

## Exception

Exception มี 3 ประเภท

1. Exception extends มาจาก Throwable (Checked)

Ex. IOException, ParseException, SQLException

2. RuntimeException extends มาจาก Exception

Ex. IOException, ParseException, SQLException

3. Error extends มาจาก Throwable เป็น Exception ที่ไม่ปกติที่โปรแกรมไม่ควรจัดการ

Ex. AssertionError, IOError, LinkageError, VirtualMachineError

## Try-Catch Block

Syntax :

```
try {  
    // Code that may throw an exception  
} catch (Exception e) {  
    // Do something with the exception using  
    // reference e  
}
```

**try** -> จะ throw เมื่อโปรแกรมมีข้อผิดพลาด

**catch** -> จัดการกับ exception ซึ่งจะมีการ define ประเภทของ exception และสามารถอ้างอิงถึง exception นั้นได้

ตัวอย่างโปรแกรมที่ไม่ได้ใส่ try catch

```
class Test {  
    public static void main(String[] args) {  
        int[] arr = new int[3];  
        for(int i = 0; i <= arr.length; i++) {  
            arr[i] = i * 2;  
        }  
        System.out.println("Done");  
    }  
}
```

เมื่อรันโปรแกรมจะเกิดอะไรขึ้น ???

Answer : loop รอบสุดท้าย i จะเป็น 3 แต่ได้กำหนด array ขนาดเท่ากับ 3 ไว้เลยทำให้เกิด exception ตอน runtime

Output : Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3 at com.example.Test.main(TestException.java:8)

ถ้า exception ไม่ถูก handle JVM จะมีให้มีการ default exception handle ดังนี้

1. ปรี้นท์รายละเอียดของ exception
2. ปรี้นท์ stack trace (ลำดับชั้นของ method ที่ทำให้เกิด exception)
3. Terminate program

ตัวอย่างโปรแกรมที่ใส่ try catch

```
class Test {  
    public static void main(String[] args) {  
        try {  
            int[] arr = new int[3];  
            for(int i = 0; i <= arr.length; i++) {  
                arr[i] = i * 2;  
            }  
        } catch(ArrayIndexOutOfBoundsException e) {  
            System.out.println("Exception caught");  
        }  
        System.out.println("Done");  
    }  
}
```

Output : Exception caught

Done


Example

```
SimpleDateFormat sdf = new SimpleDateFormat("MM/dd");  
Date date = sdf.parse("01-10"); // Compile-time error  
System.out.println(date);
```

**\*\*signature ของ parse method จะ throws ParseException (Extends from Exception)**

```
public Date parse(String source) throws ParseException
```

“throws” เป็น keyword ที่แสดงว่า exception สามารถ throw, checked exception เท่านั้นที่ require เมื่อประกาศใช้ method parse


```
try {
    SimpleDateFormat sdf = new SimpleDateFormat("MM/dd");
    Date date = sdf.parse("01-10");
    System.out.println(date);
} catch (ParseException e) {  สามารถ catch เป็น superclass ได้
    System.out.println("ParseException caught");
}
```

หรือจะ catch แบบนี้ก็ได้

```
try {
    SimpleDateFormat sdf = new SimpleDateFormat("MM/dd");
    Date date = sdf.parse("01-10");
    System.out.println(date);
} catch (ParseException e) {
    System.out.println("ParseException caught");
} catch (Exception e) {
    System.out.println("Exception caught");
}
```

**\*\*ถ้ามีหลาย catch ห้ามเอา superclass ไปอยู่บนสุดก่อน subclass**

```
try {
    SimpleDateFormat sdf = new SimpleDateFormat("MM/dd");
    Date date = sdf.parse("01-10");
    System.out.println(date);
} catch (Exception e) {
    System.out.println("Exception caught");
} catch (ParseException e) {
    System.out.println("ParseException caught");
}
```

 compile error

**\*\*ใน catch block ถ้ามีการ catch exception ที่ไม่ถูกประเภทจะทำให้ compile error ด้วย**

```
try {
    SimpleDateFormat sdf = new SimpleDateFormat("MM/dd");
    Date date = sdf.parse("01-10");
    System.out.println(date);
} catch (SQLException e) { // Compile-time error
    System.out.println("ParseException caught");
}
```

## Multi- Catch

Syntax :

```
try {
    // Code that may throw one or
    // two exceptions
} catch (Exception1 | Exception2 e) {
    // Do something with the caught
    // exception using reference e
}
```

## Finally

```
try {
    // Code that may throw an
    // exception }
finally {
    // Block that is always executed
}
```

**\*\*catch block เป็น optional จะมีทั้ง catch block หรือ finally block ก็ได้**


finally block จะถูก excute เสมอไม่ว่าจะเกิด exception หรือไม่เกิด แต่ถ้าสั่ง System.exit();

โปรแกรมจะถูก terminate ทันทีและไม่ทำงานใน finally โดยทั่วไปแล้วเคสนี้จะเกิดขึ้นยากเพราะไม่ค่อยทำกัน

## Mutiple-Catch and Finally

เป็นการเอาประเภทของ exception ที่อยู่ในแต่ละ catch block มารวมอยู่ใน catch block เดียวกัน

```
try {
    ...
} catch (ArithmeticException | IndexOutOfBoundsException e) {
    e.printStackTrace();
    return res;
}
```



“|” หมายถึง OR จากรูปไม่ว่าจะ throw `ArithmeticException` หรือ `IndexOutOfBoundsException` ก็จะเข้ามาทำงานใน catch นี้ แต่ถ้าอยากรู้ว่าเป็น exception ประเภทไหนก็สามารถใช้ `instanceof` ตรวจสอบได้

```
try {
    ...
} catch (ArithmeticException | IndexOutOfBoundsException e) {
    if(e instanceof ArithmeticException) {
        // Do something else if the exception type
        // is ArithmeticException
    }
    e.printStackTrace();
    return res;
}
```

**\*\*ตัวแปรที่อยู่ใน multi-catch ถือว่าเป็น final ไม่สามารถเอาไป new ได้**

```
try {
    ...
} catch (ArithmeticException | IndexOutOfBoundsException e) {
    if(e instanceof ArithmeticException) {
        // Compile-time error
        e = new ArithmeticException("My Exception");
    }
} catch (Exception e) {
    e = new Exception("My Exception"); // It compiles!
    throw e;
}
```

**\*\*ไม่สามารถรวม subclass และ superclass ให้อยู่ใน multi-catch block เดียวกันได้**

```
try {
    ...
} catch (ArithmeticException | RuntimeException e) {
    // The above line generates a compile-time error
    // because ArithmeticException is a subclass of
    // RuntimeException
    e.printStackTrace();
    return res;
}
```

## Try-With-Resources

- Resource จะถูกปิดโดยอัตโนมัติหลังจากที่ทำงานใน try block

version ก่อนหน้านี้จะใช้ finally block มาใช้ในการ close resources และตั้งแต่ Java 7 ก็มีการใช้ try-with-resources block มาใช้ในการ close resource โดย 1 resources หรือมากกว่าที่ประกาศใน try block จะถูก close resource โดยไม่ต้องใช้ finally

```
try (AutoCloseableResource r = new AutoCloseableResource()) {  
    // Code that may throw an exception  
} catch (Exception e) {  
    // Handle exception  
} finally {  
    // Always executes  
}
```

### Example

```
try (BufferedReader br =  
    new BufferedReader(new FileReader("/file.txt"))) {  
    int value = 0;  
    while((value = br.read()) != -1) {  
        System.out.println((char)value);  
    }  
} catch (IOException e) {  
    e.printStackTrace();  
}
```

BufferedReader จะปิดเมื่อหลังจากทำใน try block เสร็จซึ่งจะเทียบเท่ากับการ close resource ใน finally block ตามตัวอย่างด้านล่าง

```

BufferedReader br = null;
try {
    int value = 0;
    br = new BufferedReader(new FileReader("/file.txt"));
    while((value = br.read()) != -1) {
        System.out.println((char)value);
    }
} catch (IOException e) {
    e.printStackTrace();
} finally {
    try {
        if (br != null) br.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

ถ้าต้องการประกาศมากกว่า 1 resource ก็ให้คั่นระหว่าง resource ด้วยเครื่องหมาย ;

```

try (FileReader fr = new FileReader("/file.txt");
    BufferedReader br = new BufferedReader(fr)) {
    ...
}

```

**\*\*resource ที่ประกาศใน try-with resources จะไม่สามารถใช้นอก try block ได้เพราะว่า**

1. Out of scope
2. Resource ปิดไปแล้วหลังจากหลุดจาก try block

```

try (BufferedReader br =
    new BufferedReader(new FileReader("/file.txt"))) {
    ...
}
String line = br.readLine(); // Compile-time error

```

ไม่ใช่ทุก class ที่จะใช้ try-with-resource ได้

```

class MyResource {
    void useResource() { }
}
...
try (MyResource r = new MyResource()) { // Compile-time error
    r.useResource()
}

```

ถ้าจะใช้ try-with-resource ต้อง implement

- java.lang.AutoCloseable -> extends Exception

```
void close() throws Exception;
```

- java.lang.Closeable -> extends IOException

```
void close() throws IOException;
```

Example

```
public class TryWithResource implements AutoCloseable{

    public static void main(String[] args) throws Exception {
        try (TryWithResource twr = new TryWithResource()) {
            throw new Exception("test throws exception");
            //throw new RuntimeException("throws RunTime");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void close() throws Exception {
        System.out.println("close before");
    }
}
```

## Custom exceptions

เป็น class ที่เขียนขึ้นมาเองเพื่อจัดการกับ exception ที่เกิดขึ้นแบ่งออกเป็น Checked, Unchecked

- checked extends จาก Exception หรือ subclass
- unchecked extends จาก RuntimeException หรือ subclass

Example: checked



```

public class CustomerException_Checked extends Exception {
    public CustomerException_Checked(String message) {
        super(message);
    }
}

public class Test_Checked {
    public String findByName(String name) throws CustomerException_Checked {
        if ("".equals(name)) {
            throw new CustomerException_Checked("String is empty!"); 2
        }
        return "String not empty";
    }

    public static void main(String[] args) {
        Test_Checked obj = new Test_Checked();
        try {
            String str = obj.findByName(""); 1
        } catch (CustomerException_Checked e) {
            e.printStackTrace(); 3
        }
    }
}

```

Constructure

## Key Points

- Java มี exception 3 ประเภท
  1. java.lang.Exception
  2. java.lang.RuntimeException
  3. java.lang.Error
- RuntimeException และ subclass ไม่จำเป็นต้อง catch (unchecked)
- Exception และ subclass รู้กันว่าเป็น checked exception เพราะ compiler ต้องตรวจสอบว่ามีคำสั่ง try-catch อยู่หรือไม่
- ถ้า exception ถูก catch ได้มากกว่า 1 block, exception จะ catch แค่ block แรกที่ type ตรง
- ลำดับการ catch ห้าม superclass อยู่ก่อน subclass จะทำให้ compile error

- ถ้า code มีการเรียก method ที่มี throws และไม่ได้อยู่ใน try-catch block ตรงส่วนที่เรียกใช้ต้องประกาศ throws ด้วย
- ต่อเนื่องจากข้อก่อนหน้า การเรียกใช้ method จะต้อง catch หรือ ประกาศ throws
- multi-catch block ยอมให้มีมากกว่า 2 exception ที่ไม่เกี่ยวข้องกันใน 1 catch block
- finally block is ALWAYS executed
- ใน try-with-resources block , resource ที่ประกาศไว้จะ close auto หลังจากทำงานใน try-block โดยการ implement AutoCloseable หรือ Closeable
- เมื่อใช้ try-with-resources block, catch และ finally คือ optional เราสามารถสร้าง exception เองโดยการ extends Exception หรือ RuntimeException