengan business logic dari program yang dibuat.

Berikut adalah exception yang udah ada dan paling banyak dikenal di Java yaitu:

- **NullPointerException**, yaitu exception yang terjadi pas method dipanggil sama object yang null.
- **ArrayIndexOutOfBoundException**, yaitu pas array index yang dipanggil lebih dari size array yang ada.



Untuk contoh Exceptions seperti apa, kamu bisa cek pada gambar di bawah ini ya!

```
public class IncorrectFileExtensionException  extends
RuntimeException {
    public IncorrectFileExtensionException(String
errorMessage, Throwable err) {
        super(errorMessage, err);
    }
}
```

BINAR

## Terus, bedanya Error dengan Exceptions itu apa?

Nice question! Ada 3 perbedaan antara Error dengan Exception dilihat dari cara kerja dan kondisinya. Coba kamu lihat dulu deh yang ada dibawah ini.

Yep, kamu pasti menunggu penjelasan secara lebih detail, kan? Sekarang ayo kita analisis beberapa istilah yang muncul pada tabel sebelumnya.

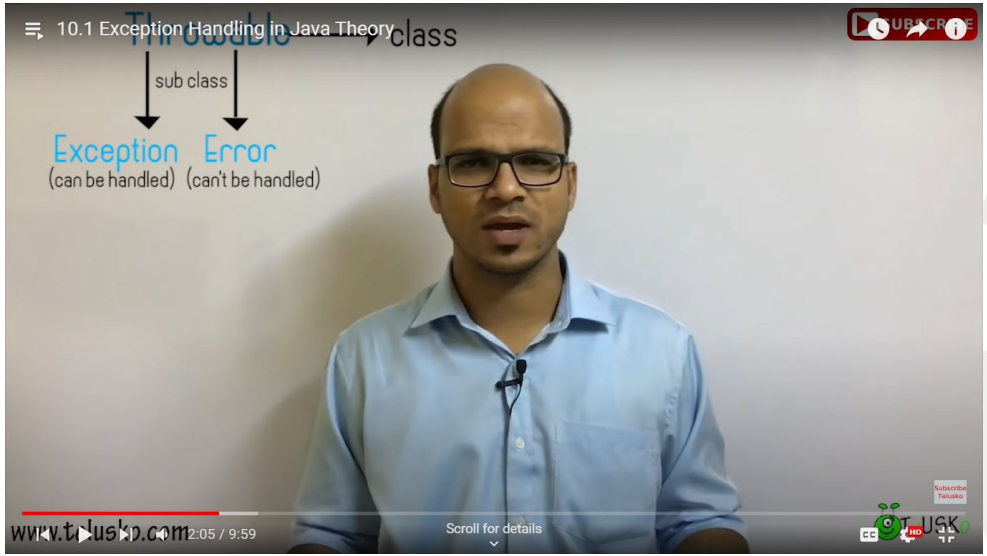
- **Checked** artinya bisa dideteksi oleh compiler ketika proses compilation berjalan.

  Sedangkan **unchecked** artinya nggak bisa dideteksi sama compiler. Sehingga, unchecked nggak bakal muncul pas compilation.

- **Error tidak dapat di-recovery,** artinya kalau error terjadi pas aplikasi berjalan, error bakal tetap muncul dan nggak bisa diperbaiki kecuali dengan me-restart aplikasi.

Kalau kita balik lagi nih ke poin perbedaan error dengan exception, kita bisa ambil kesimpulan bahwa Error pada Java merupakan kondisi **unexpected yang serius** dan nggak bisa dihandle aplikasi Java.

Sedangkan kalau exception itu kondisi unexpected yang masih bisa di handle.




Tunggu dulu, sebelum kita move on ke pembahasan selanjutnya, ada referensi video tentang teori exception handling pada Java. Yuk, buruan diklik!

**Exception Handling in Java Theory**

Okey, sekarang kita zoom lebih detail ya.

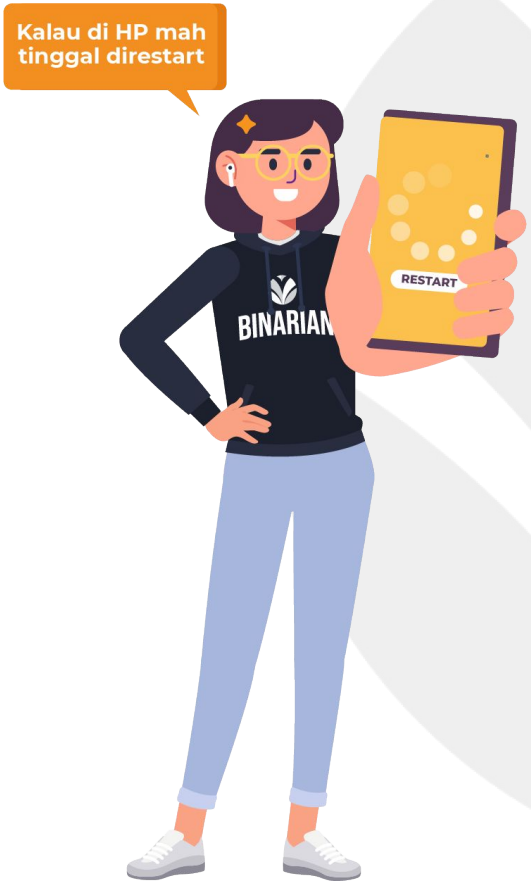
Setelah memahami konsep error dan exceptions pada situasi unexpected, sekarang kita mulai menyelami materi tentang Try-catch block.

BINAR

## Beda program, beda penanganan Error ya, gengs~

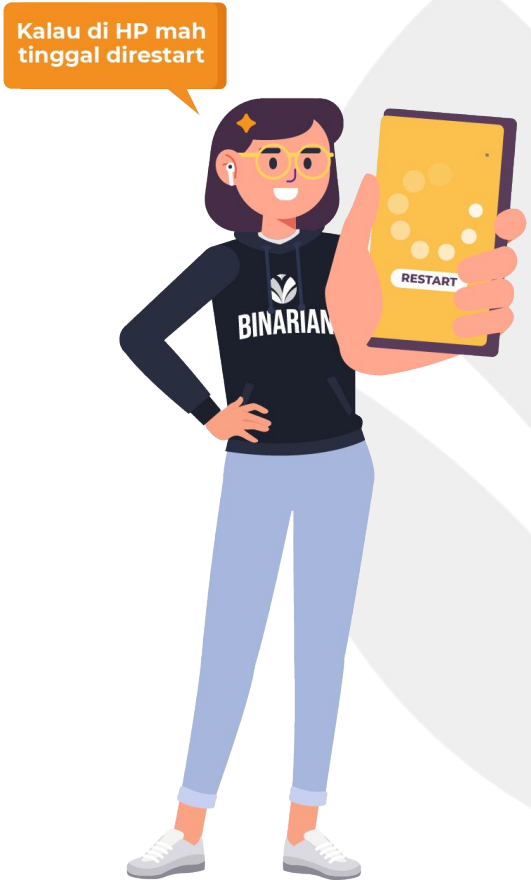
Adanya error pada program dapat menyebabkan program berhenti atau hang.

**Dari situ diperlukan mekanisme untuk memastikan bahwa program tetap bisa berjalan meskipun ada kesalahan**.



Gini deh, semisal smartphone kamu lagi nge-hang, apa yang bakal kamu lakukan?

Salah satu caranya adalah dengan melakukan restart pada smartphone yang kamu pakai~



## Kalau di Java, penanganannya kayak gimana, dong?

Cara penanganan error di Java ini berbeda sama penanganan error di smartphone kamu, bestie.

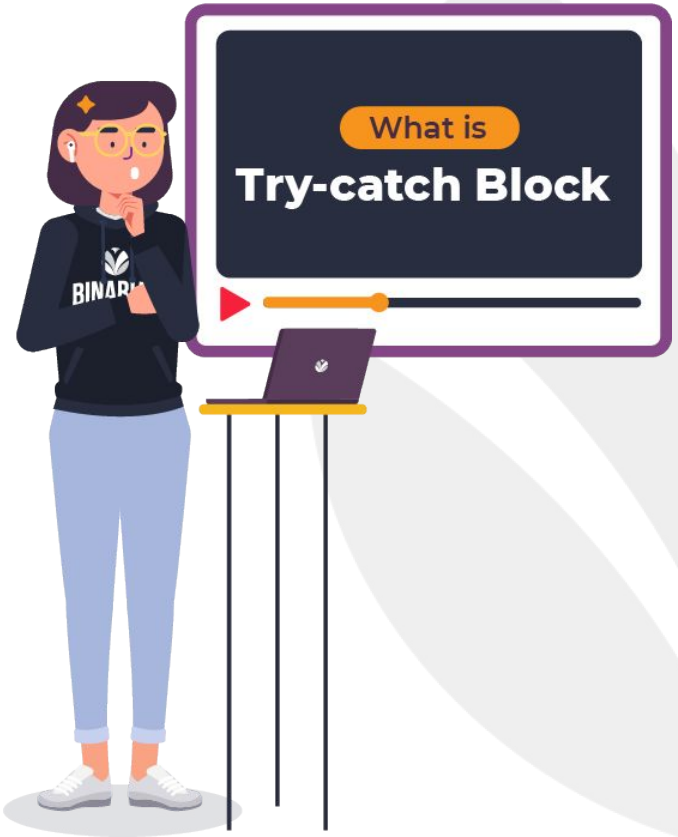
Untuk menangani error di Java, **try-catch** merupakan **dua block of code yang dipakai untuk menghandle error.**

## Emangnya Try-catch Block itu apa?

**Try block memungkinkan kita untuk tidak melanjutkan eksekusi code yang ada ketika terjadi error di try block.**

Catch block bakal dieksekusi pas terjadi error di try block. Jadi, kalau seluruh statement di try block bisa dieksekusi tanpa terjadinya error, statement yang ada di catch block nggak dieksekusi sama sekali.

Oh iya, ada beberapa bentuk lain dari try-catch, yaitu **try-catch-finally** dan **try with resource.**



BINAR

## Biar lebih mantap, kita lihat contoh di samping, ya!

Pada try block 50/0 akan dihasilkan sebuah exception, jadi code pada catch block bakal dijalankan.

Walaupun begitu, statement di bawah 50/0 yang terjadi exception nggak bakal tereksekusi karena exception udah terjadi dan cuma catch block yang bakal dieksekusi.

```
try
        {
            int data=50/0;
    System.out.println(data);
        }
catch(ArithmeticException e)
        {
            System.out.println(e);
        }
```

Beda dengan contoh sebelumnya, pada contoh ini nggak ada statement di try-block yang menghasilkan suatu exception, jadi semua code yang ada di try block bisa dieksekusi.

Pada case ini catch block nggak dieksekusi sama sekali, yaaa~

```
try
        {
            int data=50/10;
            System.out.println(data);
        }
catch(ArithmeticException e)
        {
            System.out.println(e);
        }
```

Karena sebelumnya udah disinggung duluan tentang dua bentuk dari Try Catch Block.

Sekarang kita langsung cari tahu lebih detail tentang bentuk Try-catch Finally. dan Try-catch Resources.

## Apa itu Try-catch Finally?

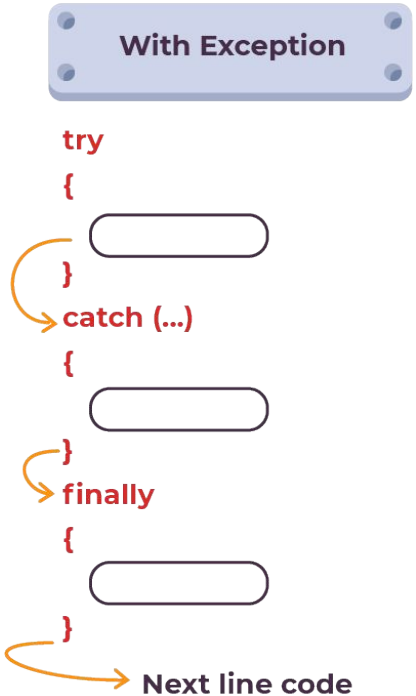
Try-catch Finally merupakan bentuk yang sama dengan try catch, bedanya ada satu tambahan block of code lagi yaitu, finally.

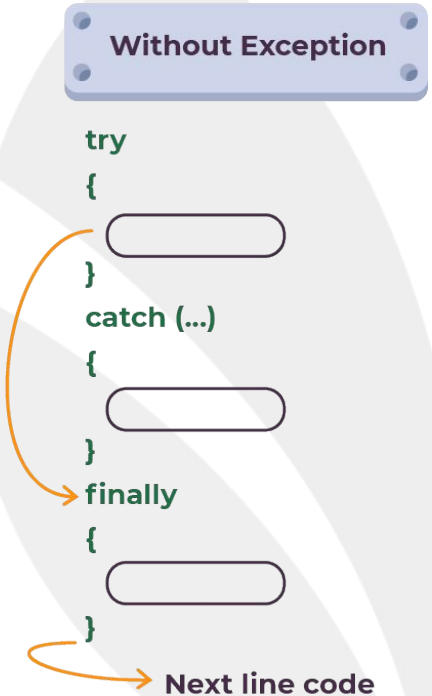
**Finally block ini adalah block yang bakal dieksekusi setelah try-catch block dijalankan.**

BINAR

Kalau ada code yang mengalami exception di try block, maka catch block bakal dieksekusi. Setelah catch block dieksekusi, maka lanjut finally block yang dieksekusi.

Kalau try block berhasil dieksekusi tanpa adanya exception, maka finally block bisa langsung dieksekusi.



## Oke, biar ada gambaran, kita pake contoh ya!

Misalnya, rumah Sabrina lagi mati lampu, waktu di cek ternyata lampunya harus diganti.

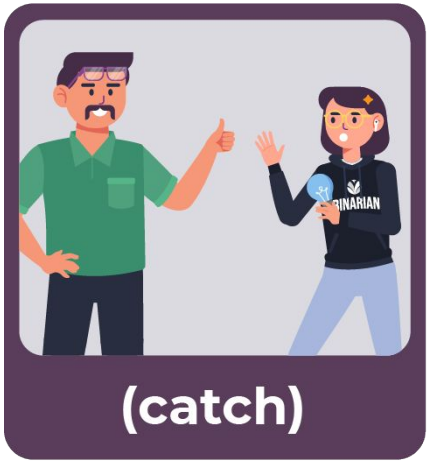
Untuk mengganti lampu tersebut, Sabrina harus berusaha **(try)** untuk memasang lampunya dengan menaiki sebuah kursi.

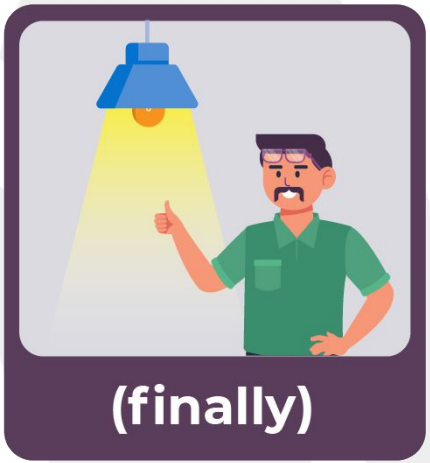
Tapi, ketika proses pemasangan lampu berjalan, ada kemungkinan yang bakal terjadi. Yaitu, bisa aja Sabrina jatuh dari kursi atau malah bisa terkena setrum pas lagi memasang lampunya.

Tentunya kemungkinan tersebut bisa diatasi dengan berbagai cara **(catch),** misalnya Sabrina minta tolong sama Mas Gun untuk memasang lampu tersebut.

Sampai akhirnya **(finally)** rumah Sabrina jadi terang lagi karena lampunya udah dipasang.

Yeayyy~



## Terus, apa sih fungsi dari Try-catch Finally?

Try-catch-finally biasanya dipakai **untuk membuka dan menutup resource**.

Sedangkan finally block berisi statement untuk menutup resource. Resource ini bisa berupa file ataupun koneksi database.

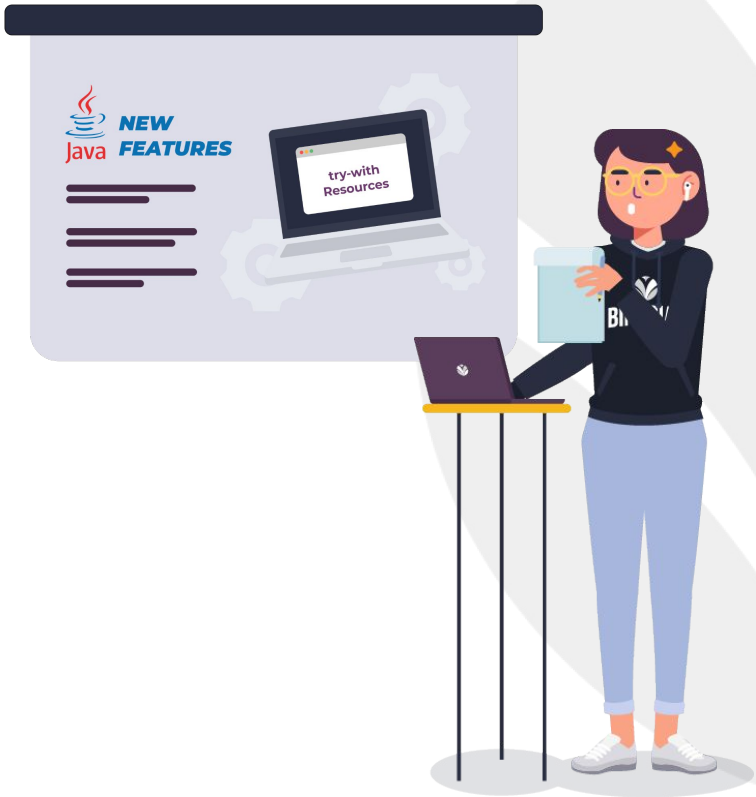
Kalau ada resource yang nggak ditutup, hal tersebut bakal menyebabkan penggunaan memory yang nggak diperlukan. Kan sayang~

## Bentuk pengembangan dari Try-catch Finally~

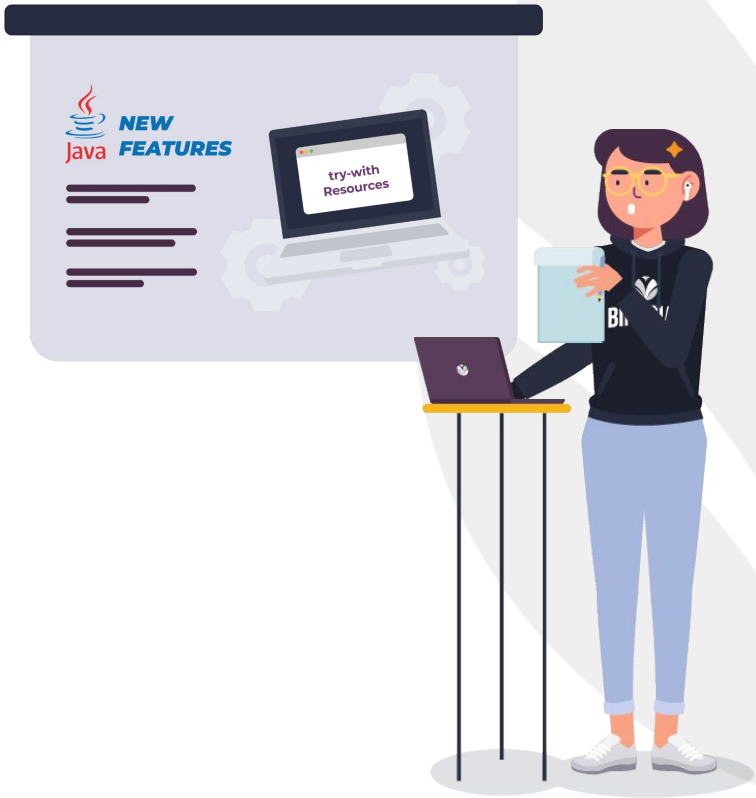
Java 7 menyediakan fitur baru yaitu try-with Resources yang berfungsi untuk mengakses suatu resource.

Apabila proses selesai, resource bakal ditutup secara otomatis.

Dari situ kita jadi tahu, bahwa **Try-with Resources merupakan pengembangan dari Try-catch Finally.**



Oh iya, pada Try-with Resource kita nggak perlu bikin statement untuk menutup resource di finally block karena udah di-handle sama **AutoCloseable Interface**.



BINAR

## Try-catch Finally kalo dipakai kayak gimana, ya?

Berikut contoh penggunaan dari Try-catch Finally, Sobat!

```
Scanner scanner = null;
try {
    scanner = new Scanner(new File("test.txt"));
    while (scanner.hasNext()) {
        System.out.println(scanner.nextLine());
    }
} catch (FileNotFoundException e) {
    e.printStackTrace();
} finally {
    if (scanner != null) {
        scanner.close();
    }
}
```

Kalau yang ini adalah contoh dari Try-with Resources~

```
try (Scanner scanner = new Scanner(new
File("test.txt"))) {
    while (scanner.hasNext()) {
        System.out.println(scanner.nextLine());
    }
} catch (FileNotFoundException fnfe) {
    fnfe.printStackTrace();
}
```

Kalau yang ini nih, contoh dari Try-with Resources yang pakai lebih dari 1 resources!

```
try (Scanner scanner = new Scanner(new
File("testRead.txt"));
    PrintWriter writer = new PrintWriter(new
File("testWrite.txt"))) {
    while (scanner.hasNext()) {
    writer.print(scanner.nextLine());
    }
}
```

Sipp! Try Catch Block yang kedua, yaitu Try Catch Resources udah dijabarin.

Selanjutnya kita bakal bahas tentang **Throw.**

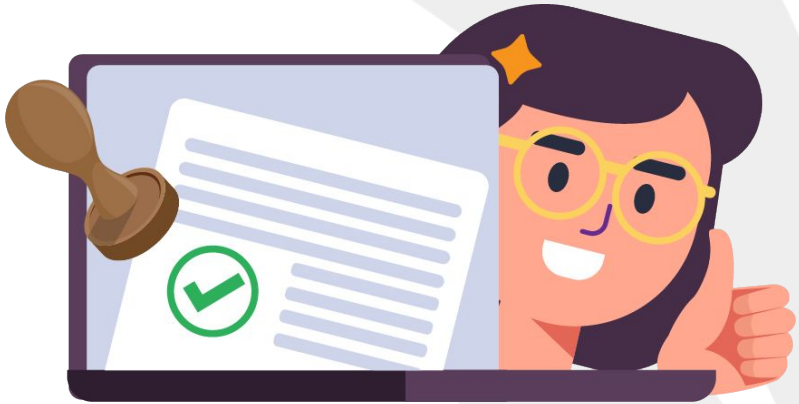
Spoilers, antara Throw dan Throws itu bakalan beda, lho. So, hati-hati ya~

## Throw itu apa sih?

**Throw adalah keyword yang dipakai untuk membentuk object yang merupakan exception untuk membuat suatu validasi.**

Validasi ini biasanya dipakai dalam mengatur suatu business flow supaya program berjalan dengan sesuai.

Ada dua keyword yang berkaitan dengan throw nih, yaitu **throw** dan **throws.**



BINAR

## Throw

Berikut adalah penggunaan throw dari sebuah custom exception dengan Class dari:

NullRequestException!

Kalau kita lihat pada gambar, Constructor dari NullRequestException punya parameter berupa message.

```
public class NullRequestException
 extends RuntimeException {
    public NullRequestException(String
errorMessage) {
        super(errorMessage);
    }
}
```

Kalau yang ini contoh saat melakukan throw dengan NullRequestException yang sudah dibuat pada contoh sebelumnya

```
If(request == null){
    throw new
NullRequestException("Request cannot be
null");
}
```

Validasi kalau request null, maka bakal throw NullRequestException dengan message **“Request cannot be null”**

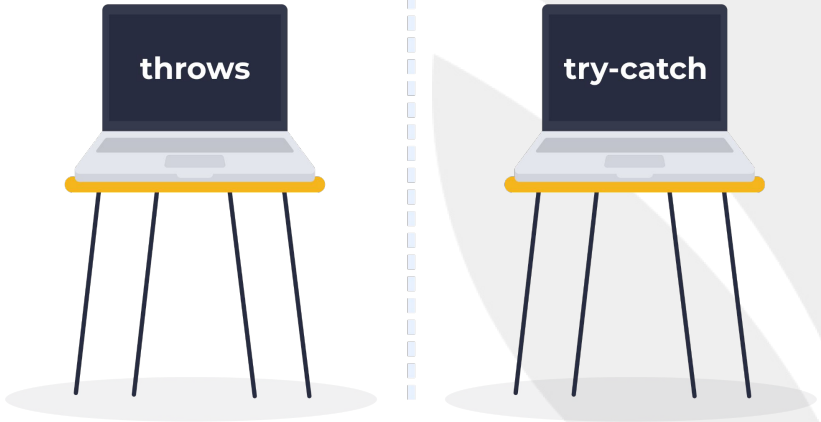

## Throws

Cara menggunakan keyword throws ini adalah dengan menuliskan keyword throws dan jenis exception-nya di sebelah parameter method pada saat sebuah method dideklarasikan.

Fungsi dari penggunaan throws ini mirip kayak try-catch, yaitu kalau ada exception terjadi di dalam code tersebut, maka blok di catch bakal di eksekusi.

Sedangkan pada throws, kalau terjadi exceptions, maka method bakal secara langsung melakukan throw exception.

Tapi, pada penggunaan throws, nggak ada block of code yang dieksekusi seperti pada try catch block.



Method *-yang memanggil method yang melakukan throws-* harus melakukan throws juga, atau pada saat memanggil method yang melakukan throws, method tersebut harus berada di dalam try-catch block

Method yang memiliki throws keyword bisa melakukan throw lebih dari satu jenis exception.

Berikut adalah contoh penggunaan dari throws:

```
public static void writeToFile(String message) throws IOException,
IllegalArgumentException {
    if(message == null) throw new IllegalArgumentException(“message cannot
be null”);
    BufferedWriter bw = new BufferedWriter(new FileWriter("myFile.txt"));
    bw.write(message);
    bw.close();
}
public static void write() throws IOException, IllegalArgumentException {
 writeToFile(“Test”);
}
public static void main(String[] args) {
try {
        write();
    } catch (IOException | IllegalArgumentException e) {
        e.printStackTrace();
    }
}
```

Method writeToFile dapat melakukan throws pada dua jenis exception, yaitu:

- **IOException,** dan.
- **IllegalArgumentException**.

IOException dapat terjadi ketika proses penulisan terjadi exception.

Sedangkan IllegalArgumentException terjadi kalau request gagal pas melakukan validasi.

Oh iya ada lagi nih, method write yang memanggil writeToFile perlu menerapkan throws IOException dan IllegalArgumentException.

Ketika memanggil method write bisa pakai try-catch block.

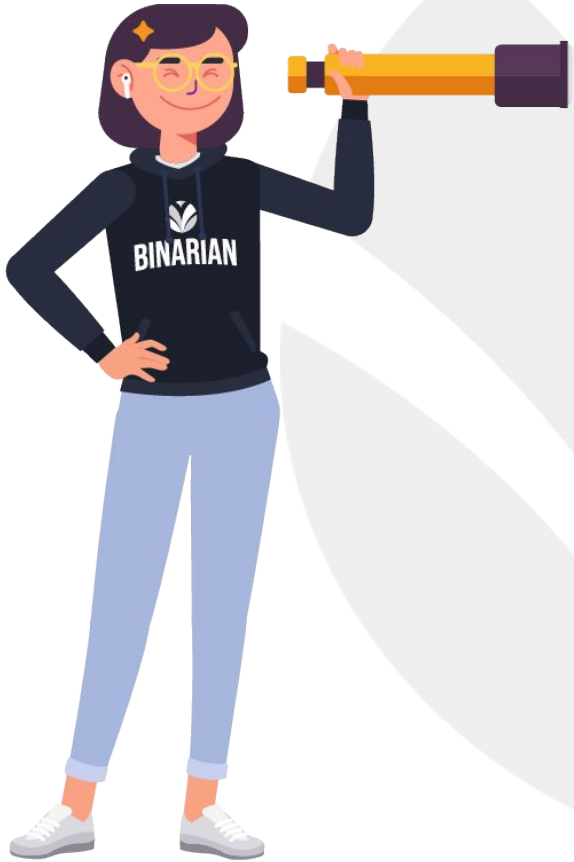
Pada catch block, bisa melakukan catch IOException dan IllegalArgumentException atau bisa juga pakai Exception aja yang lebih general.

## Coba diingat lagi Challenge Chapter sebelumnya yang kamu kerjakan~

Kamu masih ingat challenge di Chapter 2 yang udah kamu kerjakan?

Yap betul! Kamu diminta untuk membuat aplikasi pengolah nilai siswa di console untuk mengolah dan generate file.

Nah, mengacu dari challenge tersebut, validasi apa aja yang diperlukan supaya exception nggak terjadi?



## Bip bip~

Terakhir nih, sebelum kita move on ke topic selanjutnya, yuk **buat project/class dengan implementasi handling exception di project Java**.

Latihan ini dilakukan di kelas dan jangan lupa untuk mendiskusikan hasil jawabannya bersama teman sekelas dan facilitator, ya.

Selamat mencoba~

Keren ya Exception Handling di Java ini! Dengan bantuan Exception Handling, developer ga perlu pusing tujuh keliling karena bingung nyari errornya di mana.

Menurutmu, apakah error dan exception pada Java dapat dihindari? Lalu, bagaimana ya cara menghindarinya?

BINAR

# Saatnya Quiz

# 1

Keyword berikut yang tidak digunakan untuk melakukan exception handling, yaitu....

A. Catch

B. Final

C. Finally

1

**B. Final**

Keyword catch digunakan sebagai untuk menangkap error sedangkan finally merupakan keyword untuk mengeksekusi block setelah try ataupun catch dieksekusi.

# 2

Pernyataan yang salah di bawah ini yaitu....

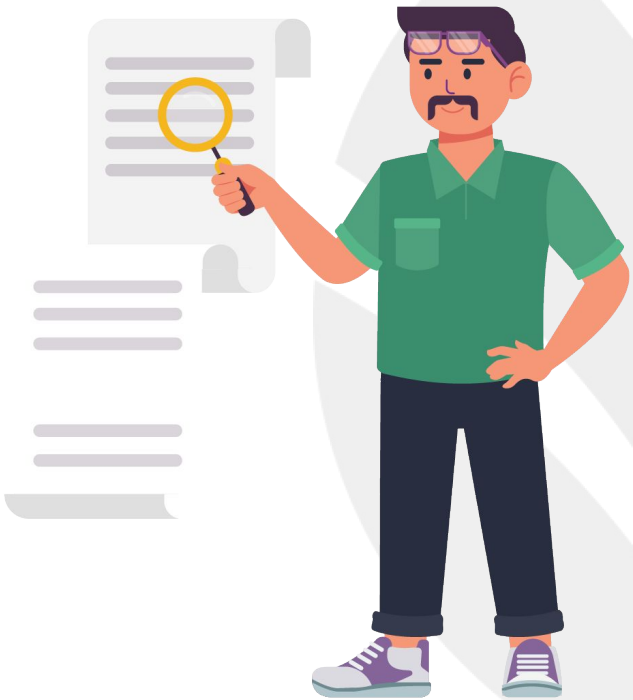
A. Apabila seluruh code pada block try berhasil dieksekusi tanpa error, catch block tidak akan dieksekusi

B. Sebuah field yang bernilai null pasti menyebabkan NullPointerexception

C. Keyword throws bisa digunakan untuk melempar suatu exception pada level method

## Referensi


1. Exception Handling in Java Practical Part 1 Try Catch
2. Exception Handling in Java Practical Part 2 Try with Multiple Catch Unchecked
3. Exception Handling in Java Practical Part 3 Checked
4. Exception Handling in Java Practical Part 4 Finally Block
5. Exception Handling in Java Practical Part 5 Throw and Throws
6. Exception Handling in Java Practical Part 6 User Defined Exception

Nah, selesai sudah pembahasan kita pada Chapter 3 Topic 1 ini.

Selanjutnya, kita bakal ngobrolin tentang dunia tes mengetes program pada topik **Unit Testing Part 1.**

Penasaran kayak gimana? Yuk, langsung ke topik selanjutnya~