J01-13

程式執行異常處理

曾瑞君 (Jim_Tzeng)

學習目標

1. Java執行時的異常
2. Exception的傳播
3. Catching or Throwing?
4. 處理Exception的好習慣

1/4

Java執行時的異常

天有不測風雲...

```
public class Test {
    private static void test() {
        int[] intArray = new int[5];
        //intArray[4] = 27;
        intArray[5] = 27;
    }
    public static void main(String[] args) {
        test();
    }
}
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
        at concept.c13.Test.test(Test.java:7)
        at concept.c13.Test.main(Test.java:10)
```

如何處理Exception?

- 正常流程：
  1. caller method 呼叫 worker method
  2. worker method 做事
  3. worker method 完事並回傳結果至 caller method

- Exception發生時：
  - Java程式發生錯誤時，JVM會拋出例外物件(Exception Object)，並說明例外發生的地方，以及例外種類。
  - 藉由這些資訊進行修正，避免再次發生。

Exception 分類

分類	Checked Exception	Unchecked Exception	
說明	已然預知風險， 必須事先預防， 避免程式中斷。	無法預知風險， 無法事先預防	
代表類別	所有例外類別都是。除： ① RuntimeException ② Error 類別和其子類別	RuntimeException 類別和其子類別	Error 類別和其子類別
		歸類程式**內部**原因： 如資料輸入異常	歸類程式**外部**原因： 如硬體，網路等
處理方式	① 方法內部自己處理。 ② 方法內部不處理但提醒呼叫者要處理。	不需要事先處理	

https://docs.oracle.com/javase/tutorial/essential/exceptions/catchOrDeclare.html

- Checked exception: These are exceptional conditions that a well-written application should anticipate and recover from.
- All exceptions are checked exceptions, except for those indicated by Error, RuntimeException, and their subclasses.

- Error: These are exceptional conditions that are **external** to the application, and that the application usually cannot anticipate or recover from.
- Errors are those exceptions indicated by Error and its subclasses

- Runtime exception: These are exceptional conditions that are **internal** to the application, and that the application usually cannot anticipate or recover from.
- Runtime exceptions are those indicated by RuntimeException and its subclasses.

OutOfMemoryError

```
public class Test {
private static void test() {
	List<String> list = new ArrayList<String>();
	while (true) {
		String s = "OutOfMemoryError Test";
		list.add(s);
		if (list.size() % 1000000 == 0) {
			System.out.println(list.size() / 100000 + " million String created!");
		}
	}
}
public static void main(String[] args) {
	System.out.println("============ Test Start!! ============");
	test();
	System.out.println("============ Test End!! ============");
	}
}
```

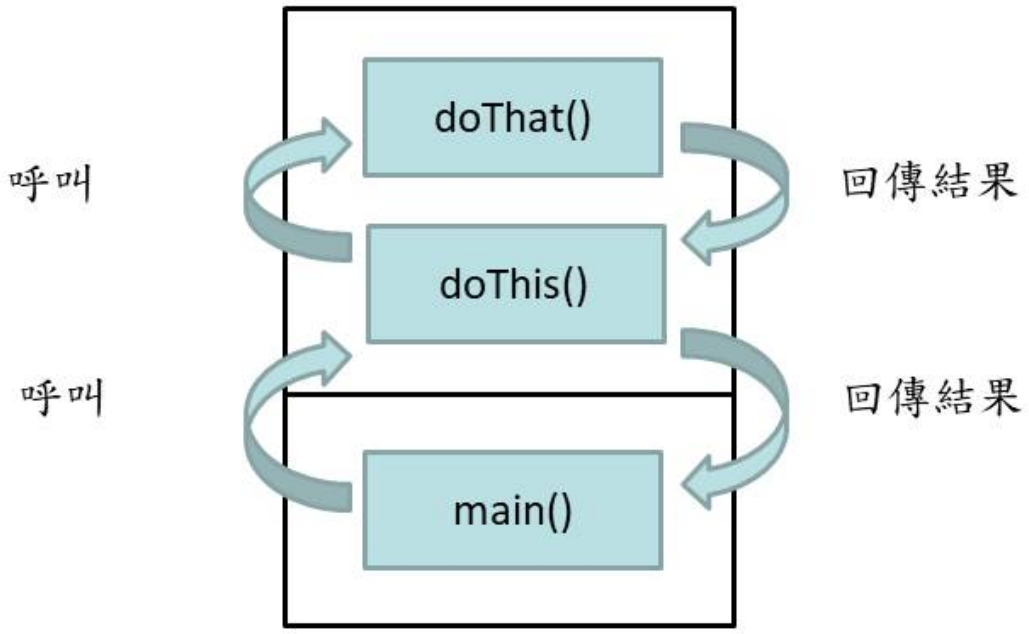
OutOfMemoryError

```
--> 1560 million String created!
--> 1570 million String created!
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
        at java.util.Arrays.copyOf(Arrays.java:2245)
        at java.util.Arrays.copyOf(Arrays.java:2219)
        at java.util.ArrayList.grow(ArrayList.java:242)
        at java.util.ArrayList.ensureExplicitCapacity(ArrayList.java:216)
        at java.util.ArrayList.ensureCapacityInternal(ArrayList.java:208)
        at java.util.ArrayList.add(ArrayList.java:440)
        at concept.c13.Test.test(Test.java:10)
        at concept.c13.Test.main(Test.java:19)
```

2/4

Exception 的傳播 & Catching? Throwing?

Method Stack



```java
public class Test1 {
    public static void doThat() {
        System.out.println("Start doThat()");
        System.out.println("End doThat()");
    }
    public static void doThis() {
        System.out.println("Start doThis()");
        doThat();
        System.out.println("End doThis()");
    }
    public static void main(String[] args) {
        System.out.println("============ Start main() ============");
        doThis();
        System.out.println("============ End main() ============");
    }
}
```

Result:

```
============ Start main() ============
Start doThis()
Start doThat()
End doThat()
End doThis()
============ End main() ============
```

- 若程式進行中遇到【Runtime Exception】，將會出錯而意外中止。

```java
public class Test2 {
    public static void uncheckedException() {
        int[] arr = new int[9];
        arr[9] = 27;
    }
    public static void doThat() {
        System.out.println("Start doThat()");
        uncheckedException();
        System.out.println("End doThat()");
    }
    public static void doThis() {
        System.out.println("Start doThis()");
        doThat();
        System.out.println("End doThis()");
    }
    public static void main(String[] args) {
        System.out.println("============ Start main() ============");
        doThis();
        System.out.println("============ End main() ============");
    }
}
```

Result:

```
============ Start main() ============
Start doThis()
Start doThat()
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
        at concept.c13.Test2.uncheckedException(Test2.java:6)
        at concept.c13.Test2.doThat(Test2.java:10)
        at concept.c13.Test2.doThis(Test2.java:15)
        at concept.c13.Test2.main(Test2.java:20)
```

- 雖然屬於【unchecked Exception】，但還是可以做預防處理；將出錯的程式導入try catch block 區塊，如此程式可以正常結束。

```
try {
        // something might error
} catch (ExceptionType name) {
        // something to deal with it
}
```

```java
public class Test3 {
    public static void uncheckedException() {
        int[] arr = new int[9];
        arr[9] = 27;
    }
    public static void doThat() {
        System.out.println("Start doThat()");
        try {
            uncheckedException();
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println("End doThat()");
    }
    public static void doThis() {
        System.out.println("Start doThis()");
        doThat();
        System.out.println("End doThis()");
    }
    public static void main(String[] args) {
        System.out.println("============ Start main() ============");
        doThis();
        System.out.println("============ End main() ============");
    }
}
```

Result:

```
============ Start main() ============
Start doThis()
Start doThat()
java.lang.ArrayIndexOutOfBoundsException: 5
        at concept.c13.Test3.uncheckedException(Test3.java:6)
        at concept.c13.Test3.doThat(Test3.java:11)
        at concept.c13.Test3.doThis(Test3.java:19)
        at concept.c13.Test3.main(Test3.java:24)
End doThat()
End doThis()
============ End main() ============
```

Edited by Ruei-Jiun Tzeng

- 若是【checked Exception】，表示一定要先做預防處理，處理方式可以是：
  - 將Exception回給caller method。語法為在method的宣告尾端加上：

```
[modifiers] return_type method_identifier ([arguments]) [throws ExceptionTypes] {
        method_code_block
}
```

但如此caller method也會面臨該如何處理的抉擇。如Test4。

  - 將出錯的程式導入 try catch block 區塊，讓程式可以正常結束，如Test5。

```java
public class Test4 {
    public static void checkedException() throws Exception {
        if (Math.random() > 0.01) {
            throw new Exception();
        }
    }
    public static void doThat() throws Exception {
        System.out.println("Start doThat()");
        checkedException();
        System.out.println("End doThat()");
    }
    public static void doThis() throws Exception {
        System.out.println("Start doThis()");
        doThat();
        System.out.println("End doThis()");
    }
    public static void main(String[] args) throws Exception {
        System.out.println("============ Start main() ============");
        doThis();
        System.out.println("============ End main() ============");
    }
}
```

Result:

```
============ Start main() ============
Start doThis()
Start doThat()
Exception in thread "main" java.lang.Exception
        at concept.c13.Test4.checkedException(Test4.java:6)
        at concept.c13.Test4.doThat(Test4.java:11)
        at concept.c13.Test4.doThis(Test4.java:16)
        at concept.c13.Test4.main(Test4.java:21)
```

```java
public class Test5 {
    public static void checkedException() throws Exception {
        if (Math.random() > 0.01) {
            throw new Exception();
        }
    }
    public static void doThat() {
        System.out.println("Start doThat()");
        try {
            checkedException();
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println("End doThat()");
    }
    public static void doThis() {
        System.out.println("Start doThis()");
        doThat();
        System.out.println("End doThis()");
    }
    public static void main(String[] args) {
        System.out.println("============ Start main() ============");
        doThis();
        System.out.println("============ End main() ============");
    }
}
```

Result:

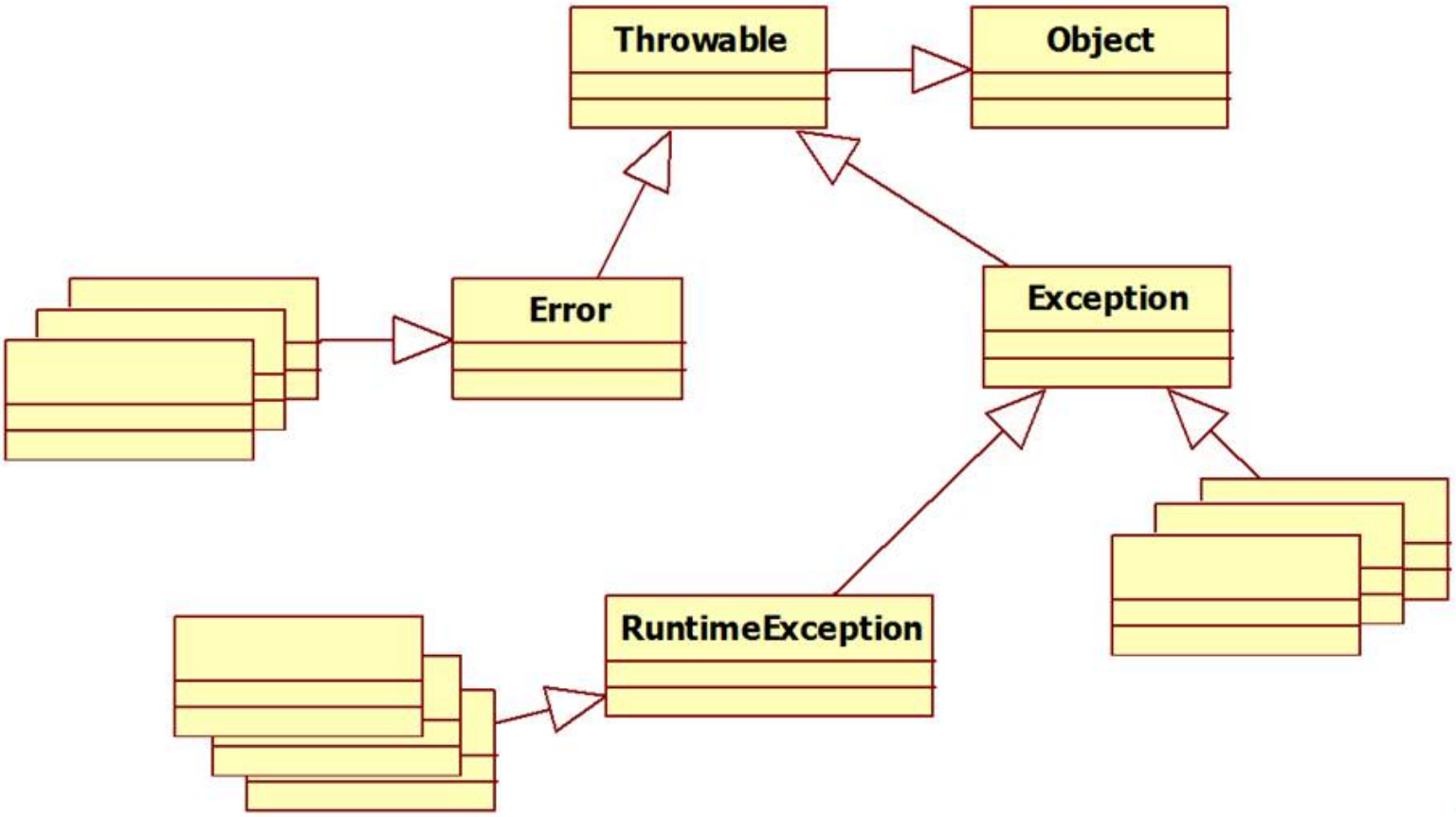
```
============ Start main() ============
Start doThis()
Start doThat()
java.lang.Exception
        at concept.c13.Test5.checkedException(Test5.java:6)
        at concept.c13.Test5.doThat(Test5.java:12)
        at concept.c13.Test5.doThis(Test5.java:20)
        at concept.c13.Test5.main(Test5.java:25)
End doThat()
End doThis()
============ End main() ============
```

Edited by Ruei-Jiun Tzeng

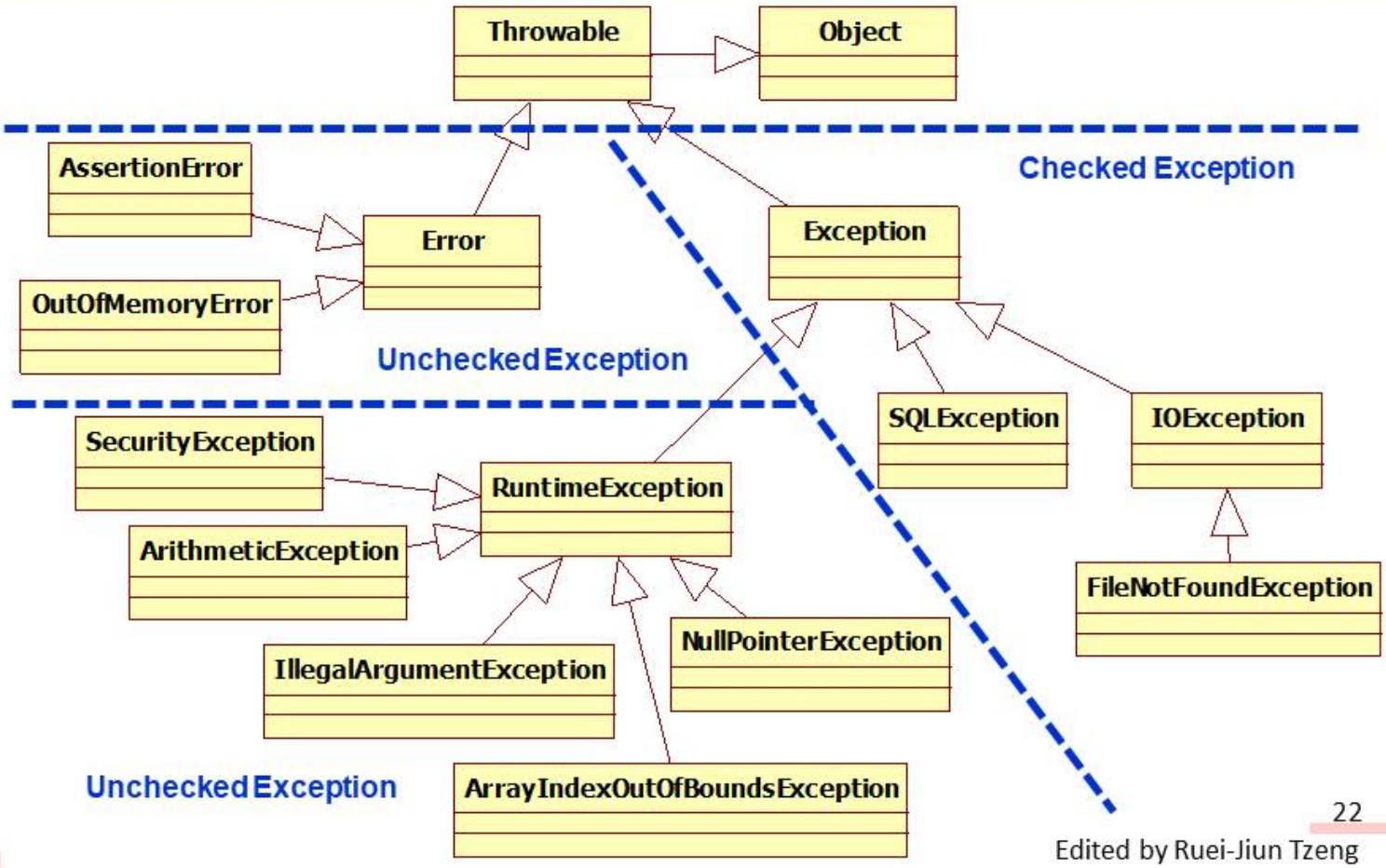
例外的始祖【Throwable】

- Throwable是Java內一種特別的類別：
  - try catch block 的【catch (...)】，只能放這種類別。
  - method 後的宣告【throws ...】，只能放這種類別。
  - 可以再分成 Exception、Error。

例外的繼承架構



認證考試常見例外類別


Throwable
Object
Checked Exception
AssertionError
Error
Exception
OutOfMemoryError
Unchecked Exception
SQLException
IOException
SecurityException
RuntimeException
ArithmeticException
FileNotFoundException
NullPointerException
IllegalArgumentException
Unchecked Exception
ArrayIndexOutOfBoundsException

4/4

處理Exception的好習慣

Best Practice (養成好習慣)

- Catch 真正的Exception，而不是父類別如Exception或Throwable。
- 檢查Exception，確認是否可以徹底的復原。
- 不需要catch所有Exception：
  - 程式的錯誤必須事先處理。
  - 確認「這個Exception是否是程式應該要處理的?」

Checked IOException

```
public class IOExceptionDemo {
    public static void main(String args[]) {
        try {
            testCheckedException();
        } catch (IOException e) {
            System.out.println(e);
        }
    }
    public static void testCheckedException() throws IOException {
        String path =
                System.getProperty("user.dir") + "/src/course/c13/temp/";
        File f = new File(path + "test.txt");
        f.createNewFile();
        System.out.println("File created or not: " + f.exists());
    }
}
```

Edited by Ruei-Jiun Tzeng

錯誤示範，問題在哪?

```
public class BadPractice {
    public static void main(String args[]) {
        try {
            testCheckedException();
        } catch (Exception e) {
            System.out.println("Failed as creating file!!");
        }
    }
    public static void testCheckedException() throws IOException {
        String path = System.getProperty("user.dir") + "/src/course/c13/temp/";
        File f = new File(path + "test.txt");
        f.createNewFile();
        System.out.println("File is created? " + f.exists());
        //deal with another logic
        int[] array = new int[4];
        array[4] = 100;
    }
}
```

Edited by Ruei-Jiun Tzeng

錯誤示範，問題在這!

- 出現Exception時只列印“Failed as creating file!!”，將導致真正原因無法發現。事實上也會出現“ArrayIndexOutOfBoundsException”。
- 不應該直接catch Exception。本例中可能出現的java.io.IOException，和java.lang.ArrayIndexOutOfBoundsException應該要分開處理。
- 建立file時，也有可能因為權限問題，而丟出另一個unchecked Exception： java.lang.SecurityException。

究竟有多少個 Exception?

```
public class MultiExceptionDemo {
    public static void createTempFile() throws IOException {
        String path = System.getProperty("user.dir") + "/src/course/c13/temp";
        System.out.println(path);
        File f = new File(path);
        File tf = File.createTempFile("ji", null, f);
        System.out.println("Temp file name: " + tf.getPath());
        int arr[] = new int[5];
        arr[5] = 25;
    }
}
```

可能的Exception

- IOException
  - 路徑的資料夾是read only，或不存在。
- IllegalArgumentException
  - 要建立暫存檔案，至少須提供前3個字元。
- ArrayIndexOutOfBoundsException
  - 超出Array長度。
- SecurityException
  - 檔案存取權限。

Catch Multiple Exceptions

```
public static void main(String args[]) {
    try {
        createTempFile();
    } catch (IOException ioe) {
        System.out.println(ioe);
    } catch (IllegalArgumentException iae) {
        System.out.println(iae);
    } catch (ArrayIndexOutOfBoundsException aiobe) {
        System.out.println(aiobe);
    } catch (SecurityException se) {
        System.out.println(se);
    } catch (Exception e) {
        System.out.println(e);
    }
}
```

Catch Multiple Exceptions

- 若catch的Exceptions間有繼承關係時，愈大的父類別Exception一定要在愈後面，避免在前面就被大的Exception所先攔截。
- 不依照此規則，無法通過編譯。

End ~~

Thank you!!