Exceptions Handling

Compiled by Kanjana Eiamsaard, Version 1.0.1 2022-03-02

Agenda

- Exception คืออะไร และทำไมต้องจัดการ
- Exception handling ด้วย try...catch
- การโยน exception
- ลำดับของคลาส Exceptions
  - Checked & Unchecked Exception
- Exception handling ด้วย try...catch...finally
- Stack trace & Exception chaining
- Assertion

ความผิดพลาด (Error)

- ความผิดพลาด (Error) คือ การกระทำที่ผิดพลาดของมนุษย์ ซึ่งก่อให้เกิดเหตุการณ์ที่ไม่คาดคิด และสิ่งที่ไม่ต้องการ
- ในขณะที่โปรแกรมกำลังทำงาน **เราป้องกันความเสียหายที่เกิดจากความผิดพลาดได้ (ทำได้ไม่ 100% ก็ควรทำ)**
  - การป้องกันทำได้โดย การตรวจสถานะหรือค่าต่าง ๆ ก่อนดำเนินการ เช่น ตรวจสอบโดเมนของข้อมูลก่อนนำไปใช้ เป็นต้น
- ทางเลือกสำหรับโปรแกรมเมอร์ในการจัดการกับข้อผิดพลาด คือ **จัดการกับข้อผิดพลาดในขณะที่เจอเลยทันที** หรือ **รายงานต่อไปยังโปรแกรมส่วนอื่นให้จัดการแทน**

เอ็กเซปชัน (Exceptions)

- การรายงานข้อผิดพลาดให้โปรแกรมส่วนอื่นจัดการแทน ทำได้ 3 วิธี
  1. การคืนค่าเป็นค่าผลลัพธ์แสดงความผิดพลาด
  2. การกำหนดค่าตัวแปรสถานะบางตัวเพื่อบ่งชี้ว่าเกิดความผิดพลาดขึ้น
  3. การใช้กลไกของเอ็กเซปชัน (exception)
- การรายงานแบบที่ 1 และ 2 หากโปรแกรมเมอร์ลืมทำ อาจนำไปสู่ความผิดพลาดต่อเนื่องจากจุดนั้นได้ หรือ โปรแกรมจะหยุดทำงานพร้อมกับรายงานข้อผิดพลาดทันที
- การรายงานแบบที่ 3
  - Exception คือ สิ่งที่บ่งบอกว่ามีความผิดพลาดเกิดขึ้นใน **ขณะที่โปรแกรม execute**
  - **บังคับ**ให้โค้ดที่รับข้อผิดพลาดต้องจัดการกับข้อผิดพลาด **ไม่สามารถละเลยได้ (Exception handling)**
  - **เมื่อจัดการข้อผิดพลาดแล้ว โปรแกรมสามารถทำงานต่อไปได้ (จุดต่างสำคัญกับแบบที่ 1 2)**

เอ็กเซปชัน (Exceptions) (ต่อ)

- **Exceptions เกิดขึ้นเมื่อใด ?**
  - เมื่อ Java พบข้อผิดพลาด
  - เมื่อไลบรารีที่เรียกใช้พบข้อผิดพลาด
  - โปรแกรมเมอร์เป็นผู้สร้างกลไกการเกิดไว้ด้วยคำสั่ง throw (โยน)
- หาก exceptions เกิดขึ้นแล้ว แต่โปรแกรมเมอร์**ไม่จัดการ จะเกิดอะไรขึ้น ?**
  - จบการทำงานในเมทอดนั้นทันที และ exceptions ถูกส่งไปยังเมทอดผู้เรียกเป็นทอด ๆ จนถึงเมทอดสุดท้าย เช่น main หรือ เมทอดเริ่มต้นในเทรดอื่น
  - หากไม่เจอผู้จัดการข้อผิดพลาดที่ใดในโปรแกรมเลย โปรแกรมจะหยุดการทำงาน และแสดงรายการข้อผิดพลาด (Stack trace) ออกมา
  - Stack trace ประกอบด้วยชื่อ exceptions และรายการเมทอดตั้งแต่จุดที่เกิด exceptions ไปจนถึงเมทอดผู้เรียก

เอ็กเซปชัน (Exceptions) (ต่อ)

- ความผิดพลาดที่นำไปสู่การเกิดเอ็กเซปชันที่พบบ่อยได้แก่
  - การพยายามใช้งานอ็อบเจกต์ด้วยตัวอ้างอิงที่มีค่าเป็น null (NullPointerException)
  - การพยายามแปลงชนิดอ็อบเจกต์โดยที่ชนิดของอ็อบเจกต์นั้นไม่ได้มีความสัมพันธ์แบบ is-a กับชนิดปลายทางที่เราต้องการแปลง (ClassCastException)
  - การอ้างอิงนอกขอบเขตสมาชิกของอาร์เรย์ (ArrayIndexOutOfBoundsException)
  - การส่งอาร์กิวเมนต์ผิดรูปแบบที่เมทอดต้องการ (IllegalArgumentException)

ตัวอย่างโปรแกรมที่ปราศจากการจัดการ exceptions

```java
public class DividedByZeroNoEH {
    public static int quotient(int dividend, int divisor){
        return dividend/divisor;
    }
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int dividend, divisor;
        System.out.print("Enter dividend number: ");
        dividend = input.nextInt();
        System.out.print("Enter divisor number: ");
        divisor = input.nextInt();
        System.out.printf("%d%n",quotient(dividend, divisor));
    }
}
```

```
E:\playground\fund2\week11>java DividedByZeroNoEH
Enter dividend number: 3
Enter dividend number: 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at DividedByZeroNoEH.quotient(DividedByZeroNoEH.java:5)
        at DividedByZeroNoEH.main(DividedByZeroNoEH.java:14)
```

```java
public class DividedByZeroNoEH2 {
    public static int quotient(int dividend, int divisor){
        if(divisor == 0){
            System.out.println("Cannot divided by zero");
            System.exit(99);
        }
        return dividend/divisor;
    }
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int dividend, divisor;
        System.out.print("Enter dividend number: ");
        dividend = input.nextInt();
        System.out.print("Enter divisor number: ");
        divisor = input.nextInt();
        System.out.printf("%d%n",quotient(dividend, divisor));
    }
}
```

```
E:\playground\fund2\week11>java DividedByZeroNoEH2
Enter dividend number: 3
Enter divisor number: 0
Cannot divided by zero
```

- DividedByZeroNoEH โปรแกรมเมอร์ลืมป้องกัน
- DividedByZeroNoEH2 ป้องกันโดยการตรวจข้อมูลก่อนนำไปใช้

ผลลัพธ์ของโปรแกรมที่ปราศจากการจัดการ exceptions

```
E:\playground\fund2\week11>java DividedByZeroNoEH
Enter dividend number: 3
Enter dividend number: 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at DividedByZeroNoEH.quotient(DividedByZeroNoEH.java:5)
        at DividedByZeroNoEH.main(DividedByZeroNoEH.java:14)
```

```
E:\playground\fund2\week11>java DividedByZeroNoEH
Enter dividend number: 4
Enter dividend number: Nodigit
Exception in thread "main" java.util.InputMismatchException
        at java.base/java.util.Scanner.throwFor(Scanner.java:939)
        at java.base/java.util.Scanner.next(Scanner.java:1594)
        at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
        at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
        at DividedByZeroNoEH.main(DividedByZeroNoEH.java:13)
```

- DividedByZeroNoEH ลืมป้องกัน
  - เกิด ArithmeticExeption: /by zero
    - ลำดับของ Stack trace คือ เมทอด quotient -> main
  - เกิด InputMismatchException
    - ลำดับของ Stack trace คือ library Scanner -> main

การจัดการเอ็กเซปชันเบื้องต้น

- Java สร้างเครื่องมือสำหรับจัดการ exceptions ไว้คือ try ... catch
- ขั้นตอนการทำงานปกติและเป็นไปได้ว่าจะเกิดความผิดพลาดให้เขียนไว้ในบล๊อกของ try
  - **หากเกิด exceptions ขึ้น โปรแกรมจะไม่ทำงานในบล๊อก try ต่อ**
- การจัดการเมื่อมี exceptions เกิดขึ้น ให้เขียนไว้ในบล๊อกของ catch
- เมื่อ exceptions มีหลายชนิด catch จึงมีได้หลายบล๊อก ขึ้นอยู่กับบล๊อกนั้นกำหนดให้จัดการกับ exception เรื่องใด
- การกำหนด catch ทำได้หลายวิธี
  - 1 exception / 1 catch
  - หลาย exception / 1 catch
    - ใช้ตัวดำเนินการ OR (|)
    - ใช้ exception ที่เป็น super class
    - exception ทุกชนิดเป็น sub class ของ Throwable

ตัวอย่างการจัดการเอ็กเซปชันเบื้องต้น

```java
public class DividedByZeroWithEH {
    public static int quotient(int dividend, int divisor) {
        return dividend / divisor;
    }

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        boolean isRetried = true;
        int dividend, divisor;
        do {
            try {
                System.out.print("Enter dividend number: ");
                dividend = input.nextInt();
                System.out.print("Enter divisor number: ");
                divisor = input.nextInt();
                System.out.printf("%d%n", quotient(dividend, divisor));
                isRetried = false;
            } catch (InputMismatchException e) {
                System.out.println("You must enter an integer. Please try again!!");
                System.err.println("Exception: " + e.toString());
                input.nextLine(); //cleaning input stream
            } catch (ArithmeticException e) {
                System.out.println("Divisor must be greater than zero. Please try again!!");
                System.err.println("Exception: " + e.toString());
            }
        } while (isRetried);
    }
}
```

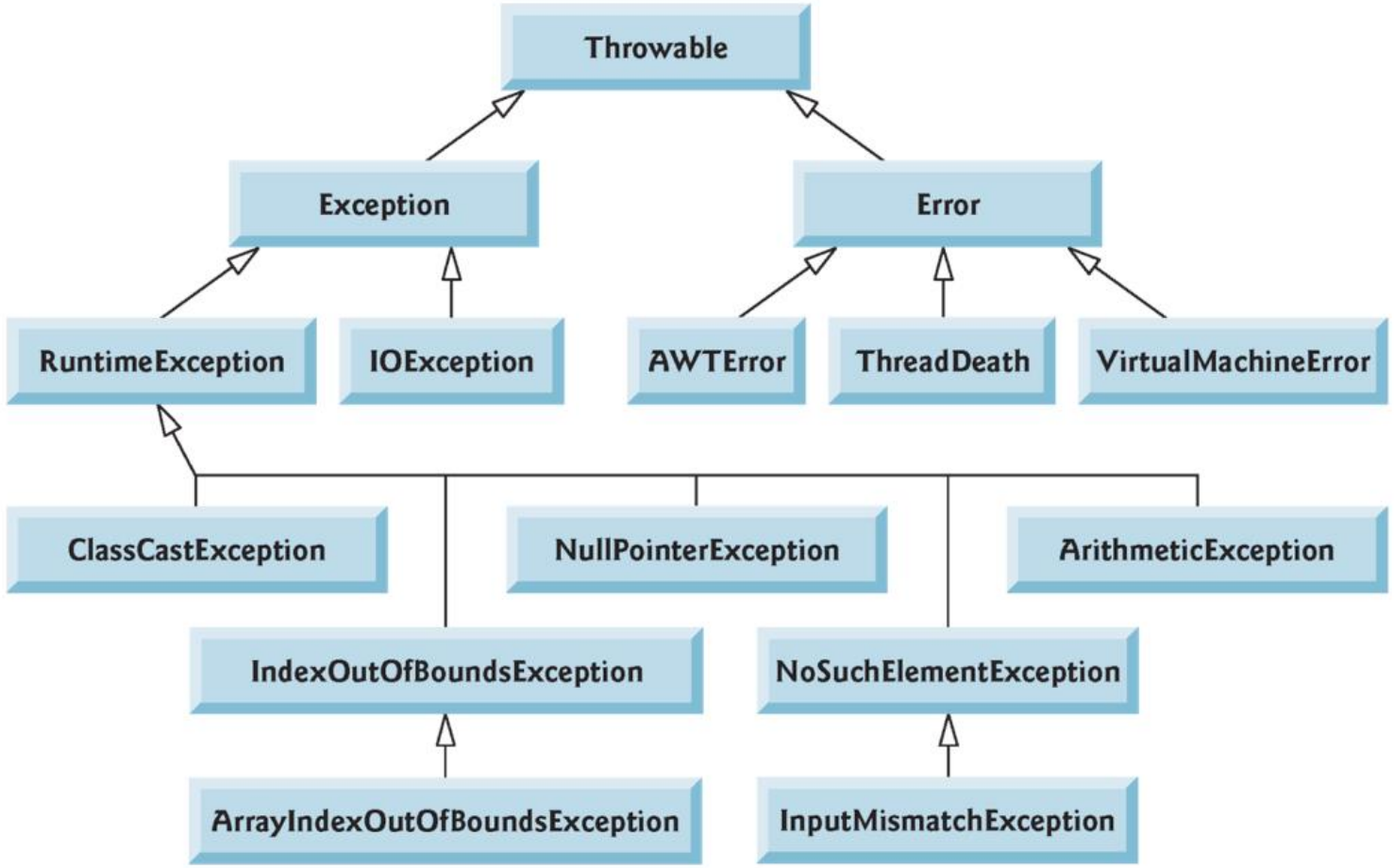
- ในบล๊อกของ try บรรจุขั้นตอนการทำงานปกติ
- จัดการ exceptions 2 ตัวคือ InputMismatchException และ ArithmeticException
- นำ do...while มาใช้เพื่อรับ input ตราบเท่าที่ข้อมูลจะถูกต้องตามเงื่อนไข

ผลลัพธ์การจัดการเอ็กเซปชันเบื้องต้น

```
E:\playground\fund2\week11>java DividedByZeroWithEH
Enter dividend number: 4
Enter divisor number: 0
Divisor must be greater than zero. Please try again!!
Exception: java.lang.ArithmeticException: / by zero
Enter dividend number: 4
Enter divisor number: d
You must enter an integer. Please try again!!
Exception: java.util.InputMismatchException
Enter dividend number: 4
Enter divisor number: 3
1
```

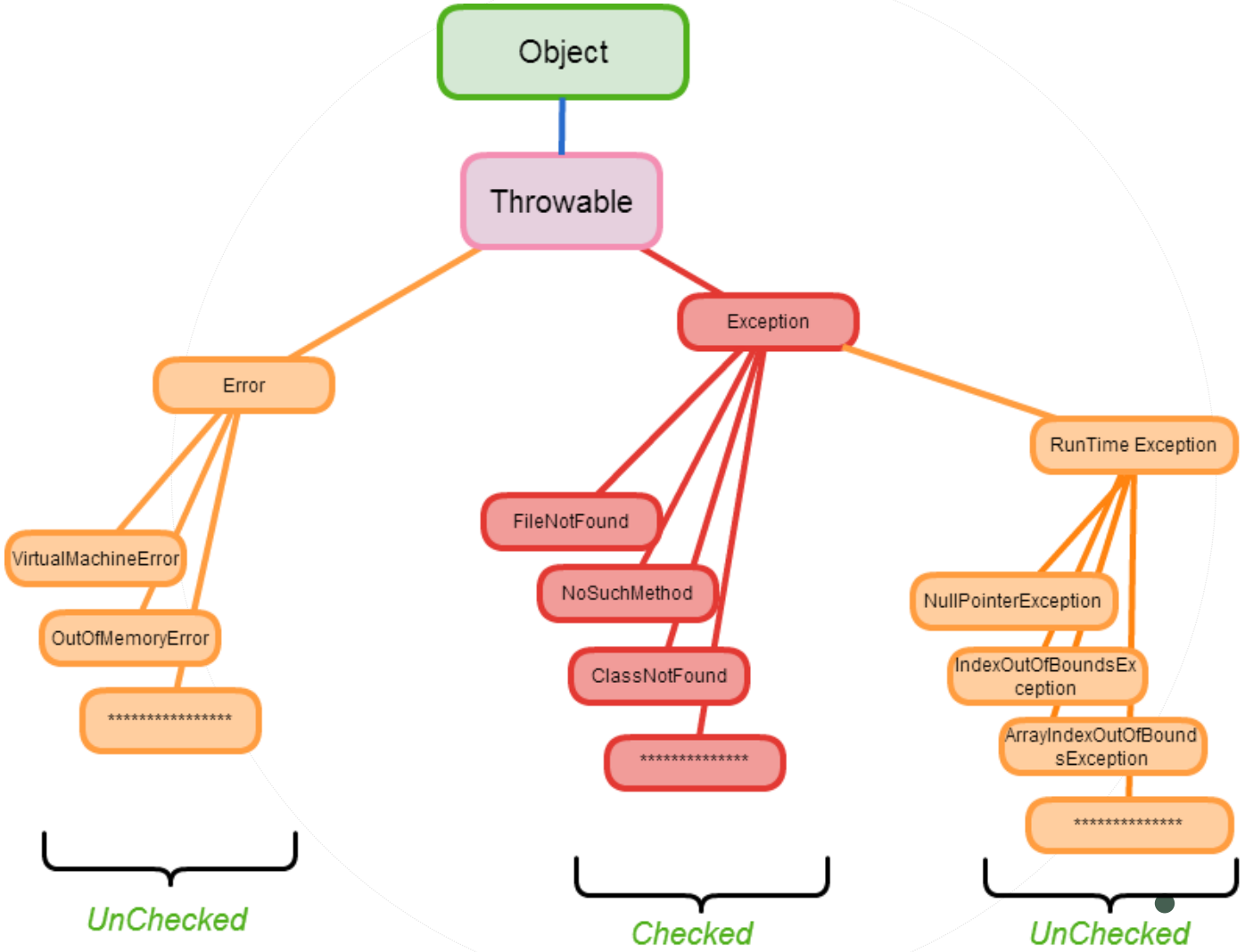
- เมื่อเกิด exception ในบล๊อก try โปรแกรมจะนำ exception ที่เกิดขึ้นไปเทียบกับ catch(Exception) ที่สร้างไว้
- ถ้ากำหนด catch ไว้ตรงกับ exception ที่เกิดขึ้นในขณะโปรแกรมรัน โปรแกรมจะย้ายมาทำงานในส่วนของ catch นั้น ทันที
  - catch(ArithmeticException) จับ exception ที่เกิดในเมทอด quotient() ซึ่งเมทอดนี้ไม่ได้จัดการ exception ทำให้โปรแกรมส่ง exception ดังกล่าวมายังผู้เรียก (main)
  - catch (InputMismatchException) ส่วนที่จับ exception ได้คือ library Scanner แต่ในไลบารีไม่มีตัวจัดการ จึงส่งต่อ exception มายังผู้เรียก (main)

ลำดับชั้นของคลาสเอ็กเซปชัน



- Java แบ่งความผิดปกติออกเป็น 2 กลุ่ม
- **Exception คือ ความผิดปรกติที่จัดการได้**
  - Checked exception คือ exception ที่ถูกตรวจสอบ
  - Unchecked exception คือ exception ที่ไม่ถูกตรวจสอบ
- **Error คือ ความผิดปกติแบบร้ายแรงเกินกว่าโปรแกรมจะจัดการได้**
  - เกิดขึ้นใน JVM เช่น StackOverflowError หรือ OutOfMemoryError

Checked & Unchecked Exception



- คลาสที่ไม่อยู่ในกลุ่ม RuntimeException ถือเป็น checked exception
- checked exception มักจะเป็นข้อผิดพลาดในลักษณะที่อยู่**นอกเหนือการควบคุมของโปรแกรม** เช่น เปิดไฟล์ไม่ได้ หรือไฟล์ไม่มีอยู่ ทำให้เกิด IOException หรือ FileNotFoundException
- **Java บังคับให้สร้างวิธีจัดการ โค้ดส่วนที่อาจทำให้เกิด checked exception ซึ่งโปรแกรมเมอร์สามารถเลือกได้ว่า จะจัดการเอง หรือ จะโยน ให้คนอื่นจัดการ**

Image source: http://ufthelp.com/

ตัวอย่างการ ละเลยการจัดการ exception ในกลุ่ม checked exception

```
7     private static boolean isReadable = false;
8     private static boolean isFileExist = false;
9
10    public static void doLoadImage(String filename) {
11        File myfile = new File(filename);
12        if (myfile.canRead() != false) {
13            isFileExist = true;
14        }
15        if (isFileExist) {
16            img = ImageIO.read(myfile);
17            if (img != null)
18                isReadable = true;
19        }
20    }
21    public static void main(String[] args) {
22        doLoadImage("fakepic.jpg");
23        if(!isFileExist || !isReadable){
24            System.out.println("Cannot read file!");
25        }else{
26            System.out.println("Load image successfully");
27        }
```

```
E:\playground\fund2\week11>javac LoadImageNoEH.java
LoadImageNoEH.java:16: error: unreported exception IOE
xception; must be caught or declared to be thrown
                img = ImageIO.read(myfile);
                                          ^
1 error
```

- IOException (กลุ่ม checked exception) เป็น exception ที่อาจเกิดขึ้นจากการเรียกใช้ เมทอด read()
- ในโค้ดตัวอย่างไม่มีการ catch หรือ throw exception Java compiler จึงไม่ยอมให้โค้ดนี้ผ่าน

ตัวอย่างการจัดการ exception ในกลุ่ม checked exception

```
7     private static BufferedImage img;
8     private static boolean isReadable = false;
9     private static boolean isFileExist = false;
10    public static void doLoadImage(String filename) {
11        File myfile = new File(filename);
12        if (myfile.canRead() != false) {
13            isFileExist = true;
14        }
15        if (isFileExist) {
16            try{
17                img = ImageIO.read(myfile);
18                isReadable = true;
19            }catch(IOException e){
20                System.err.println(e.getMessage());
21            }
22        }
23    }
24    public static void main(String[] args) {
25        doLoadImage("fakepic.jpg");
26        if(!isFileExist || !isReadable){
27            System.out.println("Cannot read file!");
28        }else{
29            System.out.println("Load image successfully");
30        }
31    }
```

```
E:\playground\fund2\week11>java LoadImageNoEH
Cannot read file!
```

- ปรับการทำงานในส่วนการเรียกใช้เมทอด read ด้วย try...catch

ตัวอย่างการโยน exception ในกลุ่ม checked exception

```
7   public class LoadImage {
8       private static BufferedImage img;
9
10      public static void doLoadImage(String filename)
11          throws IOException {
12          File myfile = new File(filename);
13          if(myfile.canRead() == false){
14              throw new IIOException("IIOException is happened");
15          }
16          img = ImageIO.read(myfile);
17          if (img == null)
18              throw new IOException("Error occurs during reading");
19      }
20
21      public static void main(String[] args) {
22          try {
23              doLoadImage("fakepic.jpg");
24          } catch (IIOException e) {
25              System.err.println(e);
26          } catch (IOException e) {
27              System.err.println(e);
28          }
29      }
30  }
```

- การส่งต่อ exception หรือ **การโยน exception** ทำโดยใช้คีย์เวิร์ด throw
- เราสามารถ throw exception ที่ได้จาก JVM, library ที่เรียกใช้ และ exception ที่สร้างเอง (throw new)
- โยน IIOException และ IOException ที่สร้างขึ้นเอง จากเมทอด doLoadImage ไปยังเมทอดผู้เรียก (main)

RuntimeException กลุ่ม Unchecked

- ความผิดพลาดที่ป้องกันได้ เช่น ArithmeticException หรือ NullPointerException
- ป้องกันได้โดยการตรวจสอบค่าก่อนการใช้งาน
- เอ็กเซปชันประเภทนี้**คอมไพเลอร์จะไม่บังคับ**ให้เราจัดการหรือประกาศในส่วนประกาศของเมทอด

บล็อก Finally

- try ... catch เกิดได้ 3 แบบ

1. ไม่เกิดเอ็กเซปชันใด ๆ ในบล็อก try เลย หลังจบบล็อก try โปรแกรมจะทำงานต่อจาก try ... catch ปกติ
2. เกิดเอ็กเซปชันในบล็อก try และ**มีบล็อก catch ที่จัดการเอ็ก**เซปชันนั้น เมื่อทำงานในส่วนบล็อก catch เสร็จ โปรแกรมจะทำงานต่อจาก try ... catch ตามปกติ (ยกเว้นถ้ามีการ throw เอ็กเซปชันอื่นในบล็อก catch นั้นอีก)
3. เกิดเอ็กเซปชันในบล็อก try แต่**ไม่มีบล็อก catch** ที่จัดการเอ็กเซปชันนั้น **เมทอดจะหยุดทำงาน** และส่งเอ็กเซปชันนั้นออกไปยังเมทอดที่เรียก
   - ในกรณีที่ 3 เราอยากให้โค้ดส่วนอื่นได้รันต่อไป ถึงแม้ว่าการจัดการ exception จะย้ายไปยังผู้เรียกแล้วก็ตาม เช่น การปิดฐานข้อมูลถึงแม้อ่านข้อมูลไม่สำเร็จ
   - ปัญหาดังกล่าวแก้ด้วยการใช้ try...catch...**finally**

บล็อก Finally (ต่อ)

- บล็อก **finally ถูกทำงานเสมอ**ไม่ว่าจะเกิดกรณีใดก็ตามใน 3 แบบที่กล่าวมา
- ถึงแม้มีการสั่ง return, break หรือ continue ที่ทำให้เกิดการออกจากบล็อก try ไปแล้ว แต่โปรแกรมส่วนที่อยู่ใน finally ก็ยังทำงาน
  - **ยกเว้นมีการเรียก System.exit()**
- การใช้ try...catch ไม่จำเป็นต้องมี finally
- **แต่ try ไร้ catch ต้องมี finally มาเสริม**

ตัวอย่างการใช้ finally

```java
public class UsingExceptions {
    public static void main(String[] args) {
        try {
            throwException();
        } catch (Exception e) {
            System.err.println("Exception handled in main()");
        }

        doesNotThrowException();
    }

    public static void throwException() throws Exception {
        try {
            System.out.println("Method throwException()");
            throw new Exception();
        } catch (Exception e) {
            System.err.println("Exception handled in throwException()");
            throw e; // rethrow the caught exception
        } finally {
            System.err.println("Finally executed in throwException()");
        }
    }
```

```java
    public static void doesNotThrowException() {
        try {
            System.out.println("Method doesNotThrowException()");
        } catch (Exception e) {
            System.err.println(e);
        } finally {
            System.err.println("Finally executed in doesNotThrowException()");
        }

        System.out.println("End of method doesNotThrowException()");
    }
```

ผลการรันเป็นดังนี้

```
Method throwException()
Exception handled in throwException()
Finally executed in throwException()
Exception handled in main()
Method doesNotThrowException()
Finally executed in doesNotThrowException()
End of method doesNotThrowException()
```

รายการย้อนรอยสแต็ก

- ด้วยคุณสมบัติที่เราสามารถ โยนต่อ และ ส่งต่อ exception ได้
- Java จึงสร้างรายการที่เรียกว่า “**Stack trace**” เพื่อให้เราสามารถดูข้อมูลการย้อนกลับของเมทอดได้ โดยไม่ต้องรอให้การย้อนกลับไปถึง main()
- printStackTrace() หรือ getStackTrace() คือ เมทอด สำหรับเรียกดูข้อมูลการย้อน

ตัวอย่างการเรียกดูรายการย้อนรอยสแต็ก

```java
public static void main(String[] args) {
    try {
        method1();
    } catch (Exception e) {
        System.err.printf("%s%n%n", e.getMessage());
        e.printStackTrace();

        // Obtain stack-trace information
        StackTraceElement[] traceElements = e.getStackTrace();

        System.out.printf("%nStack trace from getStackTrace():%n");
        System.out.println("Class\t\tFile\t\t\tLine\tMethod");
        for (StackTraceElement element : traceElements) {
            System.out.printf("%s\t", element.getClassName());
            System.out.printf("%s\t", element.getFileName());
            System.out.printf("%s\t", element.getLineNumber());
            System.out.printf("%s%n", element.getMethodName());
        }
    }
}
```

- มาหลัง ได้ทำก่อน

```java
public static void method1() throws Exception {
    method2();
}

public static void method2() throws Exception {
    method3();
}

public static void method3() throws Exception {
    throw new Exception("Exception thrown in method3()");
}
```

```
Exception thrown in method3()

java.lang.Exception: Exception thrown in method3()
    at UsingExceptions.method3(UsingExceptions.java:32)
    at UsingExceptions.method2(UsingExceptions.java:28)
    at UsingExceptions.method1(UsingExceptions.java:24)
    at UsingExceptions.main(UsingExceptions.java:4)

Stack trace from getStackTrace():
Class           File                    Line    Method
UsingExceptions UsingExceptions.java    32      method3
UsingExceptions UsingExceptions.java    28      method2
UsingExceptions UsingExceptions.java    24      method1
UsingExceptions UsingExceptions.java    4       main
```

การโยงเอ็กเซปชันเป็นทอด

- เราสามารถจัดการ exception แล้วค่อยส่งต่อได้ โดยการ catch แล้วสั่ง throw
- ใช้ประโยชน์เมื่อ ต้องการสร้าง exception ของตัวเอง ด้วยการนำ exception เดิม (ของ Java/Library) มาเป็นส่วนหนึ่งของ exception ใหม่
- เรียกการกระทำดังกล่าวว่า “**exception chaining**”

ตัวอย่าง exception chaining

```java
public class UsingChainedExceptions {
    public static void main(String[] args) {
        try {
            method1();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void method1() throws Exception {
        try {
            method2();
        } catch (Exception e) {
            throw new Exception("Exception thrown in method1()", e);
        }
    }

    public static void method2() throws Exception {
        try {
            method3();
        } catch (Exception e) {
            throw new Exception("Exception thrown in method2()", e);
        }
    }

    public static void method3() throws Exception {
        throw new Exception("Exception thrown in method3()");
    }
}
```

- exception ของ method1() เกิดจากการรวมกันระหว่าง exception ของ method2() กับ exception ของ method3()

```
java.lang.Exception: Exception thrown in method1()
    at UsingChainedExceptions.method1(UsingChainedExceptions.java:14)
    at UsingChainedExceptions.main(UsingChainedExceptions.java:4)
Caused by: java.lang.Exception: Exception thrown in method2()
    at UsingChainedExceptions.method2(UsingChainedExceptions.java:22)
    at UsingChainedExceptions.method1(UsingChainedExceptions.java:12)
    ... 1 more
Caused by: java.lang.Exception: Exception thrown in method3()
    at UsingChainedExceptions.method3(UsingChainedExceptions.java:27)
    at UsingChainedExceptions.method2(UsingChainedExceptions.java:20)
    ... 2 more
```

การสร้างคลาสเอ็กเซปชันใหม่

- คลาสที่สร้างใหม่นี้จะต้อง extends จากคลาสเอ็กเซปชันที่มีอยู่แล้ว
- ประกอบด้วย**คอนสตรักเตอร์ 4 แบบ**ดังนี้

1. **แบบที่ไม่รับพารามิเตอร์** และส่งข้อความแสดงความผิดพลาดที่กำหนดแบบโดยปริยายไว้ไปให้กับคอนสตรักเตอร์ของซูเปอร์คลาส
2. แบบที่**รับพารามิเตอร์เป็นข้อความแสดงความผิดพลาด** และส่งข้อความนั้นไปให้กับคอนสตรักเตอร์ของซูเปอร์คลาส
3. แบบที่**รับ**พารามิเตอร์เป็นข้อความแสดงความผิดพลาด**และอ็อบเจกต์ชนิด** Throwable ซึ่งจะใช้กรณีที่ต้องการโยงเอ็กเซปชันเป็นทอด (exception chaining) แล้วส่งทั้งข้อความและอ็อบเจกต์นี้ต่อไปให้กับคอนสตรักเตอร์ของซูเปอร์คลาส
4. แบบที่**รับ**พารามิเตอร์เป็นอ็อบเจกต์ชนิด Throwable ซึ่งจะใช้กรณีที่ต้องการโยงเอ็กเซปชันเป็นทอด แล้วส่งอ็อบเจกต์นี้ต่อไปให้กับคอนสตรักเตอร์ของซูเปอร์คลาส

การใช้ assert

- ในระหว่างทดสอบหรือแก้ไขโปรแกรม เรามักคาดการณ์ว่าค่าในแต่ละจุดของโปรแกรม ตัวแปรที่เราสนใจควรมีเท่าใด (ซึ่งเป็นไปได้ว่าไม่เท่ากันทุกที่)
- หากโปรแกรมที่เขียนนั้นถูกต้อง ค่าตัวแปรภายในโปรแกรมควรเป็นไปตามที่โปรแกรมเมอร์คาดการณ์ไว้
- **Java สร้างเครื่องมือสำหรับการตรวจสอบค่าที่เกิดขึ้นจริง กับ ค่าที่คาดการณ์ไว้**

```
assert expression;

assert expression1 : expression2;
```

ตัวอย่างการใช้ assert

```java
import java.util.Scanner;

public class AssertTest {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter a number between 0 and 10: ");
        int number = input.nextInt();

        assert (number >= 0 && number <= 10) : "bad number: " + number;

        System.out.printf("You entered %d%n", number);
    }
}
```

```
Enter a number between 0 and 10: 8
You entered 8

Enter a number between 0 and 10: 11
Exception in thread "main" java.lang.AssertionError: bad number: 11
    at AssertTest.main(AssertTest.java:10)
```

- ปกติแล้วจะใช้ assert ดักไว้ตามจุดต่าง ๆ เพื่อทดสอบค่าบางค่าว่าเป็นไปตามที่ควรจะเป็นหรือไม่
- ซึ่งถ้าทำงานถูก เราไม่ควรเห็น AssertionError

อ้างอิง

- https://nbviewer.jupyter.org/github/Poonna/java-book/