

1. 什麼是例外

<1>.指某些程式執行當中發生的狀況，而這些狀況會使得程式的流程異常中斷

<2>.如 嚴重的硬體錯誤(如磁碟壞軌)，單純的程式錯誤(如超出陣列索引範圍的元素)

<3>.作例外處理目的就是要避免程式異常中斷，有些十分嚴重的情形例如 JVM 系統毀損，硬碟故障，CPU 故障，這些難以處理的例外情形，就不是例外處理可以控制，此種稱為錯誤(Error)，我們稱為例外(Exception)的是一些較輕微而且還能處理的錯誤狀況

<4>.編譯程式所發生的錯誤是“編譯錯誤”而執行程式時所發生的錯誤才是我們所說的“例外”

RuntimeException 的六種例外 → unchecked

<pre>//ArithmeticException public static void 除以0的例外() { int a, b; b = 0; System.out.println("AA"); a = 10 / b; System.out.println("BB"); }</pre>	<pre>AA Exception in thread "main" java.lang.ArithmeticException: / by zero at java13_例外與斷言.Test13.除以0的例外(Test13.java:15) at java13_例外與斷言.Main.main(Main.java:6) Java Result: 1</pre>
<pre>//ArrayIndexOutOfBoundsException public static void 超出陣列索引的例外() { System.out.println("AA"); int[] a = new int[5]; a[10] = 100; System.out.println("BB"); }</pre>	<pre>AA Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10 at java13_例外與斷言.Test13.超出陣列索引的例外(Test13.java:23) at java13_例外與斷言.Main.main(Main.java:7) Java Result: 1</pre>
<pre>//StringIndexOutOfBoundsException public static void 超出字串索引的例外() { System.out.println("AA"); String a = "abcdef"; char b = a.charAt(7); System.out.println(b); System.out.println("BB"); }</pre>	<pre>AA Exception in thread "main" java.lang.StringIndexOutOfBoundsException: String index out of range: 7 at java.lang.String.charAt(String.java:646) at source.Test13.超出字串索引的例外(Test13.java:34) at source.Main.main(Main.java:8)</pre>
<pre>//NullPointerException public static void 空指標的例外() { System.out.println("AA"); Human1 人1 = null; 人1.名字 = "賴玉珊"; System.out.println("BB"); }</pre>	<pre>AA Exception in thread "main" java.lang.NullPointerException at java13_例外與斷言.Test13.空指標的例外(Test13.java:31) at java13_例外與斷言.Main.main(Main.java:8) Java Result: 1</pre>
<pre>//NumberFormatException public static void 格式錯的例外() { int x; System.out.println("AA"); x = Integer.parseInt("ABC"); System.out.println(x); }</pre>	<pre>AA Exception in thread "main" java.lang.NumberFormatException: For input string: "ABC" at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65) at java.lang.Integer.parseInt(Integer.java:492) at java.lang.Integer.parseInt(Integer.java:527) at java13_例外與斷言.Test13.格式錯的例外(Test13.java:39) at java13_例外與斷言.Main.main(Main.java:9) Java Result: 1</pre>
<pre>//ClassCastException public static void 多型的例外() { 動物 ani = new 狗(); 貓 cat; cat = (貓) ani; }</pre>	<pre>Exception in thread "main" java.lang.ClassCastException: java13_例外與斷言.狗 cannot be cast to java13_例外與斷言.貓 at java13_例外與斷言.Test13.多型的例外(Test13.java:48) at java13_例外與斷言.Main.main(Main.java:10) Java Result: 1</pre>

2. 處理例外的方法

<1>.自行處理→將例外捕捉→ try 區塊不能獨立存在，至少要有一個 catch 區塊或 finally 區塊

<2>.明白指出，將方法內產生的例外丟出

```
public static void 處理例外的語法() {  
    //      try { //不能只有一個 try 區塊  
    //  
    //      }  
    //      ///////////////////////////////////  
    try {  
  
    } catch (ArithmeticException e) {  
  
    }  
    //      ///////////////////////////////////  
    try {  
  
    } finally {  
  
    }  
    //      ///////////////////////////////////  
    try {  
  
    } catch (ArithmeticException e) {  
  
    } catch (NumberFormatException e) {  
  
    } catch (ArrayIndexOutOfBoundsException e) {  
  
    } finally {  
    }  
}
```

```
public static void 呼叫除以0的例外_自行處理() {  
    除以0的例外_自行處理();  
}  
public static void 除以0的例外_自行處理() {  
    int a, b;  
  
    try {  
        b = 0;  
        System.out.println("AA");  
        a = 10 / b; //當  
        System.out.println("CC");  
    } catch (ArithmeticException e) {  
        System.out.println(e);  
    } finally {  
        System.out.println("close");  
    }  
    System.out.println("BB");  
}
```

```
public static void 呼叫除以0的例外_丟出去1() {  
    try {  
        除以0的例外_丟出去1();  
    } catch (ArithmeticException e) {  
        System.out.println(e);  
    } finally {  
        System.out.println("一定會做");  
    }  
}  
  
public static void 除以0的例外_丟出去1() throws ArithmeticException {  
    int a, b;  
    b = 0;  
    System.out.println("AA");  
    a = 10 / b; //當  
    System.out.println("BB");  
}
```

```
public static void 呼叫除以0的例外_丟出去2() {  
    try {  
        除以0的例外_丟出去2();  
    } catch (ArithmeticException e) {  
        System.out.println(e);  
    } finally {  
        System.out.println("一定會做");  
    }  
}  
  
public static void 除以0的例外_丟出去2() throws ArithmeticException {  
    int a, b;  
  
    try {  
        b = 0;  
        System.out.println("AA");  
        a = 10 / b;  
    } finally {  
        System.out.println("最後心願");  
    }  
    System.out.println("BB"); //當掉了，不會做到  
}
```

3. 需不需要強制處理→用 compiler 來牽制

<1>.checked Exception

<2>.unchecked Exception

```
public static void checked的例外1() {  
    for (int i = 1; i <= 10; i++) {  
        System.out.println("i=" + i);  
  
        try {  
            Thread.sleep(2000);  
        } catch (InterruptedException e) {  
            System.out.println(e);  
        }  
    }  
}  
  
public static void checked的例外2() throws InterruptedException {  
    for (int i = 1; i <= 10; i++) {  
        System.out.println("i=" + i);  
        Thread.sleep(2000);  
    }  
}  
  
public static void unchecked的例外() {  
    int x;  
    System.out.println("AA");  
    x = Integer.parseInt("ABC");  
    System.out.println(x);  
}
```

4. 例外類別的建立→每個例外都是一個物件，當例外發生時，也就是產生了某個例外類別的物件實體

<1>.系統內建

(1).Java 內建的例外類別有很多，每一種類別都代表一種錯誤情況，但基本上可被 throw 丟出的例外物件，它的類別必定繼承自 “java.lang” Package 的 Throwable 類別

(2).各種例外類別並不在同一個 Package 中，而是散落在不同的 Package

<2>.使用者自定

(1).自訂一個例外類別，根據錯誤的情況，利用它來產生獨特的例外物件→原本必須繼承 Throwable 類別

(2).我們自訂的例外類別通常不是用來代表嚴重的 Error 所以一般只會讓它繼承自 Exception 類別定位為→checked

(3).我們自訂的例外若繼承自 RuntimeException 類別則定位為→unchecked

系統內建的例外類別

Exception classes 的階層 (The Exception Hierarchy)

- java.lang.Object
 - java.lang.Throwable
 - java.lang.Error //如 JVM 的錯誤，可以捕捉，但無法補救，故通常不處理，屬於 unchecked exception
 - java.lang.Exception //以下除了 RuntimeException，皆屬於 checked exceptions
 - java.lang.RuntimeException //程式的 bug，程式設計師自己小心寫作就可避免，屬於 unchecked exception
 - java.lang.ArithmeticException
 - java.lang.NullPointerException
 -
 - java.io.EOFException
 - FileOutFowerdException
 -
 - SQLException
 - 執行緒



使用者自訂

```
//歸類為 unchecked的例外
public class MyDefClass1 extends RuntimeException {

    public MyDefClass1() {
        //自動有 super();
    }
}

//歸類為 checked的例外
public class MyDefClass2 extends Exception {

    public MyDefClass2() {
        //自動有 super();
    }
}
```

5. 引發例外的方式

<1>.程式系統自動引發→用於引發系統的例外，系統已對應好每一種例外的狀況

<2>.使用者使用 throw 指令引發→用於引發自訂的例外，但要用於引發系統的例外也可以

```
public static void 系統引發1() { //unchecked
    int a = 10;
    int b = 0;
    int c;
    c = a / b;
}

public static void 使用者引發1() { //unchecked
    throw new ArithmeticException();
}

public static void 系統引發2() throws FileNotFoundException { //checked
    PrintWriter file = new PrintWriter("c://file.txt");
}

public static void 使用者引發2() throws FileNotFoundException { //checked
    throw new FileNotFoundException();
}
```

```
public static void 使用者引發3() { //unchecked
    throw new MyDefClass1();
}

public static void 使用者引發4() { //checked
    try {
        throw new MyDefClass2();
    } catch (MyDefClass2 e) {
        System.out.println(e);
    }
}

public static void 使用者引發5() throws MyDefClass2 { //checked
    throw new MyDefClass2();
}
```

6. 自行處理→補充

<1>.try 區塊內的程式碼引發了預期中的例外

→找到負責處理 exception 的 catch 區塊，再執行 finally 區塊，最後再執行正常敘述句

<2>.try 區塊內的程式碼未引發任何例外

→最後會去執行 finally 區塊，再執行正常敘述句

<3>.try 區塊內的程式碼引發了未設計去處理的例外

→找不到處理此 exception 的 catch 區塊，但 finally 區塊還是會執行，例外再丟給執行系統去處理，因此造成程式的中斷，正常敘述句不會被執行

```
public static void 自行處理的例外1() {  
    try {  
        throw new ArithmeticException(); //有補捉到 unchecked  
        // throw new NumberFormatException(); //會當掉  
        //throw new IOException(); // checked 一定要處理到  
    } catch (ArithmeticException e) {  
        System.out.println("除以0的例外發生");  
        // return;  
        //System.exit(0);  
    } catch (ArrayIndexOutOfBoundsException e) {  
        System.out.println("超出索引的例外發生");  
    } catch (NullPointerException e) {  
        System.out.println("沒有指標的例外發生");  
    } catch (ClassCastException e) {  
        System.out.println("多形轉換的例外發生");  
    } finally {  
        System.out.println("這裡是一定會做的區塊");  
    }  
    System.out.println("我是正常的程式碼");  
}
```

```
public static void 自行處理的例外2_多重捕捉() { // 7.0 版語法  
    try {  
        throw new ArithmeticException();  
        // throw new NumberFormatException();  
        //throw new IOException();  
    } catch (ArithmeticException | ArrayIndexOutOfBoundsException | ClassCastException e) {  
        System.out.println(e);  
    } finally {  
        System.out.println("這裡是一定會做的區塊");  
    }  
    System.out.println("我是正常的程式碼");  
}
```

```

public static void 自行處理的例外3() {
    try {
        //throw new ArithmeticException();
        throw new NumberFormatException();
        //throw new IOException();
    } catch (ArithmeticException e) {
        System.out.println("除以0的例外發生");
    } catch (RuntimeException e) {
        System.out.println(e);
    } finally {
        System.out.println("這裡是一定會做的區塊");
    }
    System.out.println("我是正常的程式碼");
}

```

//多重捕捉 不可以有繼承關係

```

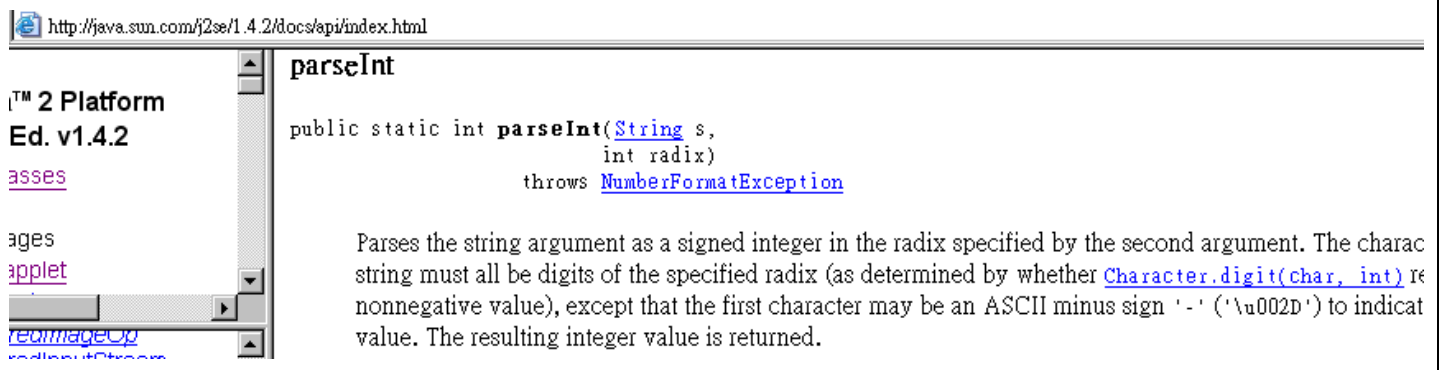
public static void 自行處理的例外4() {
    try {
        //throw new ArithmeticException();
        throw new NumberFormatException();
        //throw new IOException();
    } catch (ArithmeticException | RuntimeException e) {
        System.out.println(e);
    } finally {
        System.out.println("這裡是一定會做的區塊");
    }
    System.out.println("我是正常的程式碼");
}

```

7. 明白指出，將方法內產生的例外丟出 - 使用 throws 語法→補充

<1>.我們經常使用的 java 內建類別中，有許多擁有像這樣 “指明可能丟出例外的方法”

<2>.因此在呼叫這些方法時，最好要使用 try...catch 敘述句來處理例外



```

public static void 系統API丟出的例外() {

    int x = 120, divisor, y;
    try {
        divisor = Integer.parseInt(JOptionPane.showInputDialog("請輸入一個int值:"));
        y = x / divisor;
        System.out.println("y=" + y);
    } catch (NumberFormatException e) {
        System.out.println("錯誤:" + e.getMessage() + ";未填寫適當的除數!");
    } catch (ArithmeticException e) {
        System.out.println("錯誤:" + e.getMessage());
    }
    System.out.println("Bye Bye !");
    System.exit(0);
}

```




```
public static void unchecked丟出例外處理() {
    abc1();
}

public static void abc1() throws MyDefClass1 {

    throw new MyDefClass1(); //unchecked
}
```

```
public static void checked丟出例外處理() {
    try {
        abc2();
    } catch (MyDefClass2 e) {
        System.out.println("錯誤:" + e);
    }
}

public static void abc2() throws MyDefClass2 {

    throw new MyDefClass2(); //checked
}
```

8. 兩個例外只發生一個→同時間只會引發一個

```
public static void 呼叫兩個例外只發生一個_unchecked() {
    try {
        兩個例外只發生一個_unchecked();
    } catch (ArithmeticException e) {
        System.out.println(e);
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println(e);
    } finally {
        System.out.println("這裡是一定會做的區塊");
    }
    System.out.println("我是正常的程式碼");
}

public static void 兩個例外只發生一個_unchecked() throws ArithmeticException, ArrayIndexOutOfBoundsException {
    if ("a".equals("b")) {
        throw new ArithmeticException();
    } else {
        throw new ArrayIndexOutOfBoundsException();
    }
}
```

java.lang.ArrayIndexOutOfBoundsException

這裡是一定會做的區塊

我是正常的程式碼

```

public static void 呼叫兩個例外只發生一個_checked() {
    try {
        兩個例外只發生一個_checked();
    } catch (FileNotFoundException e) {
        System.out.println(e);
    } catch (EOFException e) {
        System.out.println(e);
    } finally {
        System.out.println("這裡是一定會做的區塊");
    }
    System.out.println("我是正常的程式碼");
}

public static void 兩個例外只發生一個_checked() throws FileNotFoundException, EOFException {
    if ("a".equals("b")) {
        throw new FileNotFoundException();
    } else {
        throw new EOFException();
    }
}

```

java.io.EOFException

這裡是一定會做的區塊

我是正常的程式碼

9. 兩個例外都會發生

```

public static void 呼叫兩個例外都會發生_unchecked1() {

    try {
        兩個例外都會發生_unchecked1();
    } catch (NullPointerException e) {
        System.out.println(e);
    } finally {
        System.out.println("這裡是一定會做的區塊2");
    }
    System.out.println("我是正常的程式碼2");
}

public static void 兩個例外都會發生_unchecked1() throws NullPointerException {
    try {
        throw new ArithmeticException();

    } catch (ArithmeticException e) {
        System.out.println(e);
        throw new NullPointerException();

    } catch (NullPointerException e) {
        System.out.println(e);
    } finally {
        System.out.println("這裡是一定會做的區塊1");
    }
    System.out.println("我是正常的程式碼1");
}

```

java.lang.ArithmeticException

這裡是一定會做的區塊1

java.lang.NullPointerException

這裡是一定會做的區塊2

我是正常的程式碼2

```
public static void 呼叫兩個例外都會發生_unchecked2() {
    兩個例外都會發生_unchecked2();
    System.out.println("我是正常的程式碼2");
}
```

```
public static void 兩個例外都會發生_unchecked2() {
    try {
        throw new ArithmeticException();

    } catch (ArithmeticException e) {
        System.out.println(e);
        try {
            throw new NullPointerException();

        } catch (NullPointerException f) {
            System.out.println(f);
        }
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println(e);
    } finally {
        System.out.println("這裡是一定會做的區塊1");
    }
    System.out.println("我是正常的程式碼1");
}
```

java.lang.ArithmeticException
java.lang.NullPointerException
這裡是一定會做的區塊1
我是正常的程式碼1
我是正常的程式碼2

```
public static void 呼叫兩個例外都會發生_checked1() {
    try {
        兩個例外都會發生_checked1();

    } catch (InterruptedException e) {
        System.out.println(e);
    } finally {
        System.out.println("這裡是一定會做的區塊2");
    }
    System.out.println("我是正常的程式碼2");
}

public static void 兩個例外都會發生_checked1() throws InterruptedException {
    try {
        throw new IOException();

    } catch (IOException e) {
        System.out.println(e);
        throw new InterruptedException();

    } catch (Exception e) {
        System.out.println(e);
    } finally {
        System.out.println("這裡是一定會做的區塊1");
    }
    System.out.println("我是正常的程式碼1");
}
```

java.io.IOException
這裡是一定會做的區塊1
java.lang.InterruptedException
這裡是一定會做的區塊2
我是正常的程式碼2

```
public static void 呼叫兩個例外都會發生_checked2() {  
    兩個例外都會發生_checked2();  
    System.out.println("我是正常的程式碼2");  
}  
  
public static void 兩個例外都會發生_checked2() {  
    try {  
        throw new IOException();  
    } catch (IOException e) {  
        System.out.println(e);  
    }  
    try {  
        throw new InterruptedException();  
    } catch (InterruptedException f) {  
        System.out.println(f);  
    }  
    } catch (Exception e) {  
        System.out.println(e);  
    } finally {  
        System.out.println("這裡是一定會做的區塊1");  
    }  
    System.out.println("我是正常的程式碼1");  
}
```

java.io.IOException
java.lang.InterruptedException
這裡是一定會做的區塊1
我是正常的程式碼1
我是正常的程式碼2

10. 自訂例外類別

```
//父親
//class Exception extends Throwable{

//    private String detailMessage; //繼承自 Throwable
//
//    public Exception() {
//    }
//
//    public Exception(String detailMessage) {
//        this.detailMessage = detailMessage;
//    }
//
//    public String getMessage() { //繼承自 Throwable
//        return detailMessage;
//    }
//}
```

```
public class MyDefClass1 extends RuntimeException {

    public MyDefClass1() {
        //自動有 super();
    }

    public MyDefClass1(String msg) {
        super(msg); //呼叫父類別的建構子
    }

    public String getMessage() {
        return "例外訊息:" + super.getMessage();
    }
}
```

```
public class MyDefClass2 extends Exception {

    public MyDefClass2() {
        //自動有 super();
    }

    public MyDefClass2(String msg) {
        super(msg); //呼叫父類別的建構子
    }

    public String getMessage() {
        return "例外訊息:" + super.getMessage();
    }
}
```

副程式自己處理例外→不合時宜

```
public static void 呼叫自訂的例外類別_自行處理() {
    自訂的例外類別_自行處理();
}

public static void 自訂的例外類別_自行處理() {
    try {
        throw new MyDefClass2("b 不可以=0");
    } catch (MyDefClass2 e) {
        System.out.println(e.getMessage());
    }
    System.out.println("我是正常的程式碼");
}
```

例外訊息:b 不可以=0
我是正常的程式碼

副程式採用丟出去的方式處理例外→正確處理方式

```
public static void 呼叫自訂的例外類別_丟出去() {
    try {
        自訂的例外類別_丟出去();
    } catch (MyDefClass2 e) {
        System.out.println(e.getMessage());
    }
}

public static void 自訂的例外類別_丟出去() throws MyDefClass2 {
    throw new MyDefClass2("b 不可以=0");
}
```

例外訊息:b 不可以=0

```
public static void 呼叫sumxy() {
    try {
        sumxy(12, 3);
    } catch (MyDefClass2 e) {
        System.out.println(e.getMessage());
    }
}

public static void sumxy(int x, int y) throws MyDefClass2 {
    int sum = 0;
    sum = x + y;
    if (sum > 10) {
        throw new MyDefClass2("兩數相加不能大於10");
    }
    System.out.println("sum=" + sum);
}
```

11. 處理例外要注意的事

<1>.例外類別排列要由小到大

<2>.補抓例外不一定要等於例外但要能含蓋例外

<3>.覆蓋時，子類別的方法所丟的例外規則要是父親的子集→checked 才算，unchecked 不受限

<4>.多型處理例外要注意的事

例外類別排列要由小到大

```
public static void 例外的類別名稱範圍要由小到大1() {
    try {
        throw new NullPointerException();
    } catch (NullPointerException e) {
        System.out.println("沒有指標的例外發生");
    } catch (Exception e) {
        System.out.println("AAAAAA");
    } catch (Throwable e) {
        System.out.println("BBBBBB");
    } finally {
        System.out.println("這裡是一定會做的區塊");
    }
}
```

```
public static void 例外的類別名稱範圍要由小到大2() {
    try {
        throw new NullPointerException();
    } catch (Throwable e) {
        System.out.println("BBBBBB");
    } catch (Exception e) {
        System.out.println("AAAAAA");
    } catch (NullPointerException e) {
        System.out.println("沒有指標的例外發生");
    } finally {
        System.out.println("這裡是一定會做的區塊");
    }
}
```

含蓋例外

```
public static void 含蓋例外1() {
    try {
        throw new InterruptedException();
    } catch (Exception e) {
        System.out.println(e);
    } catch (Throwable e) {
        System.out.println(e);
    } finally {
        System.out.println("這裡是一定會做的區塊");
    }
}
```

```
public static void 含蓋例外2() {
    try {
        含蓋例外引發2();
    } catch (Exception e) {
        System.out.println(e);
    } finally {
        System.out.println("這裡是一定會做的區塊");
    }
}

public static void 含蓋例外引發2() throws InterruptedException {
    throw new InterruptedException();
}
```

覆蓋時，子類別的方法所丟的例外規則

1. 子類別可不丟
2. 可丟 父類別的子集 → 小於等於父類別
3. 如果 父類別沒丟，子類別也不能丟

```
//自訂例外
class ExcpA extends Exception {
}

class ExcpB extends ExcpA {
}

class ExcpC extends Exception {
}
```

Exception > ExcpA > ExcpB
Exception > ExcpC

```
public class ExceptionDemo1P {

    public void abc() throws ExcpA, ArrayIndexOutOfBoundsException {
    }
}

class ExceptionDemo1C extends ExceptionDemo1P {

    //可丟 ExcpA 或 ExcpB 但不能丟 ExcpC 或 不丟也可以
    public void abc() throws ExcpB, NullPointerException {
    }
}
```

父 ExcpA
子 ExcpA
ExcpB
不丟

父 ExcpB
子 ExcpB
不丟

父 沒丟
子 不能丟

多形處理例外要注意的事

```
class ExceptionDemo2P { //父

    public void abc() throws ExcpA {
    }
}

class ExceptionDemo2C extends ExceptionDemo2P { //子

    public void abc() { //子類別可以不丟
    }
}
```



```
//Compiler階段會認為 a 是 ExceptionDemo2P的型態
public void 多型的例外處理1() {
    ExceptionDemo2P a = new ExceptionDemo2C();
    a.abc();
}

public void 多型的例外處理2() {
    ExceptionDemo2P a = new ExceptionDemo2C();
    try {
        a.abc();
    } catch (ExcpA e) {
    }
}

public void 多型的例外處理3() throws ExcpA {
    ExceptionDemo2P a = new ExceptionDemo2C();
    a.abc();
}
```

12. 處理例外的方法

- <1>.toString()方法→會顯示例外的類別名稱和產生例外的原因
- <2>.getLocalizedMessage() · getMessage()方法→會顯示例外發生的原因但不會把例外的名稱顯示出來
- <3>.printStackTrace()方法→會把產生這個例外的相關類別名稱以及是第幾行程式碼產生這個例外的

```
public static void 處理例外訊息的方法() {
    try {
        InputStream f = new FileInputStream("test.txt");
    } catch (FileNotFoundException e) {
        System.out.println("===toString()===");
        System.err.println(e.toString()); //自動幫我們呼叫 toString()這個方法

        System.out.println("===getLocalizedMessage()===");
        System.err.println(e.getLocalizedMessage());
        System.out.println("===getMessage()===");
        System.err.println(e.getMessage());

        System.out.println("===printStackTrace()===");
        e.printStackTrace(); //像自然引發，會有引發的行數
    }
}
```

==toString()==

java.io.FileNotFoundException: test.txt (系統找不到指定的檔案。)

test.txt (系統找不到指定的檔案。)

==getLocalizedMessage()==

==getMessage()==

==printStackTrace()==

test.txt (系統找不到指定的檔案。)

java.io.FileNotFoundException: test.txt (系統找不到指定的檔案。)

at java.io.FileInputStream.open(Native Method)

at java.io.FileInputStream.<init>(FileInputStream.java:106)

at java.io.FileInputStream.<init>(FileInputStream.java:66)

at j1201.Test.處理例外訊息的方法(Test.java:12)

at j1201.Main.main(Main.java:6)

13. Assertion 斷言

<1>.把斷言想成是進階的 Exceptionhandling · Exception handling 會丟出 exception 和 error 但斷言只會丟出 Assertion Error

<2>.例外是程式中非預期的錯誤→例外處理是在這些錯誤發生時所採取的措施。

預期程式中應該會處於何種狀態，例如某些情況下某個值必然是多少，稱之為斷言 (Assertion)

斷言有兩種情況：成立或不成立。當預期結果與實際執行相同時，斷言成立，否則斷言失敗。

<3>.斷言的語法：

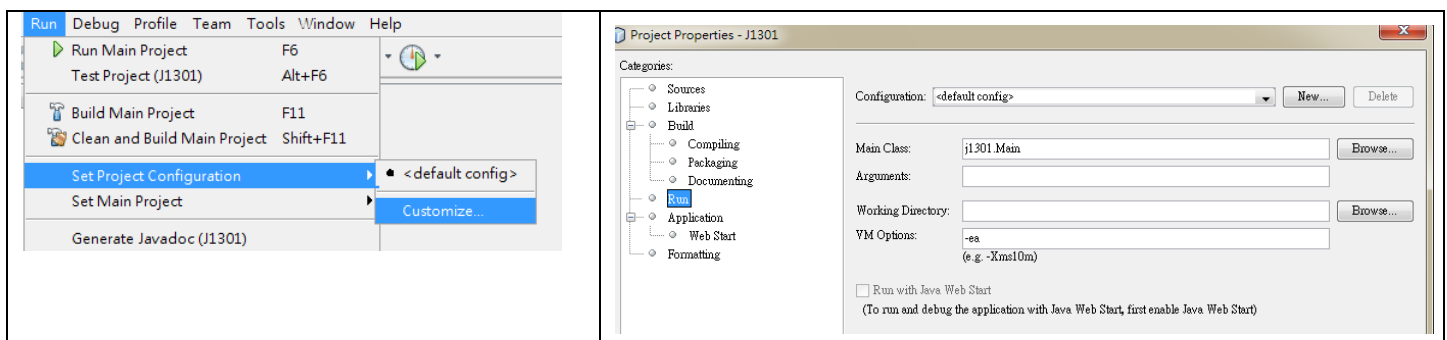
assert (expression1);

assert (expression1): expression2;

<4>.斷言功能是在 JDK 1.4 之後提供的，由於斷言使用 **assert** 作為關鍵字，為了避免以前在 JDK 1.3 或更早之前版本的程式使用了 **assert** 作為變數，而導致的名稱衝突問題，預設上執行時是不啟動斷言檢查的，如果您要在執行時啟動斷言檢查，可以使用 **-enableassertions** 或是 **-ea** 引數

java -ea AssertionDemo

<5>.netbeans 的斷言開啟方式：



```
public class AssertionDemo1 {

    public static void main(String[] args) {
        int x = 5;
        //
        //
        //
        assert x >= 10 : "x 沒有 >= 10 ";
        System.out.println("abc");
    }
}
```

```
Exception in thread "main" java.lang.AssertionError: x 沒有 >= 10
    at j1301.AssertionDemo1.main(AssertionDemo1.java:10)
Java Result: 1
```

14. 斷言的使用

<1>.什麼地方不要使用斷言

- (1).public method 的 arguments checking→依規定 public 方法的參數永遠都要 precondition 檢查，但斷言卻不一定會開啟
- (2).side effects→斷言不一定會開啟，就會影響到變數的值

```
public class AssertionDemo2 {

    int i = 0;

    public static void main(String[] args) {
        AssertionDemo2 t = new AssertionDemo2();
        t.g();
        System.out.println(t.i);
    }
    private int g() {
        //
        //
        assert ++i < 10;
        return i;
    }
}
```

```
public class AssertionDemo3 {

    int i = 0;

    private boolean check() {
        i = 1;
        return true;
    }

    public static void main(String[] args) {
        AssertionDemo3 t = new AssertionDemo3();
        assert t.check();
        System.out.println(t.i);
    }
}
```

<2>.什麼地方可以使用斷言→private 方法參數的檢查

- (1).precondition 預先的檢查→以確保在進行某些操作之前，一個先決條件已經滿足
- (2).postcondition 事後的檢查→以確保在某些操作後，一個既定的條件仍然滿足
- (3).流程不變性的檢查

```
private int getDays(int year, int month) {
    assert (month >= 1) && (month <= 12) : month;
    int day = 0;
    //
    //計算某年某月有幾天
    //
    assert (day >= 28) && (month <= 31) : day;
    return day;
}

public void f(int x) {
    if ((x % 2) == 0) {
        System.out.println("even");
        return;
    } else if ((x % 2) == 1) {
        System.out.println("odd");
        return;
    }
    assert false : "我斷言 程式不會跑到這裡來";
}
```

```

public void g(int x) {
    switch (x) {
        case MALE:
            //.....
            break;

        case FEMALE:
            //.....
            break;

        default:
            assert false :
                "我斷言 x 若不是 MALE，就一定是 FEMALE。" + x;
    }
    //.....
}

public void h(int x) {
    if (x > 0) {
        //.....
    } else {
        assert (x < 0) : "我斷言 x 一定小於 0。" + x;
        //.....
    }
    //.....
}

```

15. 斷言與例外

```

public class AssertionDemo5 {

    int i = 0;

    public boolean check() {
        i /= 0;
        return false;
    }

    public static void main(String[] args) {
        AssertionDemo5 t = new AssertionDemo5();
        assert t.check();
        System.out.println(t.i);
    }
}

```

Exception in thread "main" java.lang.ArithmeticException: / by zero
 at source.AssertionDemo5.check(AssertionDemo5.java:8)
 at source.AssertionDemo5.main(AssertionDemo5.java:14)

```

public class AssertionDemo6 {

    int i = 0;

    public int message() {
        return i /= 0;
    }

    public static void main(String[] args) {
        AssertionDemo6 t = new AssertionDemo6();
        assert false : t.message();
        System.out.println(t.i);
    }
}

```

Exception in thread "main" java.lang.ArithmeticException: / by zero
 at source.AssertionDemo6.message(AssertionDemo6.java:8)
 at source.AssertionDemo6.main(AssertionDemo6.java:13)