

บทที่ 10: การจัดการข้อผิดพลาด (Exception Handling)

บรรยายโดย ผศ.ดร.ธรรดา วิเชษฐ์ ริติจรูญโรจน์

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

หัวข้อ



- ข้อผิดพลาดคืออะไร
- การสร้างคลาสประเภทข้อผิดพลาดขึ้นมาใหม่
- วิธีการจัดการข้อผิดพลาด
- หลักการการจัดการกับข้อผิดพลาด
- ความแตกต่างระหว่าง if-else กับ try-catch

หัวข้อ



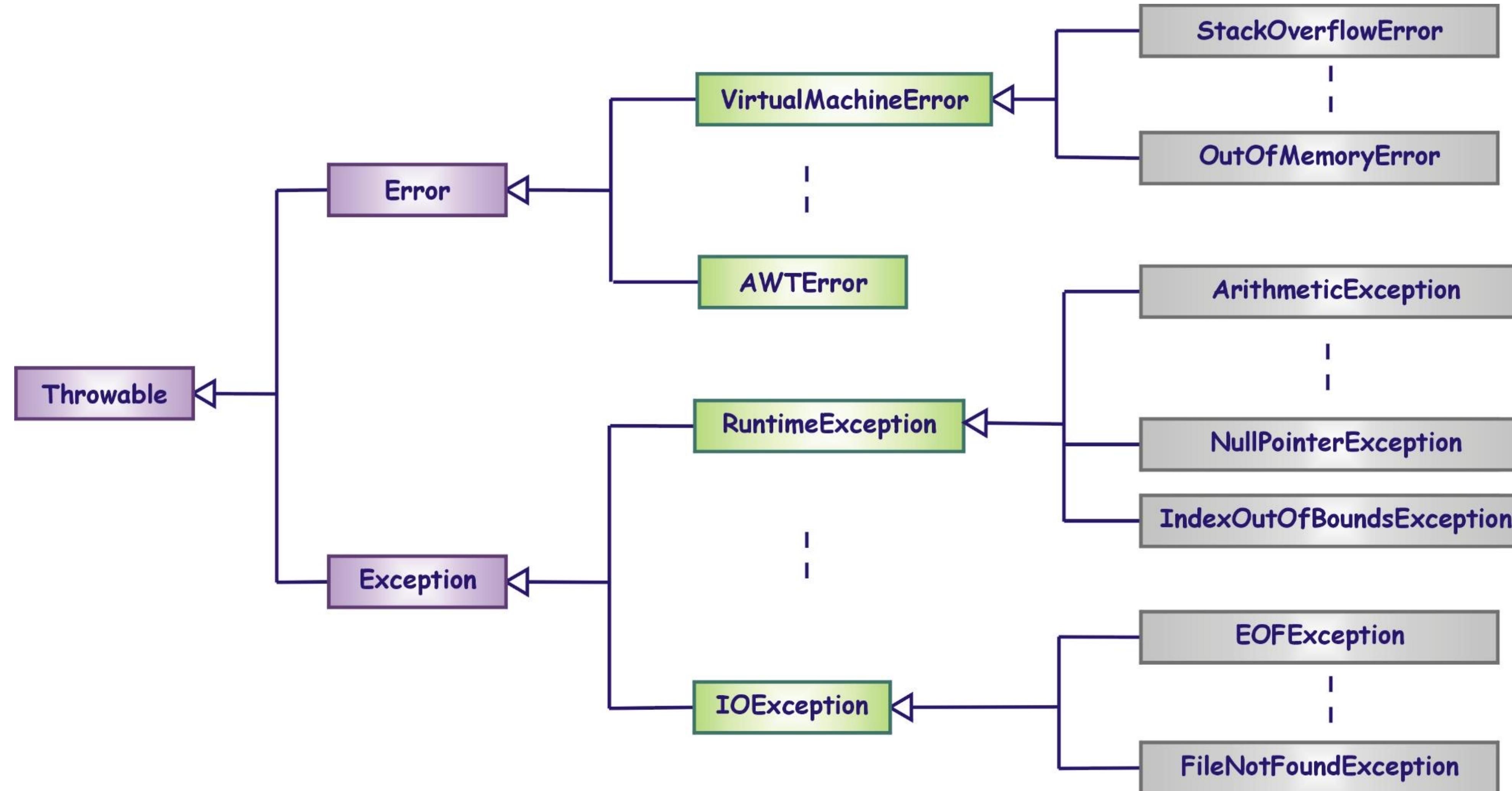
- ข้อผิดพลาดคืออะไร
 - การสร้างคลาสประเภทข้อผิดพลาดขึ้นมาใหม่
 - วิธีการจัดการข้อผิดพลาด
 - หลักการการจัดการกับข้อผิดพลาด
 - ความแตกต่างระหว่าง if-else กับ try-catch

ข้อผิดพลาด



- ข้อผิดพลาดที่อาจเกิดขึ้นขณะรันโปรแกรม แบ่งเป็น 2 ประเภท
 - **Error** เป็นข้อผิดพลาดที่ **ไม่สามารถแก้ไข** และ **จัดการได้** ส่วนมากเกิดจากการขาดแคลนทรัพยากรของคอมพิวเตอร์ อาทิ เช่น system crash หรือ out of memory โดยส่วนมากจะเกิดในสถานะ Runtime เช่น *VirtualMachineError, OutOfMemoryError*
 - **Exception** เป็นข้อผิดพลาดที่ **สามารถแก้ไข** หรือ **จัดการได้** ส่วนมากจะเกิดจากโค้ดของนักพัฒนาโปรแกรม พบมากในสถานะ runtime และ compile time เช่น *FileNotFoundException, ArrayIndexOutOfBoundsException*
- ข้อผิดพลาดจะกำหนดเป็นอปเจ็คของคลาสต่าง ๆ โดยมีคลาส **Throwable** เป็นคลาสราก

คลาสที่เกี่ยวข้องกับข้อผิดพลาด



คลาสประเภท Exception



- Exception เป็นข้อผิดพลาดที่**เกิดในขณะรันโปรแกรม**ภาษา Java ซึ่งสามารถแบ่งออกเป็น 2 ประเภท คือ
 - RuntimeException
 - เป็นข้อผิดพลาดที่อาจ**หลีกเลี่ยงได้**หากมีการเขียนโปรแกรมที่ถูกต้อง
 - IOException
 - เป็นข้อผิดพลาดที่ภาษาจาวากำหนดให้**ต้องมีการจัดการ** หากมีการเรียกใช้เมธอดที่**อาจเกิดข้อผิดพลาดประเภทนี้**ได้

คลาสประเภท Exception ที่สำคัญและพบบ่อย

- ArithmeticException
- ArrayIndexOutOfBoundsException
- EOFException
- FileNotFoundException
- InterruptedException
- IOException
- NullPointerException
- NumberFormatException

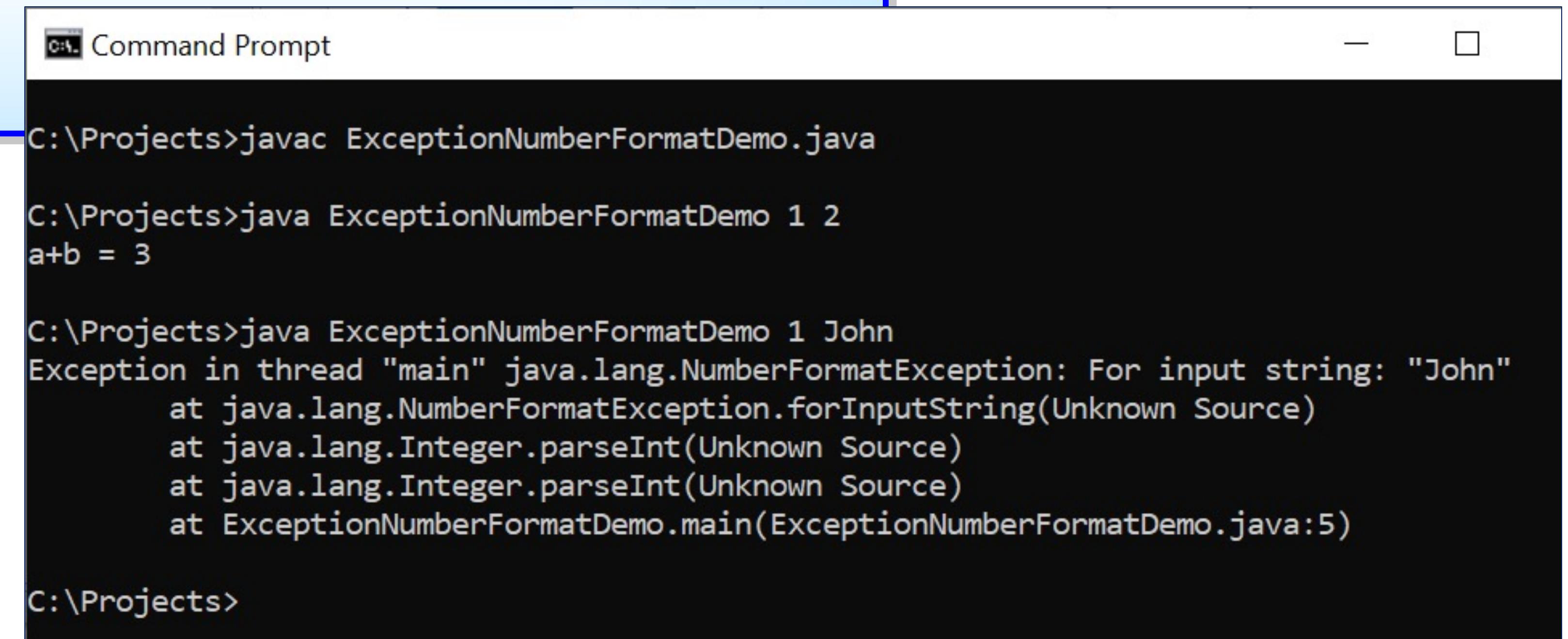
ตัวอย่างการเกิด ArithmeticException

```
public class ExceptionArithmeticDemo {  
    public static void main(String args[]) {  
        int a = 1;  
        int b = 0;  
        System.out.println("a/b = " + a/b);  
    }  
}
```

```
Output - Lab10 (run) ×  
run:  
Exception in thread "main" java.lang.ArithmeicException: / by zero  
    at lecture.ExceptionArithmeticDemo.main(ExceptionArithmeticDemo.java:8)  
C:\Users\supan\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1  
BUILD FAILED (total time: 0 seconds)
```

ตัวอย่างการเกิด NumberFormatException

```
public class ExceptionNumberFormatExceptionDemo {  
    public static void main(String args[]) {  
        int a = Integer.parseInt(args[0]);  
        int b = Integer.parseInt(args[1]);  
        System.out.println("a+b = " + (a+b));  
    }  
}
```

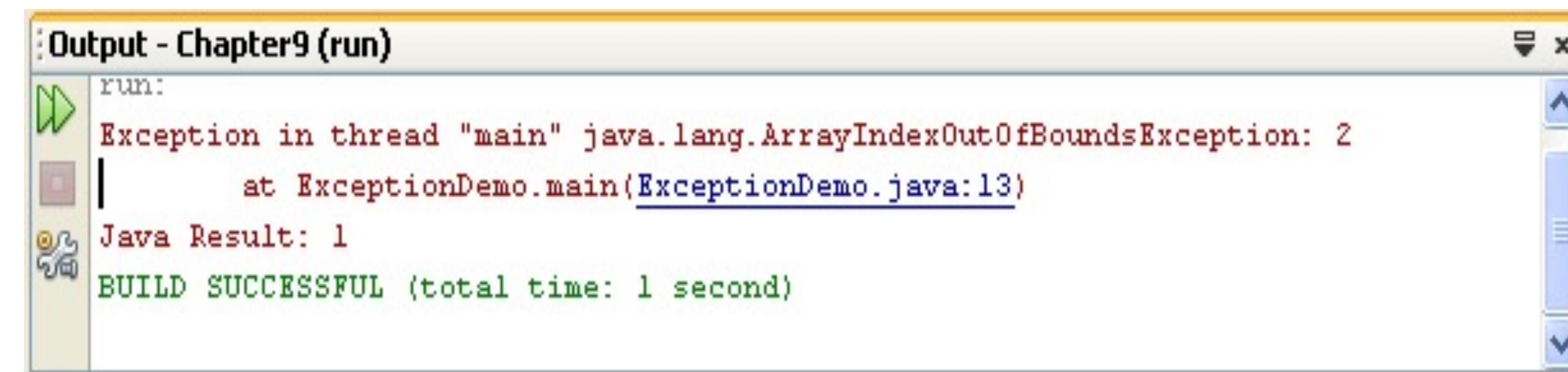


The screenshot shows a Windows Command Prompt window titled "Command Prompt". It displays the execution of a Java program named "ExceptionNumberFormatExceptionDemo". The first two executions provide correct integer inputs (1 and 2) and output the sum (3). The third execution provides a non-integer input ("John") which triggers a `NumberFormatException`, causing the application to crash and displaying the full stack trace.

```
C:\Projects>javac ExceptionNumberFormatExceptionDemo.java  
  
C:\Projects>java ExceptionNumberFormatExceptionDemo 1 2  
a+b = 3  
  
C:\Projects>java ExceptionNumberFormatExceptionDemo 1 John  
Exception in thread "main" java.lang.NumberFormatException: For input string: "John"  
        at java.lang.NumberFormatException.forInputString(Unknown Source)  
        at java.lang.Integer.parseInt(Unknown Source)  
        at java.lang.Integer.parseInt(Unknown Source)  
        at ExceptionNumberFormatExceptionDemo.main(ExceptionNumberFormatExceptionDemo.java:5)  
  
C:\Projects>
```

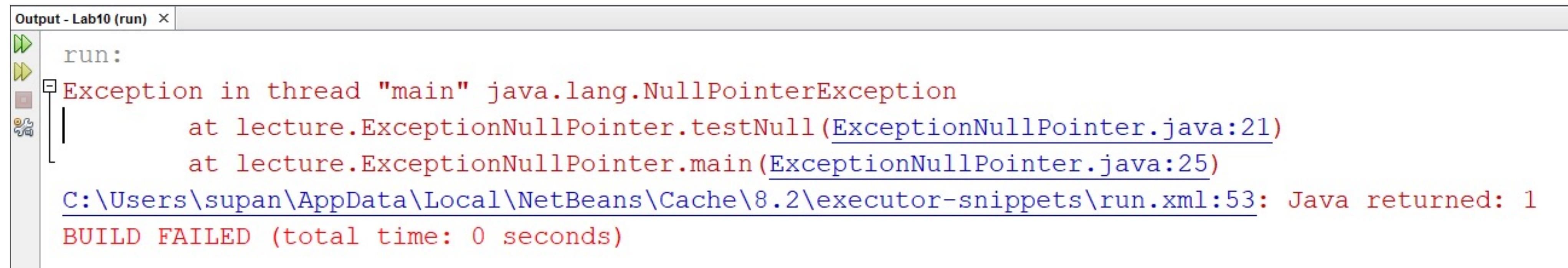
ตัวอย่างการเกิด ArrayIndexOutOfBoundsException

```
public class ExceptionDemo {  
    public static void main(String args[]) {  
        int [] num = {10,20};  
        System.out.println(num[2]);  
        System.out.println("Hello");  
    }  
}
```



ตัวอย่างการเกิด NullPointerException

```
public class ExceptionNullPointer {  
    private Student s1;  
    public void testNull() {  
        System.out.println("Name: " + s1.getName());  
    }  
    public static void main(String[] args) {  
        new ExceptionNullPointer().testNull();  
    }  
}
```



The screenshot shows the NetBeans IDE's Output window titled "Output - Lab10 (run) x". The window displays the following text:

```
run:  
Exception in thread "main" java.lang.NullPointerException  
    at lecture.ExceptionNullPointer.testNull (ExceptionNullPointer.java:21)  
    at lecture.ExceptionNullPointer.main (ExceptionNullPointer.java:25)  
C:\Users\supan\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1  
BUILD FAILED (total time: 0 seconds)
```

หัวข้อ



- ข้อผิดพลาดคืออะไร
- การสร้างคลาสประเภทข้อผิดพลาดขึ้นมาใหม่
- วิธีการจัดการข้อผิดพลาด
- หลักการการจัดการกับข้อผิดพลาด
- ความแตกต่างระหว่าง if-else กับ try-catch

การสร้างคลาสประเท **Exception** ขึ้นใหม่

การสร้างคลาสประเท Exception ขึ้นมาใหม่ สามารถทำได้โดยนิยามคลาสได ๆ ให้สืบทอดมาจากคลาสที่ชื่อ Exception โดยทั่วไป คลาสที่ชื่อ Exception จะมี constructor สองรูปแบบคือ

- **public Exception()**
- **public Exception(String s)**

ดังนั้น คลาสที่สืบทอดมาจากคลาสที่ชื่อ Exception **ควรจะมี constructor ทั้งสองแบบ** โดยรูปแบบหนึ่งจะมี argument ที่มีชนิดข้อมูลเป็น String และมีคำสั่งแรกใน constructor เป็นคำสั่ง super(s);

```
public class MyOwnException extends Exception {  
    public MyOwnException (String s) {  
        super(s);  
    }  
}
```

การเขียนเมธอดเพื่อส่งออกเจ็คประกาย Exception



เมธอดที่ต้องการส่งออกเจ็คประกาย Exception เมื่อเกิดข้อผิดพลาดขึ้นในคำสั่งใด จะต้องเรียกใช้คำสั่ง `throw` เพื่อจะสร้างออกเจ็คของคลาสประกาย Exception ขึ้นมา

รูปแบบ

```
throw new ExceptionType( [arguments] )
```

นอกจากนี้ คำสั่งประกาศเมธอดนั้นจะต้องมีคำสั่ง `throws` เพื่อกำหนดให้คำสั่งในเมธอดอื่น ๆ ที่เรียกใช้ เมธอดนี้ต้องเขียนคำสั่งในการจัดการกับข้อผิดพลาดนี้

ตัวอย่างคลาส FileHandler



```
import java.io.*;  
  
public class FileHandler {  
    public static void openFile(String filename) throws MyOwnException {  
        File f = new File(filename);  
        if (!f.exists()) {  
            throw new MyOwnException("File Not Found");  
        }  
    }  
  
    public class MyOwnException extends Exception {  
        public MyOwnException (String s) {  
            super(s);  
        }  
    }  
}
```

ตัวอย่างโปรแกรมที่มีการจัดการกับข้อผิดพลาด



```
public class FileOpener {  
    public static void main (String args[]) {  
        try {  
            FileHandler.openFile(args[0]);  
            System.out.println("Open successful");  
        } catch (MyOwnException ex) {  
            System.err.println(ex);  
        }  
    }  
}
```

หัวข้อ



- ข้อผิดพลาดคืออะไร
- การสร้างคลาสประเภทข้อผิดพลาดขึ้นมาใหม่
- วิธีการจัดการข้อผิดพลาด
- หลักการการจัดการกับข้อผิดพลาด
- ความแตกต่างระหว่าง if-else กับ try-catch

การจัดการกับ Exception



การจัดการกับ Exception มี 2 แบบ คือ

- ใช้คำสั่ง **try-catch**
- ใช้คำสั่ง **throws** ในการประกาศเมธอดที่จะมีการเรียกใช้เมธอดได ๆ ที่อาจส่งออกเจ็คประเภท Exception ซึ่งแบ่งได้เป็น 3 แบบ
 - Throwing
 - Rethrowing
 - Wrapping

การจัดการกับ Exception



การจัดการกับ Exception มี 2 แบบ คือ

- ใช้คำสั่ง **try-catch**

- ใช้คำสั่ง **throws** ในการประกาศเมธอดที่จะมีการเรียกใช้เมธอดได ๆ ที่อาจส่งออกเจ็คประเภท Exception ซึ่งแบ่งได้เป็น 3 แบบ
 - Throwing
 - Rethrowing
 - Wrapping

คำสั่ง try...catch



ภาษาจาวามีคีย์เวิร์ด try ที่เป็นคำสั่งที่ใช้ในการจัดการกับเมธอดหรือคำสั่งที่อาจเกิดข้อผิดพลาดซึ่งจะส่งออกเจ็คประเภท Exception ในขณะรันโปรแกรม

รูปแบบ

```
try {  
    [statements]  
}
```

โปรแกรมภาษาจาวาจะสั่งงานชุดคำสั่งที่อยู่ในบล็อกที่ลงทะเบียน และหากเกิดข้อผิดพลาดขึ้นในคำสั่งประเภทใดก็จะมีการส่ง **อุปเจ็คของข้อผิดพลาดประเภท Exception** นั้นขึ้นมา

คำสั่ง try...catch



ในการณีที่ต้องการจัดการกับข้อผิดพลาดที่เกิดขึ้น โปรแกรมจะต้องมีชุดคำสั่งอยู่ในบล็อกของคีย์เวิร์ด catch ที่จะระบุชนิดของอปเจ็คในคลาสประเภท Exception ที่ต้องการจัดการ

รูปแบบ

```
catch (ExceptionType argumentName) {  
    [statements]  
}
```

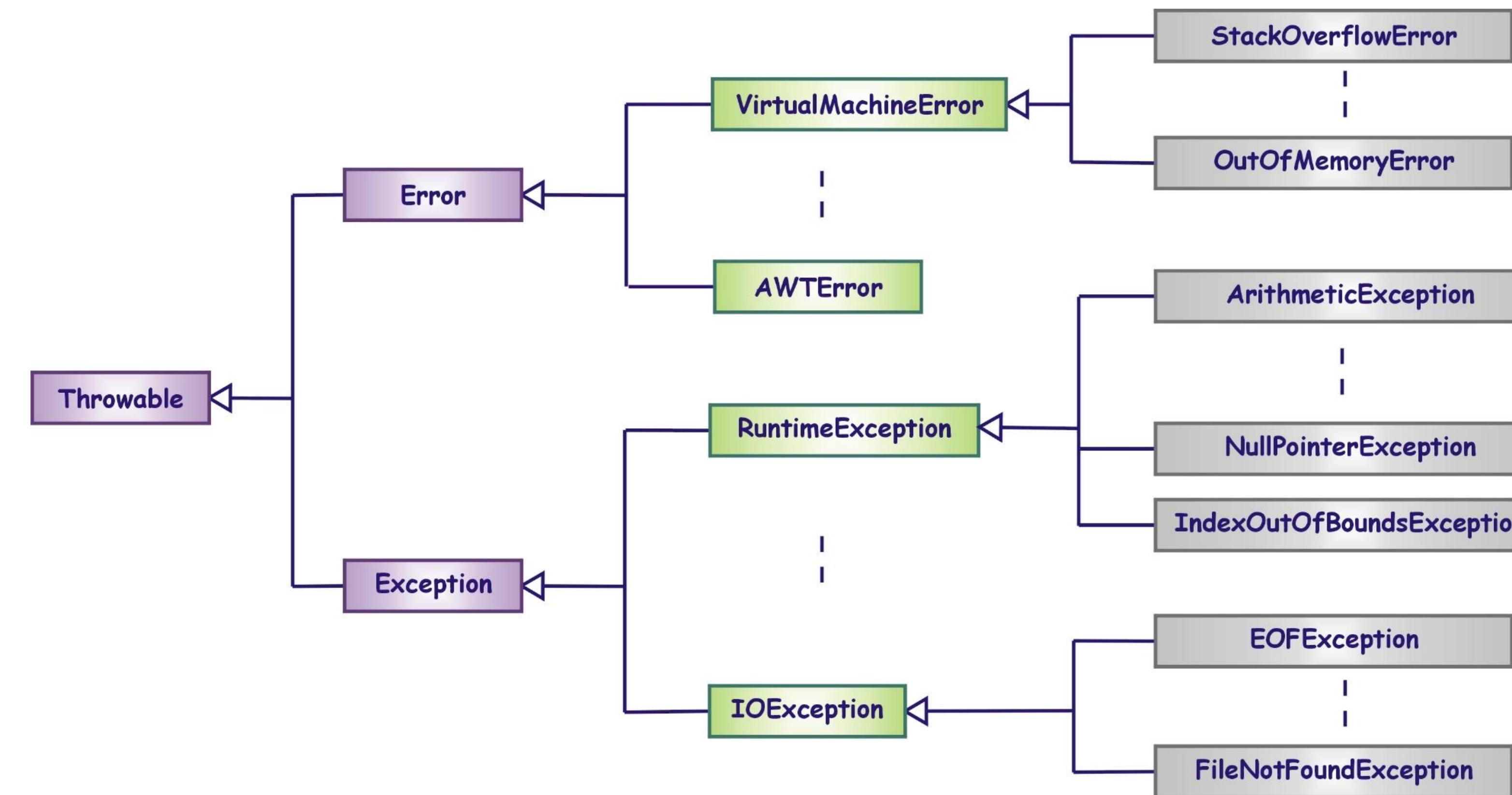
ตัวอย่างโปรแกรมที่มีการจัดการกับข้อผิดพลาด

```
public class ExceptionHandlingDemo {  
    public static void main(String args[]) {  
        int [] num = {10,20};  
        try {  
            System.out.println(args[2]);  
        } catch(ArrayIndexOutOfBoundsException ex) {  
            System.out.println("There is no third");  
        }  
    }  
}
```

The screenshot shows the 'Output' window of an IDE during the execution of the 'ExceptionHandlingDemo' program. The window title is 'Output - Chapter9 (run)'. The output text is:
run:
There is no third
BUILD SUCCESSFUL (total time: 0 seconds)

การจัดการกับข้อผิดพลาด

เราสามารถจะจัดการกับอปเจ็คของ Exception โดยใช้คลาสที่เป็น superclass ของ Exception นั้นได้
อาทิเช่น **เราสามารถจัดการกับ FileNotFoundException โดยใช้ IOException หรือ Exception**
แทนได้



การจัดการกับข้อผิดพลาดหลาย ๆ ประเภท



- โปรแกรมภาษาจาวาสามารถจะมีชุดคำสั่งของบล็อก catch ได้มากกว่าหนึ่งชุดสำหรับในแต่ละบล็อกคำสั่ง try
- ชนิดของอปเจ็คประเภท Exception ที่อยู่ในชุดคำสั่งของบล็อก catch จะต้องเรียงตามลำดับการสืบทอด
- ในกรณีที่มีข้อผิดพลาดเกิดขึ้น ภาษาจาวาจะพิจารณาว่าเป็นข้อผิดพลาดชนิดใด ซึ่งการที่จะจัดการกับอปเจ็คประเภท Exception นั้นจะพิจารณาจากคลาสที่มีการสืบทอดตามลำดับซึ่ง

การจัดการกับข้อผิดพลาดหลาย ๆ ประเภท



โปรแกรมภาษาจาวาสามารถจะมีชุดคำสั่งของบล็อก catch ได้มากกว่าหนึ่งชุดสำหรับในแต่ละบล็อกคำสั่ง try

```
public class SeqExceptionExample {  
    public void testNumber(int a, int b) {  
        try{  
            System.out.println(a/b);  
        }catch(ArithmetricException e){  
            e.printStackTrace();  
        }catch(Exception e){  
            e.printStackTrace();  
        }  
    }  
    public static void main(String[] args) {  
        new SeqExceptionExample().testNumber(5, 0);  
    }  
}  
  
java.lang.ArithmetricException: / by zero  
at TryCatchProject.SeqExceptionExample.testNumber(SeqExceptionExample.java:15)  
at TryCatchProject.SeqExceptionExample.main(SeqExceptionExample.java:23)
```

การจัดการกับข้อผิดพลาดหลาย ๆ ประเภท



ชนิดของอปเจ็คประเภท Exception ที่อยู่ในชุดคำสั่งของบล็อก catch จะต้องเรียงตามลำดับการสืบทอด

```
public class SeqExceptionExample {  
    public void testNumber(int a, int b) {  
        try{  
            System.out.println(a/b);  
        }catch(Exception e){  
            e.printStackTrace();  
        }catch(ArithmeticException e){  
            e.printStackTrace();  
        }  
    }  
    public static void main(String[] args) {  
        new SeqExceptionExample().testNumber(5, 0);  
    }  
}
```

Exception in thread "main" java.lang.RuntimeException: Uncompilable source code -
exception java.lang.ArithmeticException has already been caught
at TryCatchProject.SeqExceptionExample.testNumber(SeqExceptionExample.java:14)
at TryCatchProject.SeqExceptionExample.main(SeqExceptionExample.java:24)

การจัดการกับข้อผิดพลาดหลาย ๆ ประเภท



ในกรณีที่มีข้อผิดพลาดเกิดขึ้น ภาษาจาวาจะพิจารณาว่าเป็นข้อผิดพลาดชนิดใด ซึ่งการที่จะจัดการกับอุปกรณ์ประเภท Exception นั้นจะพิจารณาจากคลาสที่มีการสืบทอดตามลำดับชั้น

```
public class ExceptionHandlingDemoV2 {  
    public static void main(String args[]) {  
        try {  
            int i = Integer.parseInt(new Scanner(System.in).nextLine());  
            System.out.println(4 / i);  
        } catch(ArithmetricException ex) {  
            System.out.println(ex.toString());  
        } catch(NumberFormatException ex) {  
            System.out.println("Invalid numeric format");  
        }  
    }  
}
```

ตัวอย่างโปรแกรมที่ไม่ถูกต้อง



```
public class ExceptionHandlingDemoV3 {  
    public static void main(String args[]) {  
        try {  
            int i = Integer.parseInt(args[0]);  
            System.out.println(4 / i);  
            System.out.println(args[2]);  
        } catch (RuntimeException ex) {  
            System.out.println(ex.toString());  
        } catch (ArrayIndexOutOfBoundsException ex) {  
            System.out.println("There is no third command line argument");  
        }  
    }  
}
```

Class ArrayIndexOutOfBoundsException

```
java.lang.Object  
    java.lang.Throwable  
        java.lang.Exception  
            java.lang.RuntimeException  
                java.lang.IndexOutOfBoundsException  
                    java.lang.ArrayIndexOutOfBoundsException
```

บล็อก finally



- ภาษา Java มีคีย์เวิร์ด `finally` ที่จะมีชุดคำสั่งอยู่ในบล็อกเพื่อระบุให้โปรแกรมทำชุดคำสั่ง ดังกล่าวหลังจากสิ้นสุดการทำงานของชุดคำสั่งในบล็อก `try` หรือ `catch`
- ภาษา Java จะทำชุดคำสั่งในบล็อก `finally` เสมอ แม้ว่าจะมีคำสั่ง `return` ในบล็อก `try` หรือ `catch` ก่อนก็ตาม
- กรณีเดียวที่จะไม่ทำชุดคำสั่งในบล็อก `finally` คือมีคำสั่ง `System.exit();`

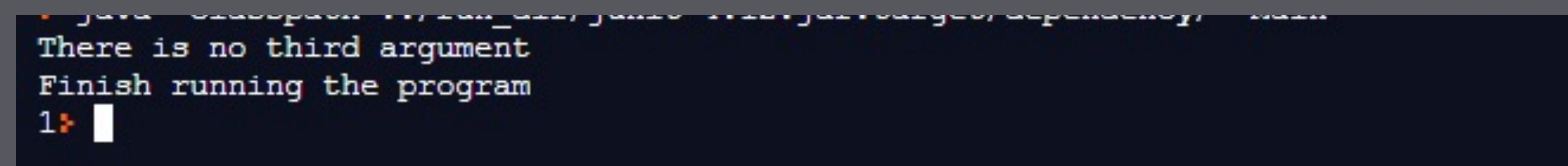
ตัวอย่างโปรแกรม



```
public class FinallyDemo {  
    public static void main(String args[]) {  
        try {  
            System.out.println(args[2]);  
            System.out.println("Hello");  
        } catch(ArrayIndexOutOfBoundsException ex) {  
            System.out.println("There is no third argument");  
        } finally {  
            System.out.println("Finish running the program");  
        }  
    }  
}
```

ตัวอย่างโปรแกรม

```
public class Main {  
    public static void main(String args[]) {  
        Main obj = new Main();  
        System.out.println(obj.myMethod(args));  
    }  
    public int myMethod(String args[]) {  
        try {  
            System.out.println(args[2]);  
            return 0;  
        } catch(ArrayIndexOutOfBoundsException ex) {  
            System.out.println("There is no third argument");  
        } finally {  
            System.out.println("Finish running the program");  
            return 1;  
        }  
    }  
}
```



A terminal window showing the execution of a Java program. The output is:
There is no third argument
Finish running the program
1> █

การ Union ของคำสั่ง Try-Catch



เพื่อที่จะลดโค้ดที่ซ้ำซ้อนหรือเหมือนกันลง ตั้งแต่ **jdk 1.7** เป็นต้นมา จะสามารถรวมโค้ดในส่วนของ catch ที่มีการทำงานเหมือนกันเข้ามาไว้ใน catch อันเดียวกันได้ โดยอาศัยตัวดำเนินการ | ดังแสดงในตัวอย่าง

```
public class TestUnionCode {  
    public void testCode(String filePath) {  
        try {  
            // some code  
        } catch (IOException | NumberFormatException ex) {  
            // handle  
        }  
    }  
}
```

การใช้ try-with-resources



ตั้งแต่ jdk 1.7 เป็นต้นมา การจัดการทรัพยากรที่จำเป็นต้องปิดหรือคืนทรัพยากรนั้น สามารถใช้งาน **try-with-resources** ได้ ซึ่ง try-with-resources จะช่วยอ่านวิความสะดวกเกี่ยวกับการเชื่อมต่อ และ/หรือการคืนทรัพยากรให้กับระบบ โดยที่เมื่อสิ้นสุดการทำงานภายใน try-with-resources แล้ว ทรัพยากรเหล่านั้นจะถูกคืนให้ระบบโดยอัตโนมัติ โดยไม่จำเป็นต้องเขียน close อีกต่อไป

```
try /* resources declaration */ {
    /* statement block */
}
```

ตัวอย่างโปรแกรมโดยอาศัย Try-Catch

```
public class TryResourceDemo {  
    public static void main(String args[]) {  
        Connection con = null;  
        Statement stm = null;  
        ResultSet rs = null;  
        try {  
            String sql = "SELECT * FROM User";  
            con = getConnection();  
            stm = con.createStatement();  
            rs = stm.executeQuery(sql);  
            while(rs.next()) {  
                rs.getString("USERNAME");  
                rs.getString("EMAIL");  
            }  
        } catch (SQLException sqle) {  
        } finally {  
            closeResource(rs, stm, con);  
        }  
    }  
}
```

```
public static void closeResource (ResultSet  
        rs, Statement stm, Connection con) {  
    if (rs != null) {  
        try {  
            rs.close();  
        } catch (SQLException e) {}  
    }  
    if (stm != null) {  
        try {  
            stm.close();  
        } catch (SQLException e) {}  
    }  
    if (con != null) {  
        try {  
            con.close();  
        } catch (SQLException e) {}  
    }  
}
```

ตัวอย่างโปรแกรมโดยอาศัย try-with-resources

```
String sql = "SELECT * FROM UserAccount";
try (Connection con = getConnection();
     Statement stm = con.createStatement();
     ResultSet rs = stm.executeQuery(sql)) {
    while(rs.next()) {
        rs.getString("USERNAME");
        rs.getString("EMAIL");
    }
} catch (SQLException e) { }
```

อ้างอิง : <https://www.lordgift.in.th/2017/01/java-try-with-resouces-java-7.html>

อย่างไรก็ตาม วัตถุที่สร้างในวงเล็บของ Try ต้องเป็นคลาสที่สืบทอดมาจากอินเตอร์เฟส AutoCloseable เท่านั้น และในวงเล็บของ Try ต้องเป็นการประกาศและสร้างวัตถุเท่านั้น

อ้างอิง : <https://docs.oracle.com/javase/8/docs/api/java/lang/AutoCloseable.html>

การจัดการกับ Exception



การจัดการกับ Exception มี 2 แบบ คือ

- ใช้คำสั่ง **try-catch**

- ใช้คำสั่ง **throws** ในการประกาศเมธอดที่จะมีการเรียกใช้เมธอดได ๆ ที่อาจส่งออกเจ็คประเภท Exception ซึ่งแบ่งได้เป็น 3 แบบ
 - Throwing
 - Rethrowing
 - Wrapping

คำสั่ง throws



รูปแบบการใช้คำสั่ง throws มีดังนี้

```
[modifier] return_type methodName([arguments]) throws  
ExceptionType [,ExceptionType2] {  
    ...  
}
```

ตัวอย่าง

```
public void openfile(String s) throws FileNotFoundException {  
    ...  
}
```

คำสั่ง throws

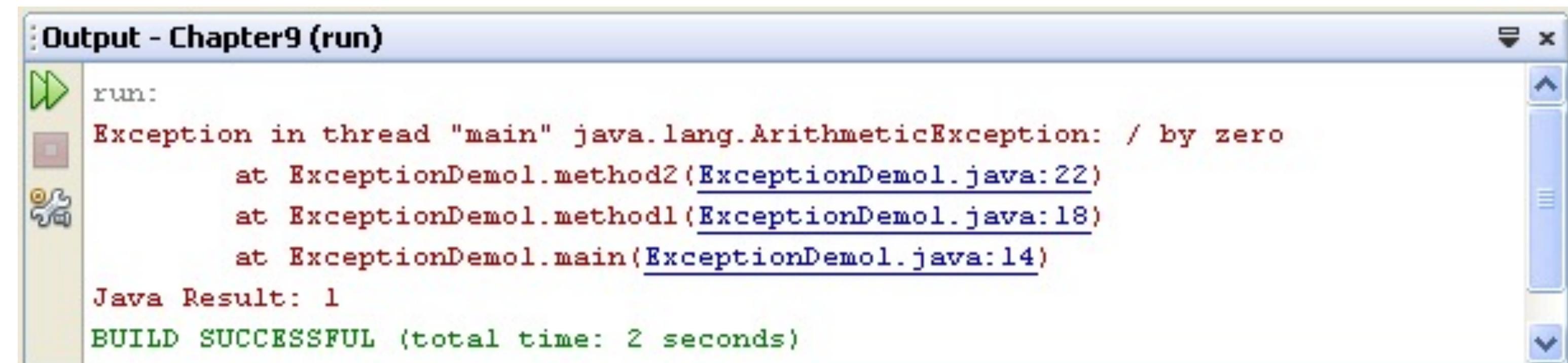
เมธอดใด ๆ สามารถที่จะจัดการกับ Exception โดยใช้คำสั่ง throws ได้มากกว่าหนึ่งประเภท
ตัวอย่าง

```
public void openFile(String s) throws  
FileNotFoundException, UnknownHostException {  
    ...  
}
```

กรณีที่มีการใช้คำสั่ง throws ส่งต่อไปเรื่อย ๆ แล้วเมธอด main() ซึ่งเรียกใช้เมธอดสุดท้ายที่ใช้คำสั่ง throws ไม่มีการจัดการกับอปเจ็คประเภท Exception ดังกล่าว **โปรแกรมจะเกิดข้อผิดพลาดในขั้นตอนการรันโปรแกรม** เมื่อมีข้อผิดพลาดของอปเจ็คประเภท Exception ดังกล่าวเกิดขึ้น

ตัวอย่างโปรแกรมที่ไม่มีการจัดการกับ Exception

```
public class ExceptionDemo1 {  
    public static void main(String args[]) {  
        ExceptionDemo1 ex1 = new ExceptionDemo1();  
        ex1.method1();  
    }  
    public void method1() throws ArithmeticException {  
        method2();  
    }  
    public void method2() throws ArithmeticException {  
        System.out.println(2/0);  
    }  
}
```



The screenshot shows the 'Output - Chapter9 (run)' window of an IDE. It displays the following text:

```
run:  
Exception in thread "main" java.lang.ArithmetiException: / by zero  
    at ExceptionDemo1.method2(ExceptionDemo1.java:22)  
    at ExceptionDemo1.method1(ExceptionDemo1.java:18)  
    at ExceptionDemo1.main(ExceptionDemo1.java:14)  
Java Result: 1  
BUILD SUCCESSFUL (total time: 2 seconds)
```

หลักการ Rethrowing และ Wrapping

การ throwing คือการส่งต่ออปเจ็คประเภท Exception ออกไปสู่ผู้ที่เรียกใช้ให้เป็นคนจัดการกับข้อผิดพลาด ด้วยคำสั่ง throws แล้วยังสามารถทำได้อีก 2 แบบ ได้แก่

- Rethrowing คือ การใช้งาน try-catch ให้ดักจับอปเจ็คประเภท Exception ที่เกิดขึ้นและส่งต่อออกไปด้วย throws อีกทีหนึ่ง โดย ไม่มีการเปลี่ยนแปลงประเภทของอปเจ็ค Exception นั้น
- Wrapping คือ การใช้งาน try-catch ให้ดักจับอปเจ็คประเภท Exception ที่เกิดขึ้นและส่งต่อออกไปด้วย throws อีกทีหนึ่ง โดย มีการเปลี่ยนแปลงประเภทของอปเจ็ค Exception นั้น ใหม่มีความจำเพาะหรือซัดเจนยิ่งขึ้น

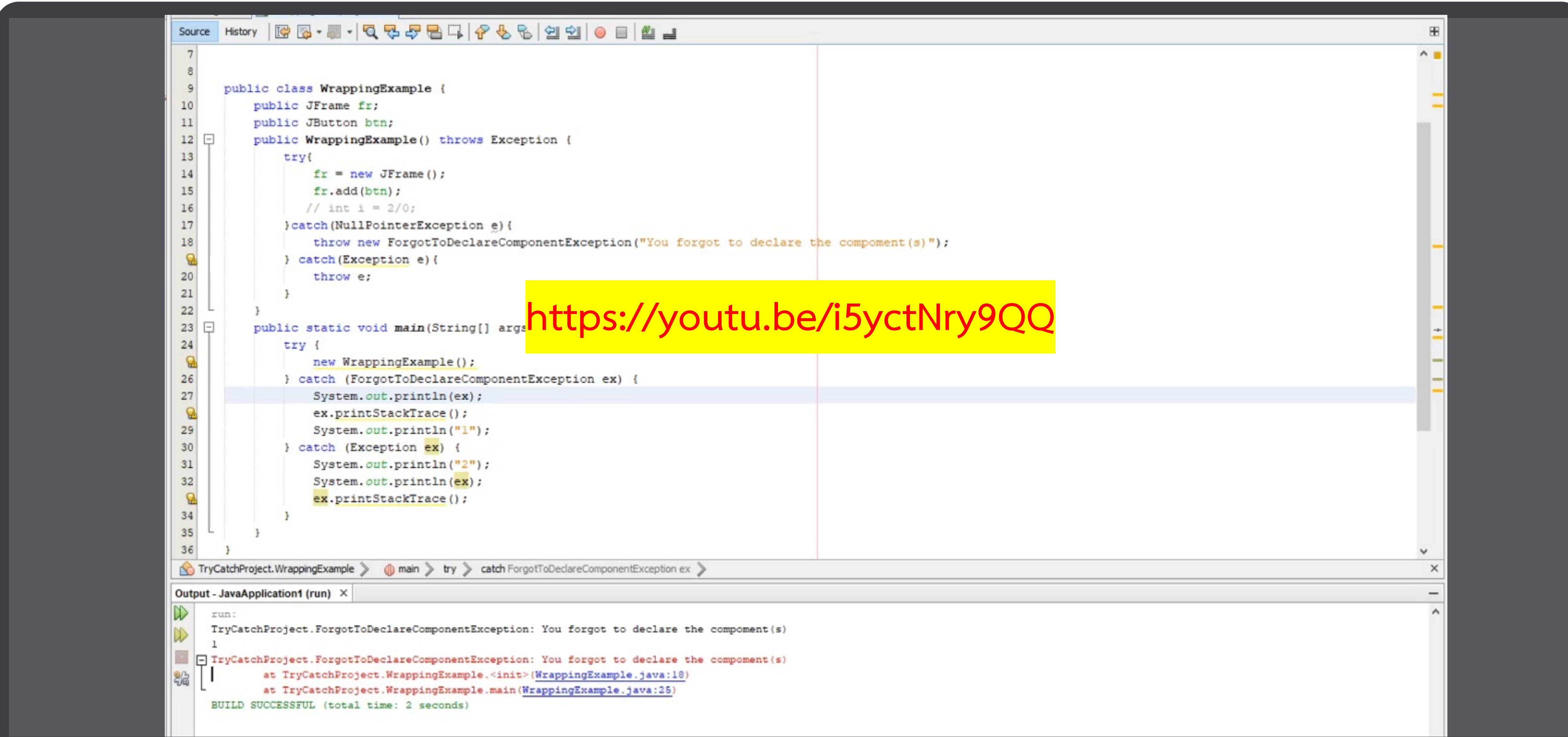
```
public class ForgotToDeclareComponentException extends Exception {  
    public ForgotToDeclareComponentException(String name) {  
        super(name);  
    }  
}
```

หลักการ Rethrowing และ Wrapping



```
public class WrappingExample {  
    public JFrame fr; public JButton btn;  
    public WrappingExample() throws ForgotToDeclareComponentException,Exception {  
        try{  
            fr = new JFrame();  
            fr.add(btn);  
        }catch(NullPointerException e){ // Wrapping  
            throw new ForgotToDeclareComponentException("You forgot to declare the component");  
        } catch(Exception e){ throw e;} // Rethrowing  
    }  
    public static void main(String[] args) {  
        try {  
            new WrappingExample();  
        } catch (ForgotToDeclareComponentException ex) {  
            System.out.println(ex);  
            ex.printStackTrace();  
        } catch (Exception ex) {  
            System.out.println(ex);  
            ex.printStackTrace();  
        }  
    }  
}
```

ข้อสังเกตฯ



The screenshot shows a Java IDE interface with the following details:

- Code Editor:** Displays a Java class named `WrappingExample`. The code demonstrates exception handling, specifically catching `NullPointerException` and other exceptions, and printing stack traces.
- Output Window:** Shows the command line output for running the project. It includes the command `run:`, the exception message `TryCatchProject.ForgotToDeclareComponentException: You forgot to declare the component(s)`, the number `1`, another exception message, and the build log `BUILD SUCCESSFUL (total time: 2 seconds)`.
- Callout Box:** A yellow rectangular box highlights the URL <https://youtu.be/i5yctNry9QQ>, which likely provides a video explanation of the code or concept shown in the IDE.

กฎของการกำหนดเมธอดแบบ overridden

อย่างไรก็ตาม เมธอดแบบ overridden จะไม่อนุญาตให้มีการจัดการอปเจ็คประเภท Exception โดยใช้คำสั่ง throws มากชนิดกว่าที่เมธอดเดิมจัดการอยู่ได้

```
import java.io.*;
public class Main {
    public static void main(String[] args) { new OverrideExceptionV2().myMethods(); }
}
public class Parent {
    public void myMethods() throws FileNotFoundException { }
}
public class OverrideExceptionV2 extends Parent {
    public void myMethods() throws FileNotFoundException, IOException {
        new FileInputStream("temp.txt");
    }
}
```

```
Main.java:16: error: myMethods() in OverrideExceptionV2 cannot override myMethods() in Parent
    public void myMethods() throws FileNotFoundException, IOException {
                           ^
overridden method does not throw IOException
```

หัวข้อ



- ข้อผิดพลาดคืออะไร
- การสร้างคลาสประเภทข้อผิดพลาดขึ้นมาใหม่
- วิธีการจัดการข้อผิดพลาด
- หลักการการจัดการกับข้อผิดพลาด
- ความแตกต่างระหว่าง if-else กับ try-catch

ทำไม่ต้อง Try-Catch แทนที่จะใช้ If-Condition

```
public class Error01{  
    public int [] num = new int[5];  
    public int GetItem(int index){  
        try{  
            return num[index];  
        } catch(ArrayIndexOutOfBoundsException e){  
            return -1;  
        }  
    }  
}
```

```
public class Error02{  
    public int [] num = new int[5];  
    public int GetItem(int index){  
        if(index >= 0 && index < num.length)  
            return num[index];  
        else  
            return -1;  
    }  
}
```



ไม่แนะนำ



แนะนำ

ไม่ควรเลือกใช้ Try-Catch สำหรับ
จัดการ Exception ในกรณี
normal flow of control ถ้า
เป็นไปได้ เว้นแต่เกิดจากการ
ทำงานผิดพลาดของระบบ
(system failures) หรือการจัดการ
ที่ผิดพลาดในบางกรณีที่ไม่แน่นอน
(operations with potential
race conditions) อาทิเช่น เรา
ควรเขียนโค้ดเพื่อตรวจสอบลำดับ
การอ้างอิงถึงสมาชิกใน Array ก่อน
แทนที่จะ Try-Catch หรือ
Throws ออกเจ็คของ Exception
นั้นออกมา

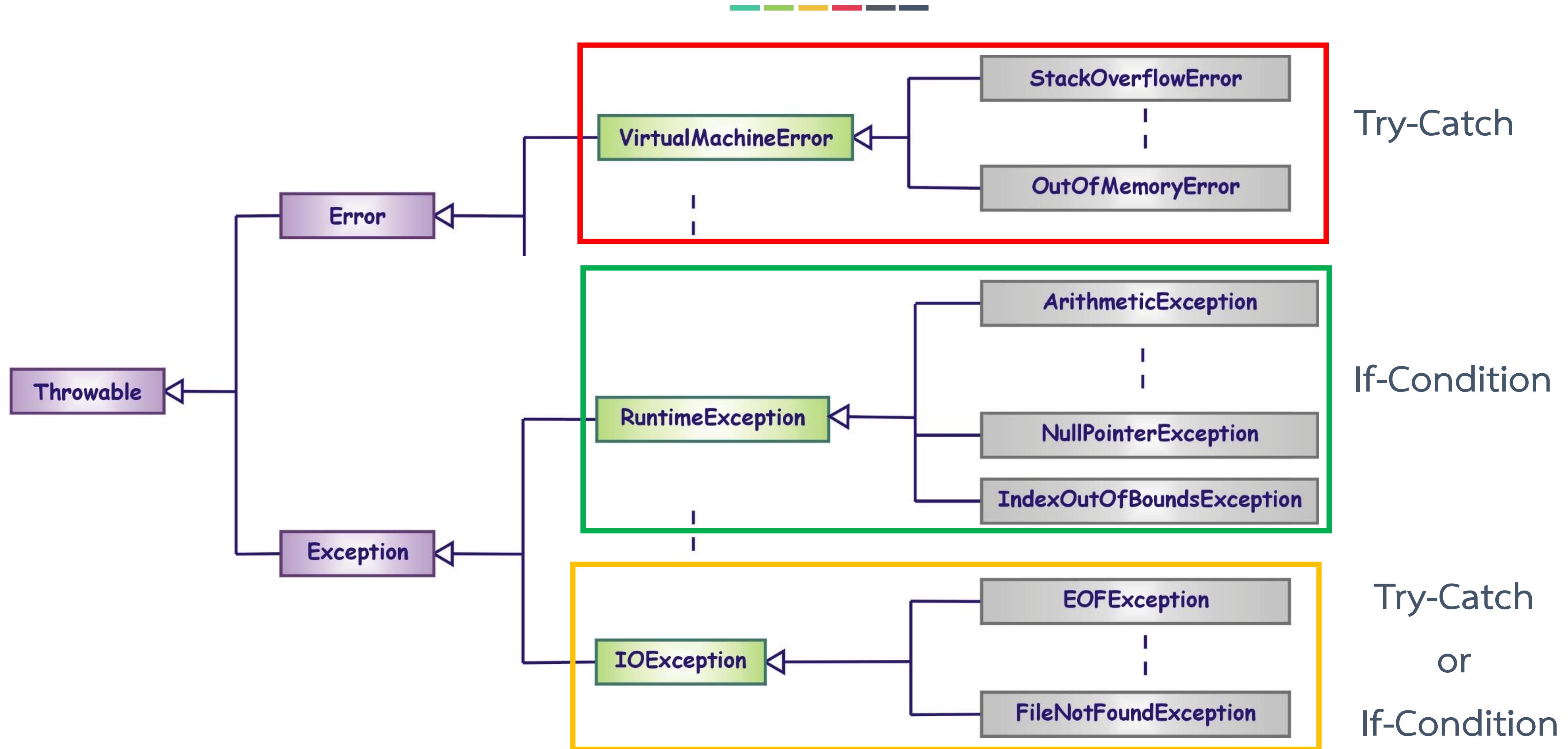
ทำไมต้อง Try-Catch แทนที่จะใช้ If-Condition

โดยทั่วไปแล้ว เราควรจะเลือกใช้ Try-Catch จัดการ Exception สำหรับส่วนที่มีการเชื่อมต่อกับปัจจัยภายนอก โปรแกรม หรือมีการเรียกใช้งานทรัพยากรภายนอก อาทิเช่น อ่าน/เขียนไฟล์, เชื่อมต่อเครื่องแม่ข่าย, เชื่อมต่ออินเตอร์เน็ต, เรียกใช้ API ข้างนอก

อย่างไรก็ตาม นักศึกษาต้องพึงพิจารณาไว้ว่าการจัดการ Exception ถือว่าเป็น heavy and expensive operation ทางด้านประสิทธิภาพ ถ้าสามารถเลือกใช้ IF-Condition แทนได้จะทำให้มีประสิทธิภาพที่ดีกว่า ซึ่งสามารถสรุปได้ดังตารางต่อไปนี้

IF-ELSE	TRY-CATCH
<ul style="list-style-type: none">It checks any condition is true or not, if true then execute the code inside the if block otherwise execute else block.‘if-else’ use to handle different conditions using condition checking.‘if-else’ is less time consuming than ‘try-catch’.‘if-else’ communicate between the data provided to the program and the condition.	<ul style="list-style-type: none">‘try’ is a section where a code is defined for tested whether the code generates an unexpected result while executing, if any unexpected result found catch block is executed to handle that situation.In case of ‘try-catch’, the system will check the system generated errors or exception during a executing process or a task.‘try-catch’ is more time consuming than ‘if-else’.‘try-catch’ communicate between the data with the conditions and the system.

ทำไมต้อง Try-Catch แทนที่จะใช้ If-Condition



เปรียบเทียบการใช้ Try-Catch กับการเขียน If-Condition



```
public class Error03{  
    public Channel channel; // สมมติ  
    public void Close()  
    {  
        if (this.channel.State.equals(CommunicationState.Opened))  
        {  
            try {  
                this.channel.Close();  
            } catch (CommunicationException e) {  
  
                // Do Something  
            } catch (TimeoutException e) {  
                // Do Something  
            }  
        }  
    }  
}
```

สรุปเนื้อหาของบท



- ข้อดีประเทหนึ่งของภาษาจาวา คือ เราสามารถเขียนโปรแกรมให้มีการตรวจจับและจัดการกับข้อผิดพลาดที่อาจเกิดขึ้นได้ โดยที่การทำงานไม่ต้องหยุดลง
- Error เป็นข้อผิดพลาดที่ไม่สามารถแก้ไขและจัดการได้
- Exception เป็นข้อผิดพลาดที่สามารถแก้ไขหรือจัดการได้
- คำสั่ง try และ catch เป็นคำสั่งที่ใช้ในการตรวจจับและจัดการกับข้อผิดพลาดที่อาจเกิดขึ้นได้โดยบล็อกคำสั่ง catch สามารถมีได้มากกว่าหนึ่งบล็อกสำหรับในแต่ละบล็อกคำสั่ง try
- คำสั่ง finally เป็นคำสั่งที่อยู่ต่อจากคำสั่ง try/catch จะถูกเรียกใช้เสมอ ยกเว้นเจอคำสั่ง System.exit(0); ก่อนเท่านั้น

สรุปเนื้อหาของบท



- ▶ คำสั่ง throws จะใส่ไว้ตรงคำสั่งประการเมื่อต้องการจัดการกับข้อผิดพลาด แต่จะให้มีเมื่อต้องการใช้เมื่อต้องการนี้เป็นตัวจัดการแทน
- ▶ เราสามารถสร้างคลาสประเภท Exception ชนิดใหม่ขึ้นได้ โดยจะต้องสืบทอดมาจากคลาสที่ชื่อ Exception และต้องมีการเรียกใช้ Constructor ของคลาส Exception ด้วย