Error Handling With Exception

Vittayasak Rujivorakul

Lecture 8

Error Handling With Exception

Vittayasak Rujivorakul

Lecture 8

Basic Exceptions

- **Exception** = ข้อผิดพลาดซึ่งละเมิด semantic rules ของ java programs (e.g., อ่านเกินขนาดของ array, อ่านเกินกว่า end of file)
- **Exception Handler** = การจัดการกับ error (สามารถแยกตัวการจัดการกับ code ปกติไว้คนละที่กันได้)
- **Exceptional Condition** = ปัญหาที่ทำให้การทำงานสะดุด
- เมื่อมีความผิดพลาด โปรแกรมสามารถทำการ throw exception :
  - สร้าง exception object ใน heap โดยใช้ new
  - เปลี่ยนเส้นทางการทำงานไปทำ exception handler แทน

Exceptions

- การ throw exception ใช้คำสั่ง throw
- ใน Exception มาตรฐาน จะมี constructors 2 แบบ
  - default constructor

```
if (t == null)
    throw new NullPointerException ( ) ;
```

  - constructor ที่มี String argument ซึ่งเราสามารถดึง String นี้ออกมาได้

```
if (t == null
    throw new NullPointerException ("t is null") ;
```

- สามารถทำการ throw Throwable object ตัวใดที่ต้องการก็ได้

การจับ Exception

- การจัดการกับ exceptions ในแบบ try -catch
- **try-catch Syntax :**

```
try {
    <try block statements>
} catch (<exception type> <identifier>) {
    <catch block statements>
}
```

Example :

```
try {
    . . .
} catch (IOException e) {
    success = false ;
    Syatem.out.println (e) ;
}
```

Try-catch example (Cont.)

- แยกรายละเอียดแต่ละ exception ; exception ที่เกิดขึ้นเท่านั้นทำงาน

```
try   {
      . . /
}
catch  (IOException e)   {
      success  =  false ;
      System.out.println  (“General  I/O  exception thrown”) ;
}
catch  (FileNotFoundException e)  {
      . . .
}
catch  (EOFException  e)   {
      . . .
}
```


Try-catch example

- ใช้ Finally block จะถูก executed เสมอไม่ว่า exception จะเกิดหรือไม่

```
try   [
      . . .
}  catch  (IOException e)   {
      . . .
finally   {
      / /  statements places here
      / /  will always get executed
}
```


Catching an Exception

- ในทางทฤษฎี การจัดการจับ exception ทำได้ 2 ลักษณะ
  - Termination : error นั้นร้ายแรง ไม่สมควรที่จะทำงานต่อไป
  - Resumption : error นั้นแก้ไขได้ สมควรลองทำซ้ำ

The exception specification

- เพื่อให้ง่ายในการค้นหาว่า method ต่าง ๆ มีการ throw exception อะไรบ้าง สามารถกำหนด exception specification ร่วมกับการประกาศ method

```
void f ( )  throws tooBig,  tooSmall,  divZero   {  . . .  }
public void write ( ) throws IOException   [
      . . .  =  new FileOutputStream (outFile) ;
      . . .
      outStream.write (beytArray) ;
      . . .
      outStream.close ( ) ;
}
```

Stmts. เหล่านี้มีการ throws IOException เพื่อให้ผู้เรียก method นี้เลือกจัดการกับ IOException ตามเหมาะสม ดังนั้นเราจึงเพิ่ม throws IOException ไว้กับ method ที่เรากำหนดขึ้น

การ Catching exception ใด ๆ

- สามารถที่จะ catch exception ใด ๆ ได้โดยการจับ base exception

  Exception

  catch (Exception e) [

  System.out.println (“caught an exception”) ;

  }
- **ซึ่งไม่เฉพาะว่าเป็นอะไร สามารถจะใช้ method ใน throwable class เพื่อช่วยเพิ่มรายละเอียด**

  String getMessage ( )

  String toString ( )

  void printStackTrace ( )

  void print StackTrace (PrintStream)

Try-catch examples

- ใช้ printStackTrace ( ) เพื่อพิมพ์ exception และ trace กลับ

  try {

  . . .

  Throw new Exception (“Here’ s my exception”) ;

  } catch (Exception e) {

  success = false ;

  e.**getMessage ( ) ;** / / show message details

  e.**toString ( ) ;** / / show short description & message details

  / / prints this Throwable and its backtrace to the specified print stream.

  e.**printStackTrace ( ) ;**

  }

Rethrowing an exception

- **สามารถทำการ rethrow exception ได้**

  try {

  . . .

  } catch (Exception) {

  System.out.println (“Occurred an exception” ;

  throw e ;
- **ทำการ install new stack trace information ได้โดย**

  catch (Exception e) {

  System.out.println (“Occurred an exception” ;

  throw e. fillInStackTrace ( ) ; / / need a Throwable object

  }

Standard Java Exception

- Throwable class เป็น class ที่อธิบายถึง exception ต่าง ๆ ที่สามารถถูก thrown ได้
- Throwable object มี 2 ชนิด
  - Error : compile-time & system errors ซึ่งเราไม่ต้องสนใจที่จะ catch
  - Exception : เป็น basic type (java.lang.Exception) ซึ่งจะถูก thrown โดย class methods ต่าง ๆ (standard Java Library, user methods & runtime accidents) ในจาวา
- Exception อื่น ๆ ขยายจาก base ซึ่งจะถูกสร้างอยู่ทั่วไปใน libraries

Runtime Exception

- Java มีมาตรฐานในการตรวจสอบ error ในช่วง runtime และทำโดยอัตโนมัติ (เช่น NullPointerException จะถูก thrown หากพยายามอ้างถึง object โดยตัวชี้ null)
- exception เหล่านี้ไม่ถูกบังคับให้ catch และจะพบได้ในช่วง runtime โดยทั่วไปจะเรียกว่า printStackTrace ( ) เมื่อโปรแกรมสิ้นสุดการทำงาน

```
public class NeverCaught {
   static void f( ) {
      throw new RuntimeException("From f( )");
   }
   static void g( ) { f( ); }
   public static void main(String [ ] args) {
      g( );
   }
}
```


การสร้าง exception ของเราเอง

- การสร้าง exception ทำโดยสืบทอดจาก exception ที่มีอยู่เดิม

```
class MyException extens Exception  {
        public MyException ( )  { }
        public MyException (String msg)  {  super (msg) ;  }
   }
   public class Inheriting  {
        public static void f ( ) throws MyException  {
                System.out.println ("Throw Myexception from f ( ) ") ;
                throw new Myexception ( ) ;
        }
   }
```


Exception restrictions

- Overridden method ใน derived class ไม่สามารถ throws exception มากกว่า method ของ based class ได้
- Overridden method ใน derived class อาจไม่ throws exception ถึงแม้จะมีการ throw ใน method ของ base class
- Interface methods ไม่สามารถเปลี่ยนลักษณะของ exception ใน based class method ได้
- Constructor ของ base และ derived class เลือก throws exception ตามชอบ
- ทำการ catch exception เฉพาะที่ระบุใน subclass นั้น ๆ แต่หากมีการ upcast และใน base class มีการ throws มากกว่า ต้องทำการ catch exception ในกรณีนั้น (compile-time checking)

Exception matching

- เมื่อมีการ throw exception การจับคู่หาตัวจัดการ exception จะเป็นไปตามลำดับที่เขียน เมื่อพบทำงานตามนั้นแล้วไม่หาต่อ

```
class Annoyance extends Exception  { }
   class Sneeze extends Annoyance  { }
   public class Human  {
        public static void main (String [ ] args)  {
                try  {
                        throw new Sneeze ( ) ;
                }  catch  (Sneeze  s)  {  / / match this first
                        System.out.println ("Caught Sneeze") ;
                }  catch (Annoyance a)  { / / try after previous failed
                        System.out.println ("Caught Annoyance" ) ;
                }
        }
   }
```

การใช้ Exception เพื่อ (Exception Guidelines)

- แก้ปัญหาและทำ method ซ้ำอีกครั้ง
- ทำงานต่อโดยไม่พยายามที่จะทำ method ที่ error นั้นใหม่
- เบี่ยงเบนการคำนวณเพื่อให้ได้ผลลัพธ์แบบใหม่แทน method ปกติ ทำเท่าที่ทำได้ หากผิดพลาดให้ส่งความผิดพลาดนั้นไปยังตัวเรียก ทำเท่าที่ทำได้ หากผิดพลาดให้ส่งความผิดพลาดใหม่ไปยังตัวเรียก
- หยุดการทำงานของโปรแกรม
- เพื่อความปลอดภัยให้ library และ program (debug & robustness)