IS311 Programming Concepts

JAVA

Exception Handling

การจัดการสิ่งผิดปรกติ

Introduction

- Users have high expectations for the code we produce.
- Users will use our programs in unexpected ways.
- Due to design errors or coding errors, our programs may fail in unexpected ways during execution

Introduction

- It is our responsibility to produce quality code that does not fail unexpectedly.
- Consequently, we must design error handling into our programs.

Errors and Error Handling

- An Error is any unexpected result obtained from a program during execution.
- Unhandled errors may manifest themselves as incorrect results or behavior, or as abnormal program termination.
- Errors should be handled by the programmer, to prevent them from reaching the user.

Errors and Error Handling

- Some typical causes of errors:
  - Memory errors (i.e. memory incorrectly allocated, memory leaks, "null pointer")
  - File system errors (i.e. disk is full, disk has been removed)
  - Network errors (i.e. network is down, URL does not exist)
  - Calculation errors (i.e. divide by 0)

Errors and Error Handling

- More typical causes of errors:
  - Array errors (i.e. accessing element –1)
  - Conversion errors (i.e. convert 'q' to a number)
  - Can you think of some others?

Errors and Error Handling

- Traditional Error Handling
  - 1. Every method returns a value (flag) indicating either success, failure, or some error condition. The calling method checks the return flag and takes appropriate action.
  - Downside: programmer must remember to always check the return value and take appropriate action. This requires much code (methods are harder to read) and something may get overlooked.

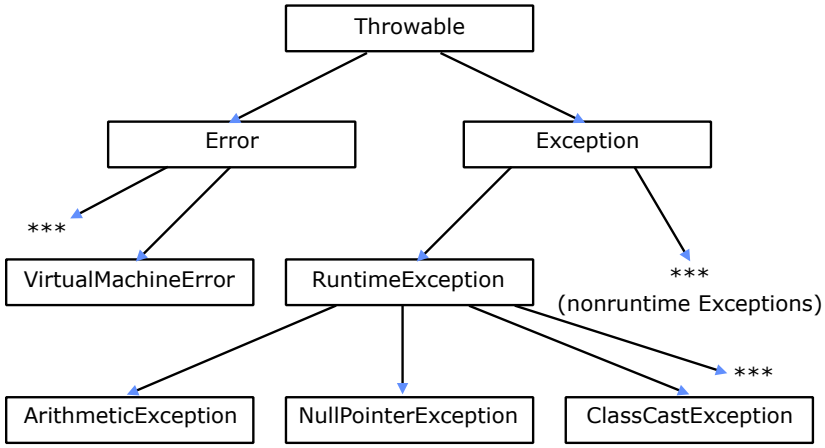
Errors and Error Handling

- **Exceptions** – a better error handling
  - Exceptions are a mechanism that provides the best of both worlds.
  - Exceptions act similar to method return flags in that any method may raise and exception should it encounter an error.
  - Exceptions act like global error methods in that the exception mechanism is built into Java; exceptions are handled at many levels in a program, locally and/or globally.

Exceptions and JVM

- When an exception takes place, the Java Virtual Machine (JVM) creates an exception object to identify the type of exemption that occurred
- The Throwable class is the super class of all error and exception types generated by the JVM or Java programs
- Three Throwable subclass categories are possible: Error, Runtime and Nonruntime Exceptions

Exceptions Hierarchy



*** หมายถึงยังมีอีก แต่ไม่กล่าวถึง
คลาสที่ระบายสีแดงหมายถึงมีข้อผิดพลาดร้ายแรง โปรแกรมทำอะไรต่อไม่ได้

2 type of Exceptions

- *Checked:* are the exceptions that are checked at compile time. Checked exceptions must be handled or declared otherwise it causes compile time error.
- *Unchecked*: are the exceptions that are not checked at compiled time. No need to handled or declared unchecked exceptions, they wont't cause any compile time error.

คลาสที่อยู่ภายใต้คลาส RuntimeException เป็นคลาสแบบ unchecked ทั้งสิ้น เราจะเรียนการจัดการ exception สำหรับคลาสในกลุ่มนี้

The `try catch finally` statement

```
try {
  //  คำสั่งต่างๆ ที่ต้องการทำและอาจก่อให้เกิดสิ่งผิดปรกติ
} catch (ExceptionType1  e)  {
  //คำสั่งที่ใช้จัดการเมื่อเกิดสิ่งผิดปรกติชนิด ExceptionType1
} catch (ExceptionType2  e)  {
  //คำสั่งที่ใช้จัดการเมื่อเกิดสิ่งผิดปรกติชนิด  ExceptionType2
  throw e;  //  โยนสิ่งผิดปรกติไปให้อ๊อบเจกต์ e จัดการต่ออีกทอด
} finally {
  //  คำสั่งที่ต้องทำเสมอ ไม่ว่าจะเกิดสิ่งผิดปรกติชนิดใดขึ้นหรือไม่ก็ตาม
}
```

Keywords for Java Exceptions

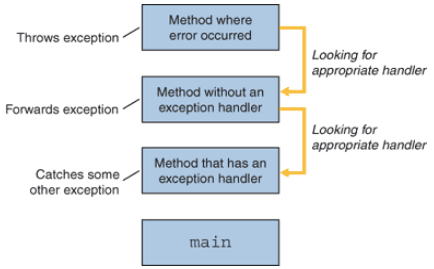
- **throws (ใช้ตอนประกาศเมท็อด)**
  Describes the exceptions which can be raised by a method.
- **throw**
  Raises an exception to the first available handler in the call stack, unwinding the stack along the way.
- **try**
  Marks the start of a block associated with a set of exception handlers.
- **catch**
  If the block enclosed by the try generates an exception of this type, control moves here.
- **finally**
  Always called when the try block concludes, and after any necessary catch handler is complete.

Exception-Handling Mechanism

1. Mechanism for creating special exception classes (whose instances are called *exception objects*)
2. The statement `throw e` is used to signal the occurrence of an exception and return control to the calling method and `e` refers to an exception object
3. The statement `try/catch` allows the calling method to "catch" the "thrown" exception object and take appropriate actions

Control Flow and Exceptions

- When exception is thrown control returns through the methods called in reverse calling order until a try statement is found with a catch block for the exception
- It is possible for a catch statement to defer handling of an exception by including a throw statement of its own

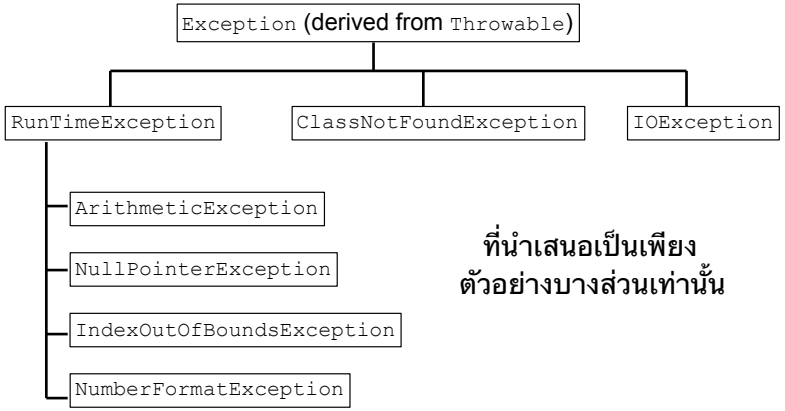


Error & Exception

- Error Class
  - Critical error which is not acceptable in normal application program
- Exception Class
  - Possible exception in normal application program execution
  - Possible to handle by programmer

Exception Class Hierarchy

Java has a predefined set of exceptions and errors that can occur during execution.



ที่นำเสนอเป็นเพียงตัวอย่างบางส่วนเท่านั้น

ตัวอย่าง System-Defined Exception

- `ArithmeticException` :
  - When an exceptional arithmetic condition has occurred
- `ArrayStoreException` :
  - When assign object of incorrect type to element of array
- `FileNotFoundException` :
  - เกิดขึ้นเมื่อมีการเปิดแฟ้มข้อมูล แต่หาแฟ้มที่ต้องการไม่พบ
- `IndexOutOfBoundsException` :
  - When beyond the bound of index in the object which use index, such as array, string, and vector
- `IllegalMonitorStateException` :
  - When the thread which is not owner of monitor involves wait or notify method

Programmer-Defined Exception

Exceptions raised by programmer

- Check by compiler whether the exception handler for exception occurred exists or not
  - If there is no handler, it is error
- Sub class of Exception class

Exception Occurrence

- Raised implicitly by system
- Raised explicitly by programmer
  - **`throw`** Statement

```
throw   ThrowableObject;
```

`Throwable` class or its sub class

ตัวอย่าง System-Defined Exception

- `NegativeArraySizeException` :
  - When using a negative size of array
- `NumberFormatException` :
  - เกิดขณะที่ทำการแปลงสตริงไปเป็นข้อมูลชนิดตัวเลข แต่สตริงอยู่ในรูปแบบที่ไม่เหมาะสม
- `NullPointerException` :
  - When refer to object as a null pointer
- `SecurityException` :
  - When violate security. Caused by security manager

Coding Exceptions

- Try-Catch Mechanism
  - Wherever your code may trigger an exception, the normal code logic is placed inside a block of code starting with the "try" keyword:
  - After the try block, the code to handle the exception should it arise is placed in a block of code starting with the "catch" keyword.

Coding Exceptions

- Try-Catch Mechanism
  - You may also write an optional "finally" block. This block contains code that is ALWAYS executed, either after the "try" block code, or after the "catch" block code.
  - Finally blocks can be used for operations that must happen no matter what (i.e. cleanup operations such as closing a file)

Exception Example

- The body of a method may call other methods as well as doing its own calculations
- Here the body of `m` will execute unless an out-of bounds exception occurs

```
void m (){
  try {
    … body of m …
  }
  catch (ArrayIndexOutOfBoundsException ae) {
    … code to recover from error …
  }
}
```

ArithmeticException example

```
public class Zero {
    public static void main(String[] args) {
        int numerator = 10;
        int denominator = 0;
        System.out.println(numerator/denominator);
        System.out.println("We never get to this statement. ");
    }
}
```

```
Exception in thread "main"
java.lang.ArithmeticException: / by zero
            at Zero.main(Zero.java:5)
```


ArrayIndexOutOfBoundsException Example

```
class Main {
    static void doSomeThing() {
        try {
            int array[] = {1,2,3,4,5};
            int a=5, b=0;
            b = array[a];
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Error: index out of range");
        }
    }
    public static void main(String []args) {
        try {
            int a=5, b=0, c=0;
            doSomeThing();
            c = a/b;
        } catch ( ArithmeticException e) {
            System.out.println("Error: divide by zero");
        }
    }
}
```


การดักจับสิ่งผิดปรกติหลายชนิดในบล็อก `try` เดียวกัน

```
class Main {
    public static void main(String []args) {
        int a=5, b=0, c=0;
        int array[] = {1,2,3,4,5};
        try {
            b = array[a];
            c = a/b;
        } catch ( ArithmeticException e) {
            System.out.println("Error: divide by zero");
        } catch ( ArrayIndexOutOfBoundsException e) {
            System.out.println("Error: index out of range");
        }
        System.out.println(c);
    }
}
```


Finally Clause

- When exception is thrown control is transferred to method containing the catch block to handle the exception
- Control does not return to procedure in which the exception was thrown unless it contains a finally clause
- The finally clause can be used to clean up the programming environment after the exceptions has been handled

Finally Block

```
class Main {
    public static void main(String []args) {
        int a=5, b=0, c=0;
        int array[] = {1,2,3,4,5};
        try {
            b = array[a];
            c = a/b;
        } catch ( ArithmeticException e) {
            System.out.println("Error: divide by zero");
        } catch ( ArrayIndexOutOfBoundsException e) {
            System.out.println("Error: index out of range");
        } finally {
            System.out.println("Message from finally block");
        }
        System.out.println("Message from the line after " +
                           "the whole try block");
    }
}
```

Throwing Exceptions

- You can throw exceptions from your own methods
- To throw an exception, create an instance of the exception class and "throw" it.
- If you throw checked exceptions, you must indicate which exceptions your method throws by using the throws keyword

```
public void withdraw(float anAmount) throws
 InsufficientFundsException
{
  if (anAmount<0.0)
    throw new IllegalArgumentException(
     "Cannot withdraw negative amt");
  if (anAmount>balance)
    throw new InsuffientFundsException("Not enough cash");
  balance = balance - anAmount;
}
```


Resource

- ดาวน์โหลดชีทและตัวอย่างโปรแกรม เรื่อง **การจัดการสิ่งผิดปรกติ** http://www.bus.tu.ac.th/usr/wanhai/is311 หน้าดาวน์โหลด