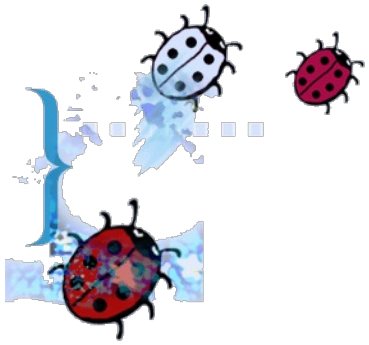
Java Exceptions



By Assoc. Prof. Rangsit Sirirangsi

Java's Exceptions

- เป็นข้อผิดพลาดที่เกิดขึ้นระหว่างการประมวลผลและมีผลทำให้ลำดับการทำงานของชุดคำสั่งภายในโปรแกรมเปลี่ยนไป ซึ่งความผิดปกติดังกล่าวอาจก่อให้เกิด error ภายในโปรแกรม
- Errors ในจาวาสามารถแบ่งออกได้เป็น 2 กลุ่ม ได้แก่
  - **Compile-time errors** เกิดขึ้นจากการโปรแกรมที่ผิด syntax ของโปรแกรมภาษาที่ใช้
  - **Run-time errors** เกิดขึ้นระหว่างการประมวลผลโปรแกรม
- Exception เป็นสิ่งผิดปกติที่เกิดขึ้นในช่วงเวลารันไทม์ (Runtime) ระหว่างการประมวลผลโปรแกรม

What exceptions can occur?

- ตัวอย่างของการจัดการ exceptions ของการรับค่าอินพุตแบบตัวเลข

```
int score = in.nextInt();
```

- ในกรณีที่ผู้ใช้ระบุค่าเป็น “**abc123**”

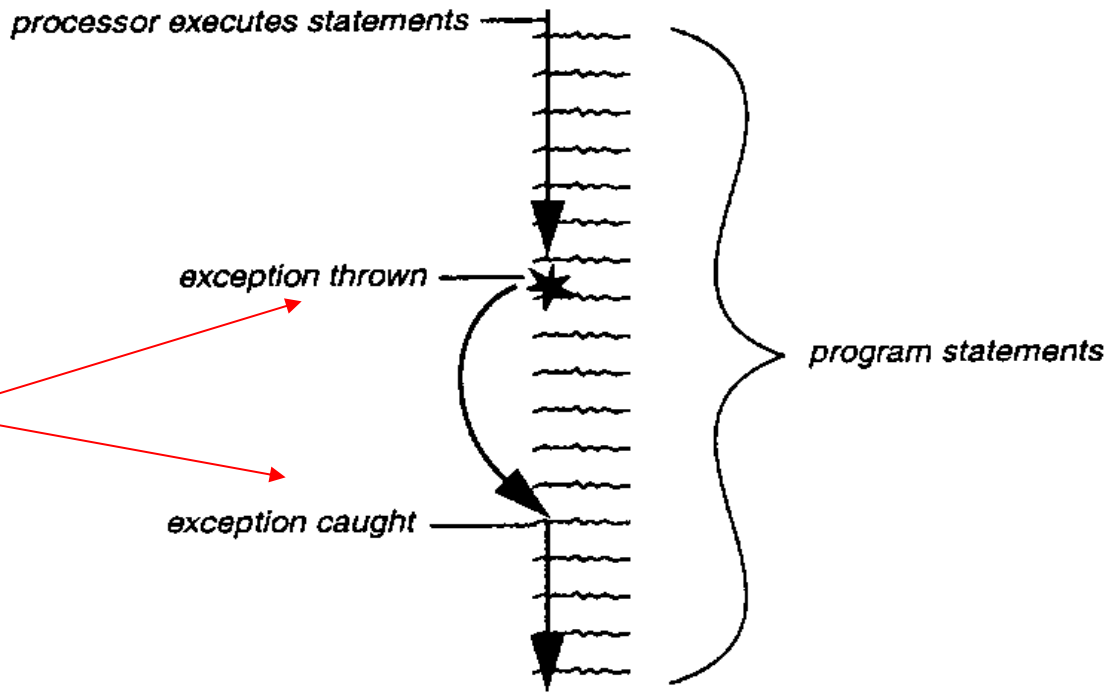
```
Exception in thread "main"    java.util.InputMismatchException
 at java.util.Scanner.throwFor(Scanner.java:819)
 at java.util.Scanner.next(Scanner.java:1431)
 at java.util.Scanner.nextInt(Scanner.java:2040)
 at java.util.Scanner.nextInt(Scanner.java:2000)
 at Test.main(Test.java:5)
```

- จาวาตรวจสอบพบความผิดปกติและนำเสนอผ่าน **InputMismatchException**
- เนื่องจาก String ไม่สามารถแปลงให้เป็น integer ได้
- เมื่อจาวาจัดการความผิดปกติดังกล่าว บ่อยครั้งที่ทำให้โปรแกรมสิ้นสุดการทำงานทันที (crash)

Java exception mechanism

- Run-time system ทำหน้าที่แจ้งให้ทราบว่าเกิด error โดยการ ***throwing an exception***

โปรเซสเซอร์จะหยุดการประมวลผลคำสั่งปัจจุบันตามลำดับ และเริ่มประมวลผลในคำสั่งอื่น ๆ ที่เป็น exception handler

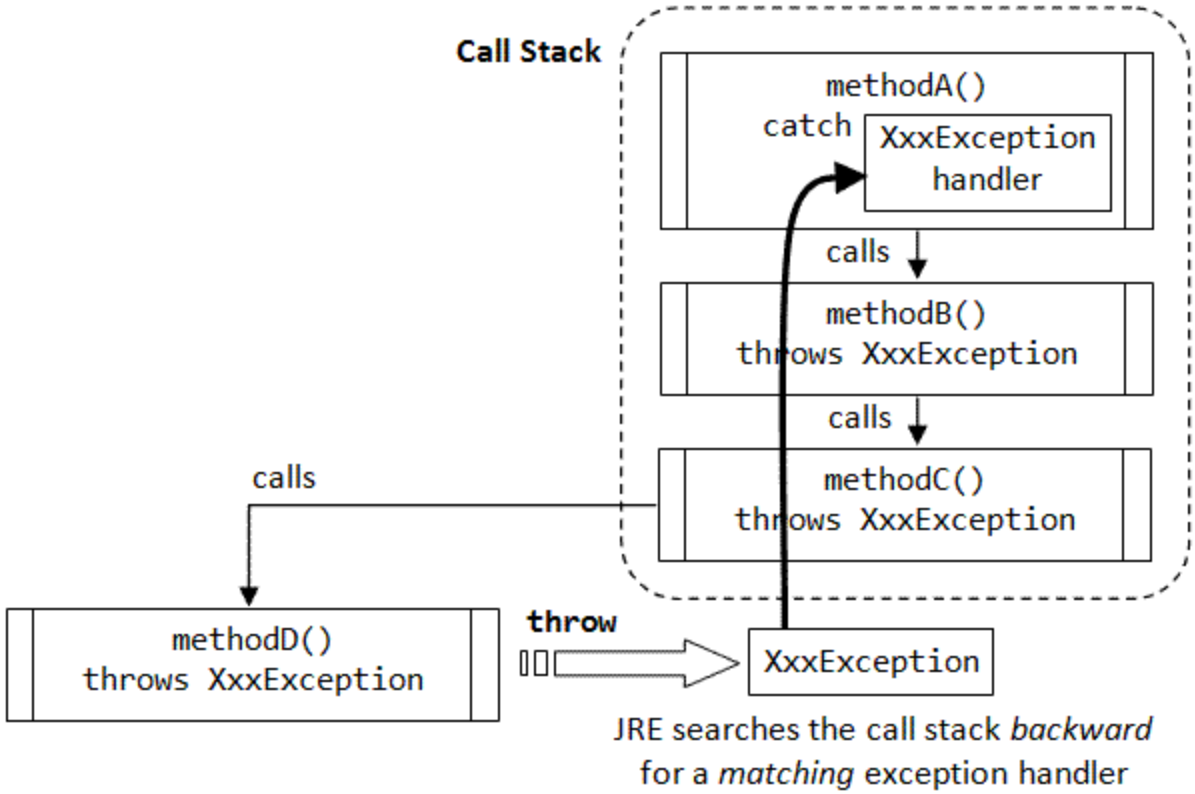


Exception Process

1. เมื่อมีความผิดปกติเกิดขึ้นภายในเมธอดจาวาจะสร้าง Exception ออปเจค
2. ผ่านค่าออปเจคไปยัง JRE (เรียกว่า throw exception)
3. Exception ออปเจคประกอบไปด้วยชนิดของ Exception และข้อมูลที่เกี่ยวข้องกับความผิดปกติที่เกิดขึ้น
4. JRE รับหน้าที่ในการค้นหากลไกการจัดการ Exception ที่เกิดขึ้น
5. โดยการไปค้นหาใน Call Stack Unit เพื่อหากลไกการจัดการที่เหมาะสม (ในกรณีนี้เรียกว่า catch)
6. ส่วนในกรณีที่ไม่พบกลไกดังกล่าวโปรแกรมจะสิ้นสุดการทำงานทันที

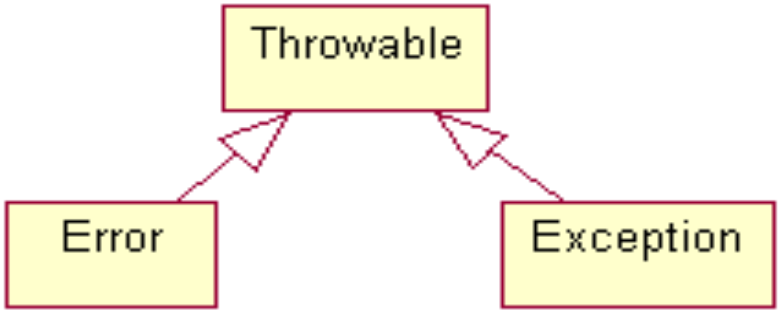
Exception Process

- เมื่อมีความผิดปกติเกิดขึ้นใน methodD() และ throw XxxException ไปยัง JRE เพื่อค้นหากลการจัดการที่เหมาะสม ในรูปนี้ได้แก่ methodA()

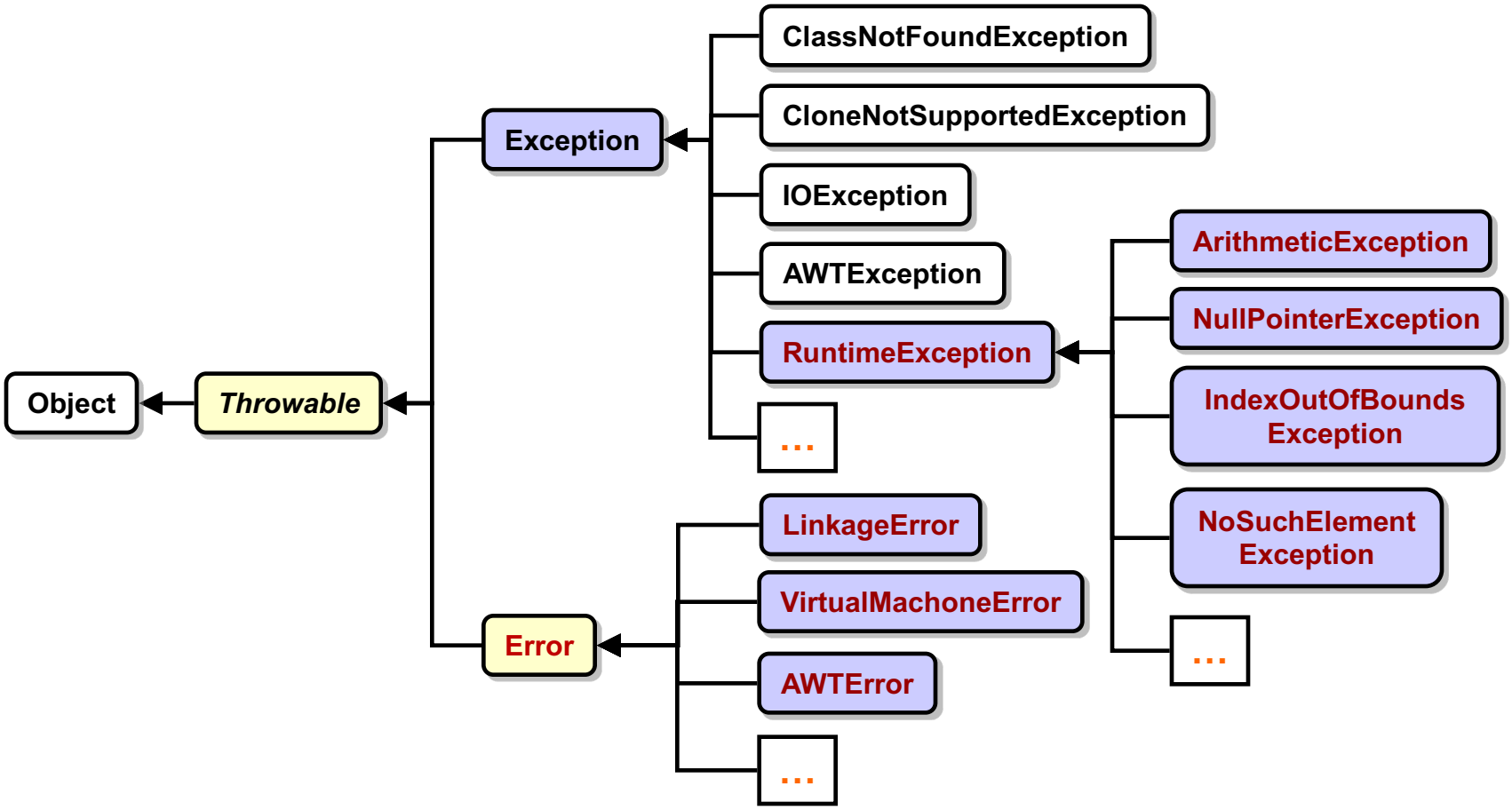


Exception classes

- จาวาจัดการความผิดปกติโดยใช้คลาส Throwable ประกอบด้วยคลาสสืบทอด
  - **Errors** เป็นความผิดพลาดร้ายแรงที่เกิดขึ้นจากกลไกภายใน เช่น JVM หรือ Linker และมีผลทำให้โปรแกรมสิ้นสุดการทำงานลงทันที จัดการไม่ได้
  - **Exceptions** เป็นความผิดปกติไม่ร้ายแรงเกิดขึ้นภายในโปรแกรม สามารถตรวจสอบและเขียนโค้ดเพื่อจัดการได้ ในกรณีที่ไม่มีการตรวจสอบและจัดการอาจก่อให้เกิดการสิ้นสุดการทำงานแบบผิดปกติ



Exceptions Hierarchy



SubClass from Exception

- ArithmeticException หารค่าตัวเลขด้วย 0 หรือปัญหาที่เกิดจากจากการคำนวณทางคณิตศาสตร์
- ArrayIndexOutOfBoundsException เมื่อค่าดัชนีน้อยกว่า 0 หรือมากกว่าความยาวที่แท้จริงของอะเรย์
- IndexOutOfBoundsException ค่าดัชนีของอะเรย์อยู่ภยนอกขอบเขต
- NullPointerException Reference ไปยังออปเจคซึ่งยังไม่มีการกำหนดค่า
- NumberFormatException การแปลงค่าเกิดความผิดพลาด โดยปกติจะมาจาก Integer.parseInt หรือ Double.parseDouble
- IllegalArgumentException เรียกใช้เมธอดที่มีจำนวนอาร์กิวเมนต์ไม่ถูกต้อง

Out of Bounds

- Java runtime ทำหน้าที่ค้นหาโค้ดที่ใช้ในการจัดการกับความผิดพลาดที่เกิดขึ้น การสร้าง exception ออปเจคและจัดการ โดย runtime system จะถูกเรียกว่าการ *throwing an exception*.
- Java runtime ตรวจสอบขอบเขตการเรียกใช้อาร์เรย์ เช่น

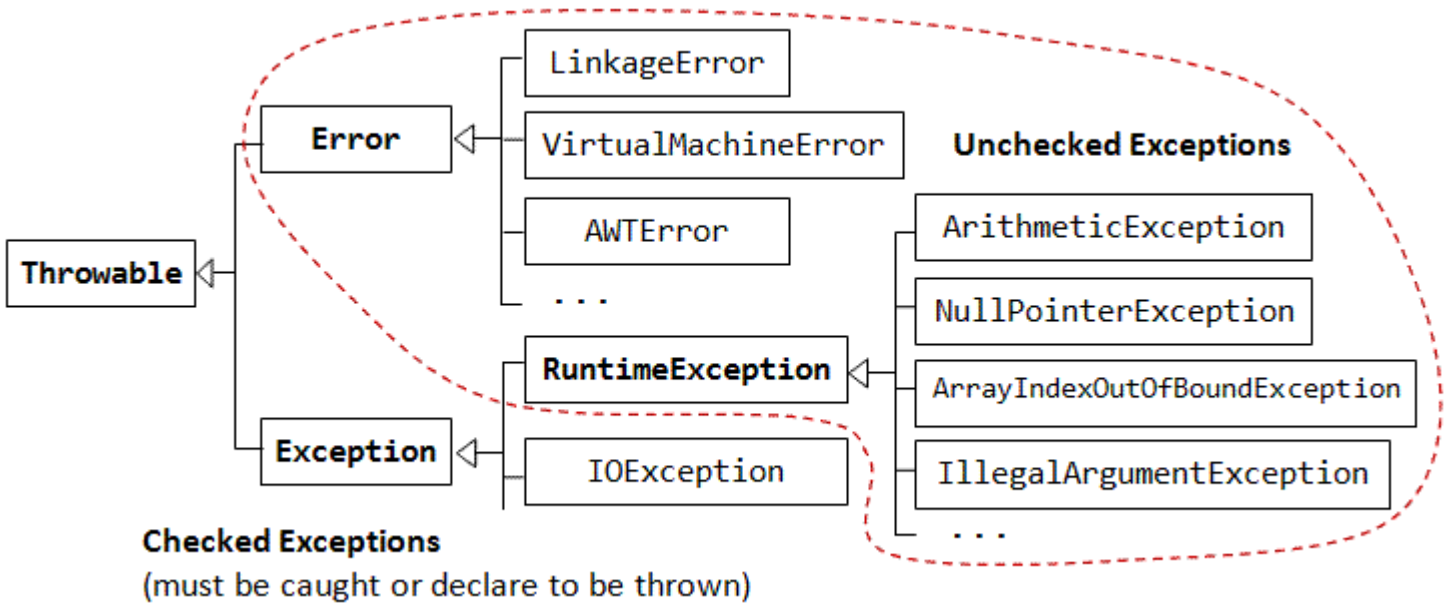
```
class  ArrayOutOfBounds   {
    public static void main( String args[] )  {
        int  students[] = new int[5];
        students[5]  =  1;
        System.out.println( " The End" );
    }
}
```

Output

```
java.lang.ArrayIndexOutOfBoundsException: 10
at ArrayOutofBounds.main(All.java:8)
```

Types of Exceptions

- Exception ประกอบไปด้วยคลาสสืบทอด 2 แบบ ได้แก่
  - ***unchecked*** สืบทอดมาจากคลาส *RuntimeException*
  - ***checked*** สืบทอดมาจากคลาส Exception **ยกเว้น**คลาสที่สืบทอดมาจากคลาส RuntimeException



Unchecked exceptions

- ***unchecked*** เป็นความผิดปกติที่เกิดขึ้นในช่วงเวลารันไทม์ (Runtime) โดยคอมไพเลอร์ไม่สามารถตรวจสอบพบได้ในช่วงเวลาคอมไพล์โปรแกรม แต่จะตรวจสอบพบได้ในช่วงเวลารันไทม์จากคลาส RuntimeException
- ดังนั้นจึงไม่จำเป็นต้องตรวจสอบและ throw Exception ภายในโปรแกรม
- Exception แบบนี้โปรแกรมเมอร์เลือกที่จะจัดการ Exceptions หรือไม่ก็ได้
  - ในกรณีที่ไม่มีการจัดการใด ๆ เมื่อ Exception ประเภทนี้เกิดขึ้น ส่วนใหญ่จะมีผลทำให้โปรแกรมสิ้นสุดการทำงาน
  - ผู้ใช้ส่วนใหญ่เลือกใช้การจัดการ Exception โดยจาวา ซึ่งได้แก่ การเรียกใช้ RuntimeException

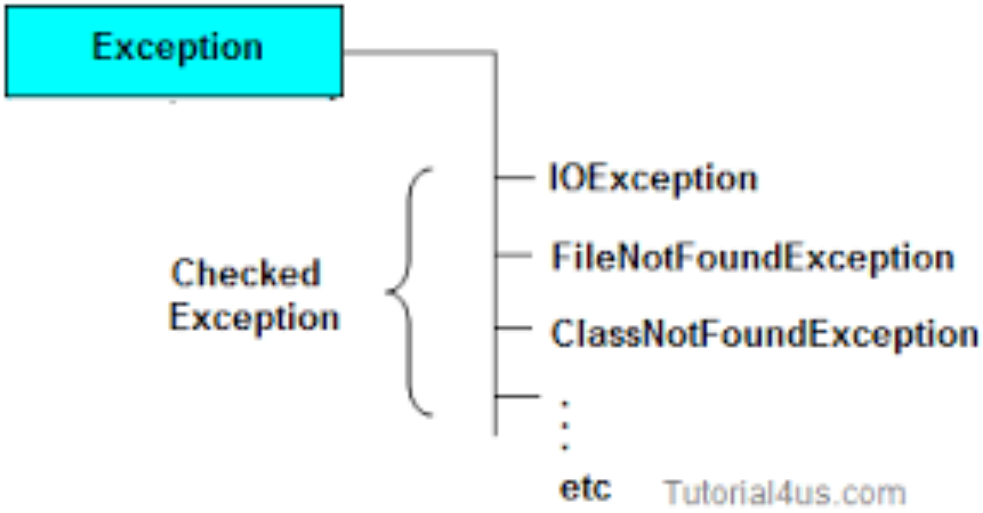
Unchecked Exception

```
public class Unchecked {

    public static void main(String[] args) {
        int num1 = 10;
        int num2 = 0;

        int result = num1/num2;
        System.out.println(result);
    }
}
```

Exception in thread "main" java.lang.ArithmeticException: / by zero
at Unchecked.main(Uncheck.java:9)

Checked Exception

- ***checked*** เป็นความผิดปกติที่เกิดขึ้นในช่วงเวลาคอมไพล์ จาวาคอมไพเลอร์ตรวจสอบได้ หากไม่จัดการจะคอมไพล์ไม่ผ่าน
- ดังนั้นเมื่อเกิดความผิดปกติขึ้นในรูปของ checked exception (เช่น IOException) ภายในเมธอด โปรแกรมเมอร์จำเป็นต้องจัดการ exception ดังกล่าวด้วยวิธีการหนึ่ง ๆ เสมอ



Checked Exception

```
class FileNotFound {

    public static void main(String[] args) throws IOException {
        FileInputStream fis = new FileInputStream("B:/myfile.txt");
        int k;

        while(( k = fis.read() ) != -1)
        {
            System.out.print((char)k);
        }
        fis.close();
    }
}
```

```
Exception in thread "main" java.io.FileNotFoundException: B:\myfile.txt (The
system cannot find the path specified)
at java.io.FileInputStream.open(Native Method)
at java.io.FileInputStream.<init>(Unknown Source)
at java.io.FileInputStream.<init>(Unknown Source)
at FileNotFound.main(FileNotFound.java:10)
```

Handling Exceptions

- ถ้าไม่มีการเขียนคำสั่งเพื่อจัดการ Exception โปรแกรมจะสิ้นสุดลงทันที
- ดังนั้นการจัดการ Exception ช่วยให้โปรแกรมทราบความผิดปกติและรันต่อไปได้
- วิธีการในการจัดการ exceptions สามารถทำได้ดังนี้
  - เลือกใช้การจัดการ exception ที่มีอยู่แล้วในจาวา (ไม่จัดการ)
  - ใช้ throws statement เพื่อจัดการกับ exception ภายในเมธอดที่ต้องการ
  - ใช้กลไกที่เรียกว่า try-catch ภายในเมธอด
  - ใช้วิธีการโยน (throws) ให้เมธอดอื่นจัดการแทน
    - หากไม่มีวิธีที่เหมาะสมในการจัดการ exception ในตำแหน่งที่เกิดขึ้น การจัดการความผิดปกติดังกล่าวอาจผ่านไปในระดับที่สูงขึ้นถัดไป

Throwing Exceptions

- ขั้นตอนการเรียกใช้ throws statement ดังต่อไปนี้
  - เลือกใช้ชนิดของ Exception ออปเจคที่ต้องการ throw
  - ทดสอบเงื่อนไขตามข้อกำหนดในโปรแกรม
  - throw Exception ออปเจคเพื่อแสดงถึงความผิดปกติที่เกิดขึ้น

```
public void myMethod() throws Exception {
        if( condition)
            throw new Exception("Message to display");
}
```

- เมื่อจาวาพบความผิดปกติเมธอด myMethod() จะสร้าง Exception ที่เหมาะสม และผ่านค่าไปยัง JRE ด้วยคำสั่ง throws Exception
- หมายเหตุ ในการประกาศเมธอดใช้คีย์เวิร์ด "throws" ส่วนที่อยู่ภายในเมธอดใช้คีย์เวิร์ด throw

Throw Statement Example

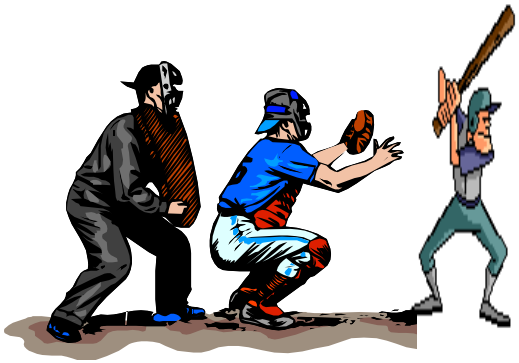
```
class Rectangle  {
     private int length;
     private int width;

     public Rectangle(int length, int width) throws Exception  {
      if (width == 0)
        throw new Exception("Width must not be 0");
      this.length = length;
      this.width = width;
     }
    public static void main( String args[] ) throws Exception  {
      Rectangle temp = new Rectangle(12, 0);
  }
}
```

```
Exception in thread "main" java.lang.Exception: Width must not be 0
    at Rectangle.<init>(Rectangle.java:8)
    at Rectangle.main(Rectangle.java:13)
```

Catching an Exception

Use

try,
catch



to

watch for
indicate
handle

} exceptions

```
try{
  code that may throw exceptions

}

catch (ExceptionType ref) {
   exception handling code
}
```

Basics of the try & catch Exception

- เป็นการจัดการความผิดพลาดอีกวิธีหนึ่งจะสามารถทำได้โดยใช้คีย์เวิร์ดประเภท try, throw, catch ตามลำดับ
- โค้ดส่วนที่อาจมีผลทำให้เกิด exception ถูกกำหนดไว้ภายในส่วนที่เรียกว่า try block
- ส่วนที่เป็น catch block จะประกอบไปด้วยโค้ดที่ใช้ในการจัดการ exception ที่เกิดขึ้นภายใน try block
- ส่วนคีย์เวิร์ด throw ใช้สำหรับการส่งกลไกควบคุมไปยังส่วนที่เป็น catch exception (thrown)
- ส่วนของ catch block อาจมีได้มากกว่าหนึ่งต่อ 1 try Block ทั้งนี้เพื่อให้สามารถใช้ได้กับ exceptions หลายชนิด

The try & catch Exception

- รูปแบบของ **try** block จะประกอบด้วยคีย์เวิร์ด **try** ตามด้วยชุดคำสั่งตั้งแต่หนึ่งหรือมากกว่าที่อยู่ภายในเครื่องหมายปีกกา

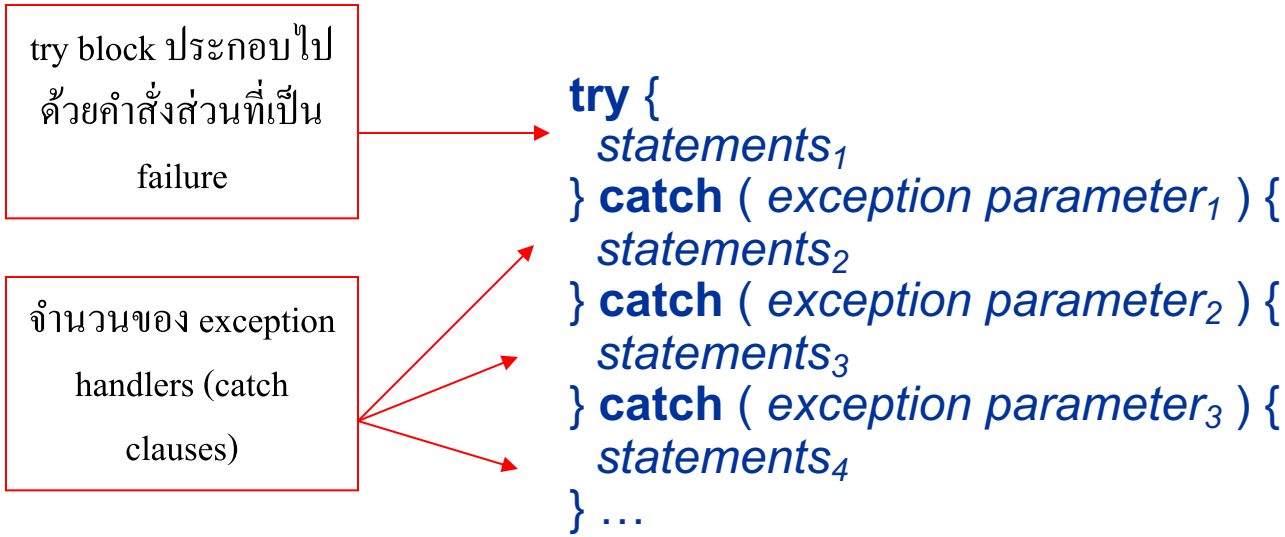
```
try{
  //java statements
}//end try block
```

- โค้ดที่ใช้สำหรับจัดการ Exception จะถูกกำหนดไว้ภายใน catch blocks ซึ่งเริ่มต้นด้วยคีย์เวิร์ด catch ตามด้วยชนิดของ exception ตามรูปแบบดังนี้

```
catch(ThrowableObjectType paramName){
  //Java statements to handle the exception
} //end catch
```

Catching exceptions

- Exceptions จะถูกจัดการหรือ “caught” โดยใช้ **try-catch** statement

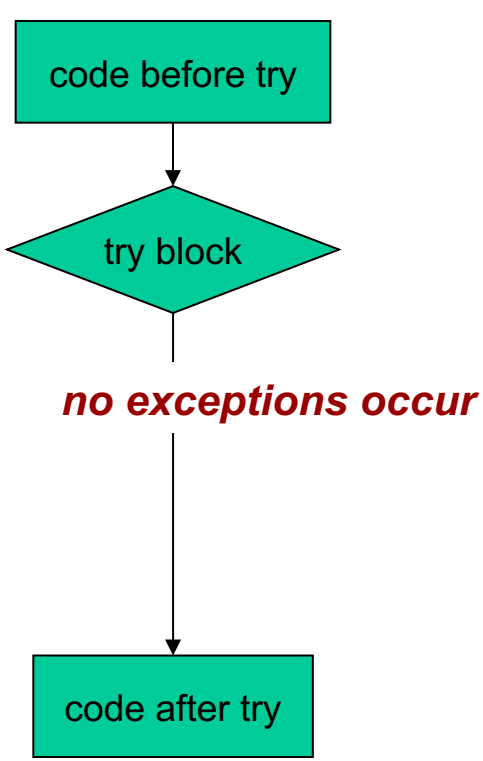


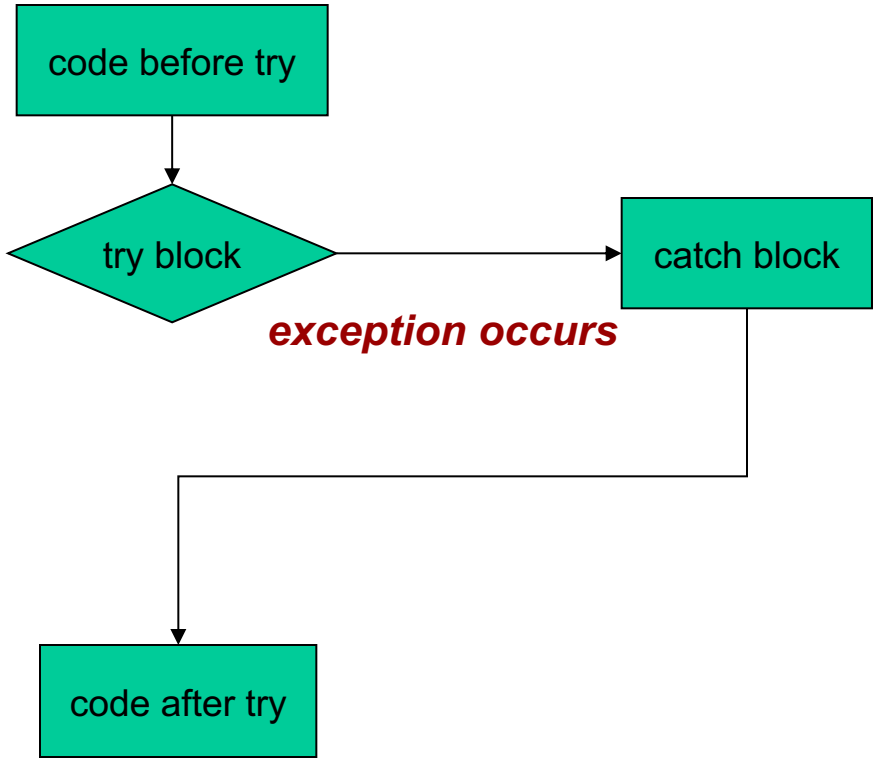
- แต่ละ *catch clause* จะมีพารามิเตอร์ที่เป็น type Exception หรือ subclasses

The catch block(s)

- **try** block จะต้องใช้ร่วมกับอย่างน้อย 1 **catch** block (หรือ 1 **finally** block) มิฉะนั้นคอมไพเลอร์จะแจ้งความผิดพลาดเสมอ
- ส่วนที่เป็น **try** และ **catch** block จะต้องไม่มีโค้ดอื่นใดแทรกอยู่ระหว่างสองส่วนได้
- หากไม่มีการ thrown exception
  - ส่วนที่เป็นโค้ดสำหรับจัดการ Exception จะถูกข้ามไป
  - การควบคุมจะกลับไปสู่ตำแหน่งภายหลัง **catch** blocks

try-catch Control Flow





Without Exception Handling


```
import java.util.Scanner;
public class NoTryCatch {

   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      String str = sc.next();
      int i = Integer.parseInt(str);
      System.out.println(i);
   }
}
```

```
10.5
Exception in thread "main" java.lang.NumberFormatException: For input
string: "10.5“ at
java.lang.NumberFormatException.forInputString(NumberFormatExceptio
n.java:48)
at java.lang.Integer.parseInt(Integer.java:456)
at java.lang.Integer.parseInt(Integer.java:497)
at NoTryCatch.main(NoTryCatch.java:8)
```

With try-catch Exception Handling


```
import java.util.Scanner;

public class WithTryCatch {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String str = sc.next();
        int i = 0;

        try {
            i = Integer.parseInt(str);
        } catch (Exception e) {
            System.out.println("Catch exception");
        }
        System.out.println(i);
    }
}
```

10.5

Catch exception

0

Try & catch Exception : Example


```
class ExceptionDemo {
  public int convert(String s)     {
     int result = 0 ;
     try       {
       result = Integer.parseInt(s) ;
      }
     catch (NumberFormatException e)       {
         System.out.println("String conversion  failed: " + e) ;
      }
     return result ;
  }
  public static void main(String[] args)    {
        ExceptionDemo e1 = new ExceptionDemo() ;
        e1.convert("123") ;
        e1.convert("abc") ;
    }
}
```

String conversion failed: java.lang.NumberFormatException: For input string: "abc"

Handling exceptions

```
class  HandlerSimple
{
        public  static  void  change( int[] course  )
        {
          System.out.println( "Start Change"  );
          course[ 10 ]  =  1;
          System.out.println( "End Change"  );
        }

        public  static  void  main( String args[] )  {
          int  students[] = new int[5];
          try  {
            System.out.println( "Start Try"  );
            change( students );
            System.out.println(  "After change"  );
         }
          catch  (ArrayIndexOutOfBoundsException e)
          {
            System.err.println( "OutOfBounds: "  +  e.getMessage());
          }
         System.out.println(  "After try "  );
        }
}
```

```
//  Start Try
//   Start Change
//   OutOfBounds: 10
//   After try
```

No Exceptions Handling

- ในกรณีที่ไม่มี try block

```
class  NoHandler  {

   public  static  void  change( int[] course )
   {
    System.out.println( "Start Change"  );
       course[ 10 ]  =  1;
       System.out.println( "End Change"  );
   }

   public  static  void  main( String args[] ) {
      int  students[] = new int[5];
      System.out.println( "Start Try"  );
      change( students );
      System.out.println(  "After change"  );
  }
}
```

```
Start Try
Start Change
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
    at Nohandler.change(NoHandler.java:6)
    at NoHandler.main(NoHandler.java:12)
```

Methods of the Throwable class

StackTrace

- `fillInStackTrace()`
- `getStackTrace()`
- `printStackTrace()`
- `setStackTrace(StackTraceElement[] stackTrace)`

Messages

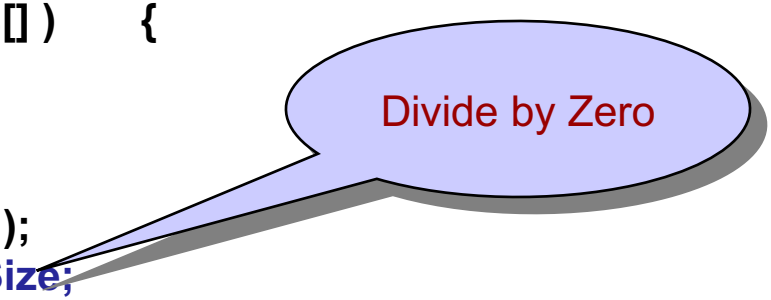
- `getLocalizedMessage()`
- `getMessage()toString()`

StackTrace & Messages

- **StackTrace** เป็นรายการของเมธอดที่ถูกประมวลผลตามลำดับที่ก่อให้เกิดการ exception. (ในกรณีจะมองเห็นได้จากผลลัพธ์ในการรันของโปรแกรมเมื่อเกิด runtime error ที่ไม่มีเมธอดสำหรับจัดการรองรับ)
- **เมธอด** printStackTrace() ใช้สำหรับแสดง stack trace เมื่อมี exception หรือ error เกิดขึ้น
- **Messages** เป็นกลไกที่ใช้ในการรับค่าในรูปของ **String** จาก exception ออปเจค
- เนื่องจากทุก ๆ exception ออปเจคจะสืบทอดมาจากคลาส **Throwable** ดังนั้นเมธอดเหล่านี้จึงสามารถนำไปใช้ได้ในส่วนของโค้ดที่ใช้ในการจัดการ exception
- ตัวอย่างเช่น ภายใน exception ออปเจคจะประกอบไปด้วยข้อมูลของ Exception ที่สามารถแสดงรายละเอียดค่าออกมาได้โดยใช้เมธอด **getMessage()**

Multiple Exceptions

```
class  MultipleExceptions {
     public static void main( String args[] )      {
       int  students[] = new int[5];
       int  classSize  =  0;
       try          {
         System.out.println( "Start Try"  );
         int  classAverage  =  10 / classSize;
         System.out.println( "After class average"  );
         students[10]  =  1;
         System.out.println(  "After array statement"  );     }
       catch  (ArrayIndexOutOfBoundsException e)     {
          System.err.println( "OutOfBounds: "  +  e.getMessage());
       }
       catch  (  ArithmeticException  e  )  {
            System.err.println( " Math Error"  + e.getMessage() );
        }
      System.out.println(  " After try " );
      }
}
```

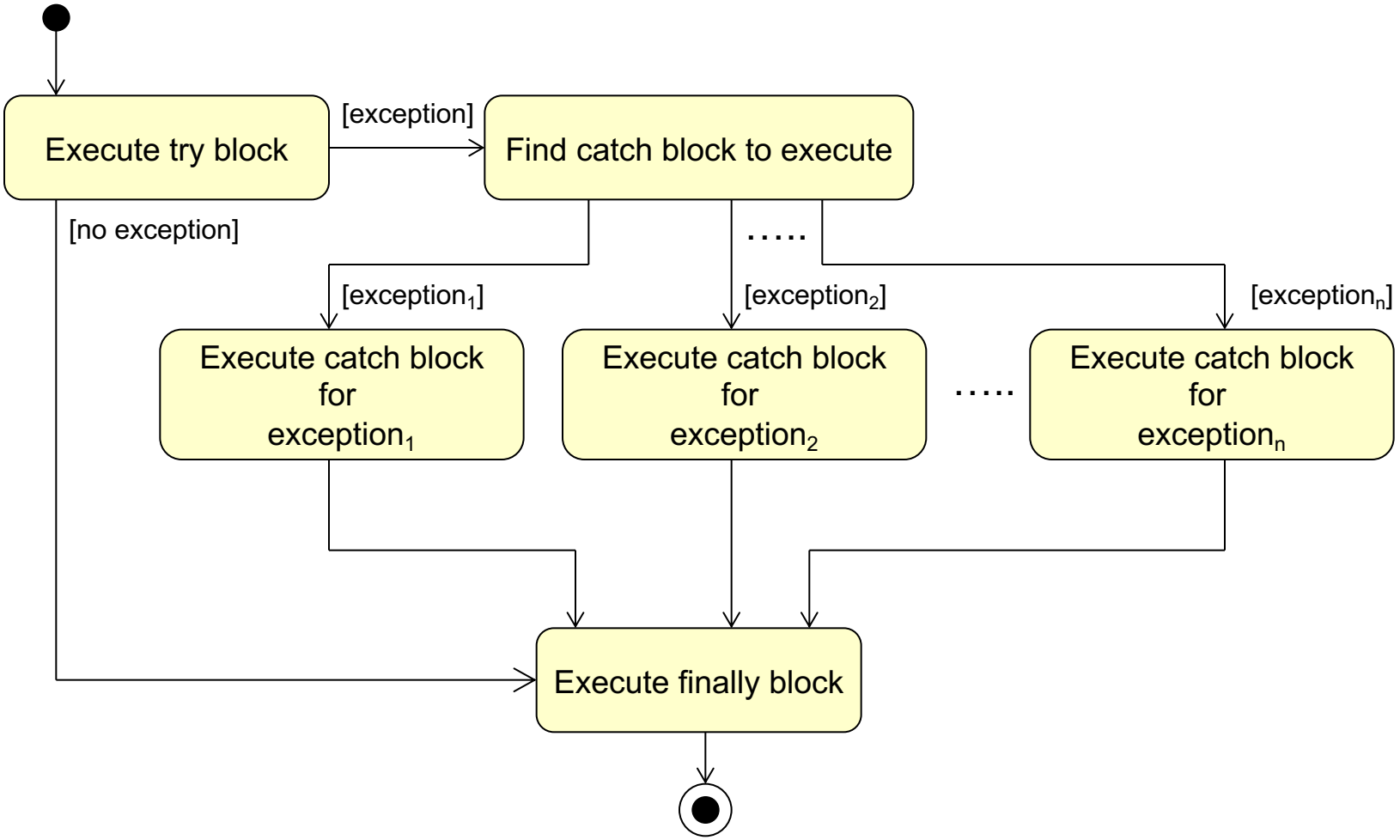


// Start Try // Math Error/ by zero // After try

The finally block

- finally block เป็นกลไกที่ยอมให้เมธอดที่กำหนดโดยผู้ใช้สามารถ clean up ค่าที่ต้องการโดยไม่สนใจกับสิ่งที่เกิดขึ้นใน try block
- ในกรณีที่ไม่มีการ thrown exceptions โค้ดภายใน **catch** blocks จะไม่ถูกประมวลผล แต่การประมวลผลจะเกิดขึ้นที่ **finally** block
- ในกรณีที่มีการ thrown exception และโค้ดภายใน **catch** blocks จะถูกประมวลผลจนเสร็จสิ้น จากนั้น control จะถูกส่งไปยัง **finally** block เพื่อประมวลผลโค้ดในส่วนของ **finally** block ต่อไป
- การใช้ finally block จึงเหมาะสมกับงานประเภทปิดไฟล์หรือคืนค่าทรัพยากรกลับสู่ระบบ

try-catch-finally block



The try-catch-finally statement

- หากต้องการประมวลผลโค้ดบางส่วน ไม่ว่า exception จะเกิดขึ้นหรือไม่ ผู้ใช้สามารถใช้ทางเลือกที่เรียกว่า finally clause โดยมีรูปแบบดังนี้:

```
try
{
  // code
  //  more code
}
catch( ExceptionType e)
{
   //handler for this exception
}
finally
{ .. }
```

- **finally block** จะประมวลผลโดยไม่สนใจว่า exceptions ถูก thrown หรือไม่ภายใน **try** block

The try-catch-finally statement


```
public class FinallyDemo {
   public void aMethod(){
    try{
      int x = 5/0;
    } catch(ArithmeticException e){
      System.out.println("In catch,terminating aMethod");
      return;
    }
    finally{
      System.out.println(" Executing finally block");
    }
      System.out.println( "Out of catch block");
  }//end aMethod
public static void main(String[] args){
        new FinallyDemo().aMethod();
  }
}//end class
```

In catch, terminating aMethod
Executing finally block

Execute return statement in catch block

- โค้ดจากโปรแกรมตัวอย่างจะเกิด **ArithmeticException** เนื่องจากมีความพยายามที่จะหารค่า integer ด้วย 0 จากนั้นกลไกควบคุมจะถูกส่งผ่านไปยัง **catch** block ที่กำหนดไว้และแสดงข้อความและประมวลผลคำสั่ง **return** ทันที
- โดยปกติแล้วการประมวลผลคำสั่ง **return** จะมีผลทำให้สิ้นสุดการทำงานของเมธอดทันที
- แต่ในกรณีนี้ก่อนที่จะสิ้นสุดการทำงานของเมธอด โค้ดภายใน **finally** จะถูกประมวลผลก่อนที่ control จะถูกส่งกลับไปยังเมธอด **main** ต่อไป
- output :

```
In catch, terminating aMethod Executing finally
block.
```

Final Block - Always Called


```
class  FinalBlockExceptionNotCalled  {
        public static void main( String args[] )  {
         int  students[] = new int[5];
          try  {
            System.out.println( "Start Try"  );
            students[2]  =  1;
            System.out.println("After array statement");
          }
          catch  (ArrayIndexOutOfBoundsException e)   {
             System.err.println( "OutOfBounds: "  + e.getMessage());
          }
          finally   {
              System.out.println(  "In Final Block"  );
           }
          System.out.println(  " After try "  );
       }
    }
```

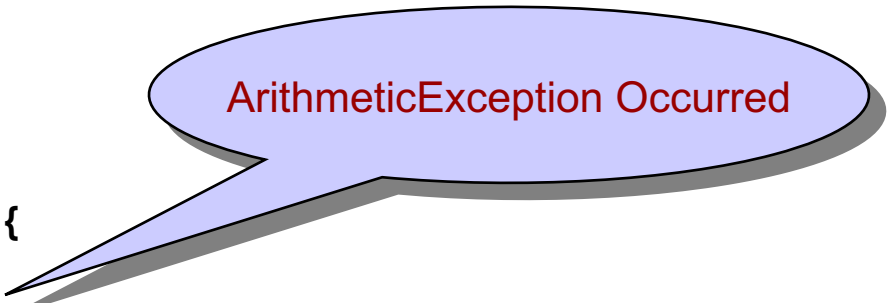
//Start Try //After array statement //In Final Block //After try

Exception Propagation

- ในกรณีที่ไม่เหมาะสมที่จะจัดการ exception ในตำแหน่งที่เกิดขึ้น การจัดการความผิดปกติดังกล่าวอาจจัดการได้ในระดับที่สูงขึ้นถัดไปได้
- โดย Exceptions จะถูกส่งผ่านกลไกควบคุมกลับไปยังเมธอดที่เรียกใช้ตามลำดับ จนกว่าจะถูกตรวจสอบและจัดการ exception นั้น ๆ ได้อย่างเหมาะสมในระดับที่สูง ๆ ขึ้นจนกว่าจะถึงระดับบนสุด
- ในกรณีที่ exception ไม่ถูกตรวจสอบและจัดการในตำแหน่งที่เกิดขึ้นได้ กลไกควบคุมจะถูกส่งคืนกลับไปยังเมธอดที่เรียกใช้เมธอดที่เกิดความผิดปกติขึ้นทันที
- กระบวนการนี้จะถูกเรียกว่า Exception Propagation

Exception Propagation

```
class MathDivider {
        private int no1, no2;
        public MathDivider(int no1, int no2) {
                this.no1 = no1;
                this.no2 = no2;
        }
       public int checkZero() throws Exception {
                if (no2 == 0)
                   throw new Exception("Divided by Zero");
                else
                   return  no2;
         }
        public int divider() throws Exception {  return  (no1/this.checkZero());
        }
}
class RunDivider {
      public static void main(String[] args)  throws Exception {
        MathDivider md = new MathDivider(5, 0);
        System.out.println(" "+ md.divider());
      }
}
```




```
Exception in thread "main" java.lang.Exception: Divided by Zero
at MathDivider.checkZero(MathDivider.java:12)
at MathDivider.divider(MathDivider.java:17)
at RunDivider.main(MathDivider.java:24)
```

Divided by Zero : Exceptions


```
class Arithmatic
{
     public static void main(String [] args) {
     int num[] = {4, 6, 7, 1};
     int divided[] = {2, 3, 0, 8};

     for (int i = 0; i < 4; ++i) {
       try {
           System.out.println( num[i] + " / "+divided[i]  + " = " +
                                                          num[i]/divided[i]);
       }
      catch (ArithmeticException ex) {
         System.out.println( ex.getMessage()); }
     }
   }
}
```

```
4 / 2 = 2
6 / 3 = 2
/ by zero
1 / 8 = 0
```

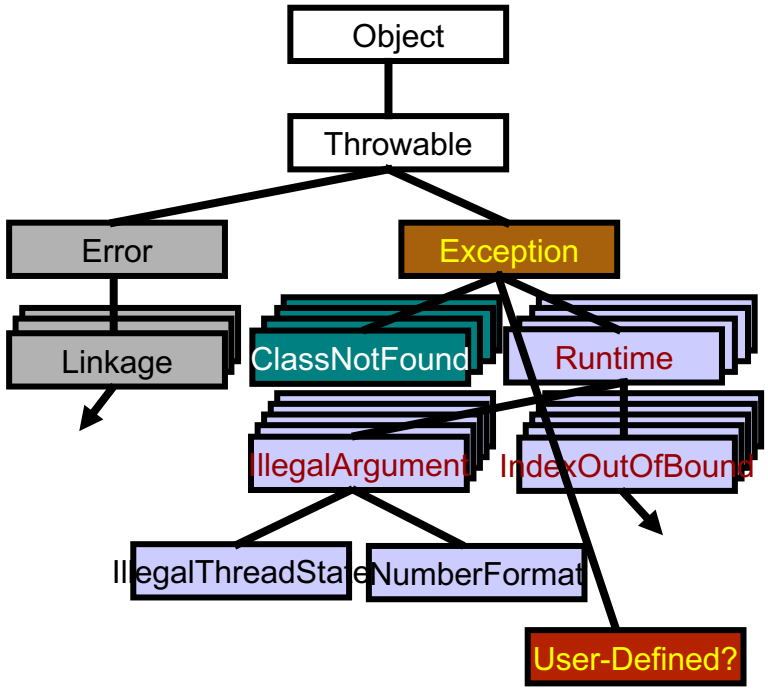
Exception Handling Example: Divide by Zero

- ตัวอย่างโปรแกรม
  - ผู้ใช้กำหนดค่าตัวเลขแบบ integers จำนวนสองค่าสำหรับการ divided
  - ต้องการ catch errors จากการ divide โดยค่า 0
  - Exceptions
    - Objects จะถูกสืบทอดมาจากคลาส **Exception**
  - หากตรวจสอบภายในคลาส **Exception** ใน **java.lang**
    - ไม่มีโค้ดที่เหมาะสมสำหรับการจัดการ divide by zero
    - ส่วนที่ใกล้เคียงที่สุดคือ **ArithmeticException**
    - สร้างคลาสที่สืบทอดมาจากคลาส exception ขึ้นใหม่

User-defined Throwable Classes

- ผู้ใช้สามารถสร้างคลาสสืบทอดจาก Exception เป็นของตัวเองได้
- โดยจาวาจะยอมให้กำหนดเงื่อนไขที่ตรงกับ Exceptions ที่ผู้ใช้ต้องการตรวจสอบ

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



Create your own exceptions

- คลาส Exception ส่วนใหญ่จะประกอบไปด้วย 2 constructors ดังนี้
  - แบบแรกจะไม่มี arguments (default)
  - แบบที่สองจะรับค่า exception message
- ทั้งสองเมธอดเรียกใช้ constructor ไปยัง SuperClass

```
public class DivideByZeroException  extends ArithmeticException
{
      public DivideByZeroException()
      public DivideByZeroException( String message )

}
```

Exception Handling Example: Divide by Zero


```
public class DivideByZeroException extends
    ArithmeticException {

    public DivideByZeroException()  {
        super( "Attempted to divide by zero" );
    }
    public DivideByZeroException( String message )   {
        super( message );
    }
}
```

- ขั้นตอนถัดไปจะเป็นการ throw Exception ในส่วนของ **try** block
- ส่วนโค้ดสำหรับจัดการ Error จะอยู่ภายใน **catch** block
- หากไม่มีการ thrown ส่วนของ **catch** blocks จะถูกข้ามไป

Your own exceptions with throw stmt


```
class DivideByZeroException extends ArithmeticException {
    public DivideByZeroException()  {
        super("Attempted to divide by zero");
    }
}
class ExceptionTest {
    public  static  void main(String [] args) {
    int num[] = {4, 6, 7, 1};
    int divided[] = {2, 3, 0, 8};
    for  (int i = 0; i < 4; ++i) {
        try  {
            if ((divided[i]) == 0)
                throw  new DivideByZeroException();
             System.out.println( num[i] + "  /  "+divided[i] +
                                    "  =  " +  num[i]/divided[i]);
        }
     catch (DivideByZeroException ex) {
        System.out.println(ex.getMessage());        }
     }   }
}
```

```
4 / 2 = 2
6 / 3 = 2
Attempted to divide by zero
1 / 8 = 0
```

Class Account


```
public class Account    {
    private double balance;
    public void deposit (double amount)    {
            balance = balance + amount;
    }
    public void withdraw (double amount)   {
            balance = balance - amount;
    }
    public double getBalance()   {
            return balance;
    }
}
```

- deposit() อาจเกิดปัญหาในกรณีที่ amount เป็นค่าติดลบ
- withdraw() อาจเกิดปัญหาในกรณีที่ amount > balance

Create your own exceptions

- สร้างคลาสที่สืบทอดมาจากคลาส Exception
- ประกอบไปด้วย 2 constructors
- ส่วน s จะเป็นรายละเอียดของ error message

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

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


```
class BalanceException {
      private int balance;
      public BalanceException(int balance) {
            this.balance = balance;
       }
       public int getBalance() {
             return balance;
       }
      public void withdraw(int amt) throws NegativeInputException,
                                                                  NotEnoughMoneyException {
             if(amt < 0) {
                throw new NegativeInputException("Withdraw amount cannot be negative");
             } else if (amt > balance) {
                 throw new NotEnoughMoneyException("Amount less than minimum Balance");
             } else {
                  balance-=amt;
             }
        }
        public void deposit(int amt) throws NegativeInputException {
             if(amt < 0) {
                throw new NegativeInputException("Deposit amount cannot be negative");
             }
            else {
                balance+=amt;
           }
    }
}
```

Test.java


```
public class Test {
      public static void main(String[] args) {
       Balance b = new Balance(100);
       try {
              b.withdraw(200);
       } catch (NegativeInputException ex) {
              ex.printStackTrace();
       } catch (NotEnoughMoneyException ex) {
              ex.printStackTrace();
       }
       try {
             b.deposit(-100);
       } catch (NegativeInputException ex) {
              ex.printStackTrace();
       }

   }
}
```

Test.java

Output

```
    NotEnoughMoneyException
  at BalanceException.withdraw(BalanceException.java:15)
  at Run.main(BalanceException.java:53)
NegativeInputException: Deposit amount cannot be negative
  at BalanceException.deposit(BalanceException.java:23)
  at Run.main(BalanceException.java:59)
```