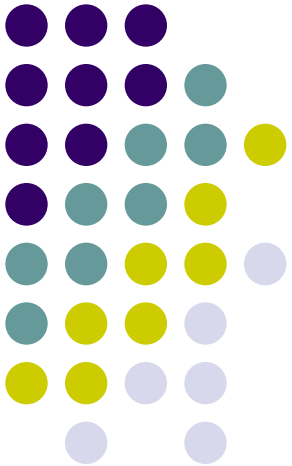
第十三章
例外處理

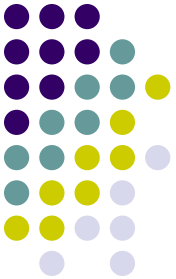
瞭解什麼是例外處理

認識例外類別的繼承架構

認識例外處理的機制

學習如何撰寫例外類別





例外的基本觀念

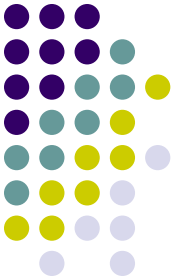
- 在撰寫程式時常見的幾種情況：

  (1) 要開啟的檔案並不存在

  (2) 存取陣列時，陣列的索引值超過陣列容許的範圍

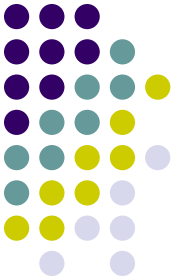
  (3) 原本預期使用者由鍵盤輸入的是整數，但使用者輸入的卻是英文字母

  - 這類不尋常的狀況稱為「例外」（exception）
  - 在Java中，所有的例外都是以類別的型態存在



例外處理的優點

- 易於使用
- 可在例外發生 時加入相對應的措施，使程式能正常結束
- 允許我們拋出例外
- 不會拖慢執行速度
- 增進程式的穩定性及效率



簡單的例外範例

- Ch13_1是個錯誤的程式：

```
01  // Ch13_1, 索引值超出範圍
02  public class Ch13_1{
03     public static void main(String[] args){
04        int arr[]=new int[5];            // 容許 5 個元素
05        arr[10]=7;                       // 索引值超出容許範圍
06        System.out.println("end of main()!!");
07     }
08  }
```

執行結果：

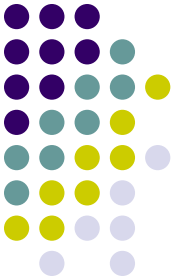
陣列索引值超出範圍

預設例外處理機制會依下面的程序做處理：
(1) 拋出例外
(2) 停止程式執行

- 執行到第5行時，會產生下列的錯誤訊息：

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 10 out
of bounds for length 5
        at Ch13_1.main(Ch13_1.java:5)
```

Array Index Out Of Bounds Exception，是 "陣列索引值超出範圍的例外" 之意



例外處理的語法

- 例外處理是由 try、 catch與finally所組成的程式區塊，其語法如下：

例外處理的語法

```
try{
    // 要檢查的程式敘述;
}
catch(例外類別 變數名稱){
    // 例外發生時的處理敘述;
}
finally{
    // 一定會執行的程式碼;
}
```

try 區塊

catch 區塊

finally 區塊

例外處理的順序

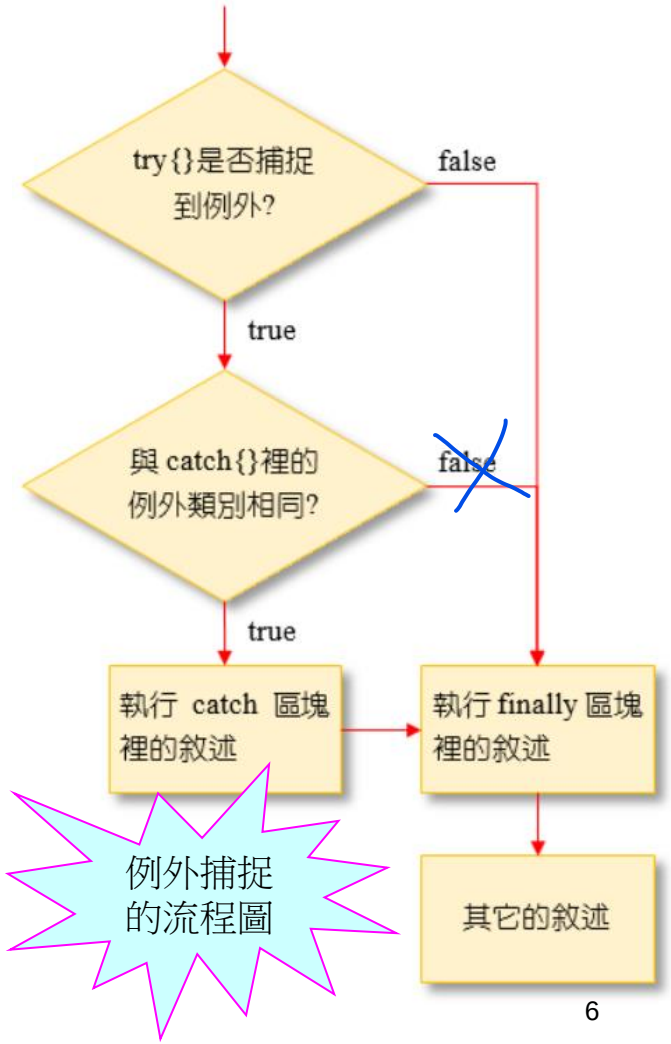
- 例外處理的順序：

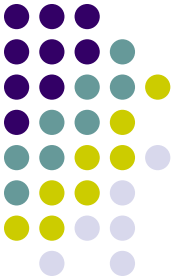
(1) try區塊若有例外發生，程式的執行便中斷，並拋出“由例外類別所產生的物件"

(2) 拋出的物件如果屬於catch() 括號內欲捕捉的例外，則catch會捕捉此例外，然後進到catch的區塊裡繼續執行

(3) 無論try程式區塊是否有捕捉到例外，或者捕捉到的例外是否與catch() 括號裡的例外相同，最後一定會執行finally區塊裡的程式碼

(4) finally的區塊執行結束後，程式再回到try-catch-finally區塊後的地方繼續執行





例外處理的實例

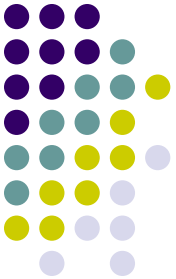
- Ch13_2是例外處理的範例：

```
01  // Ch13_2, 例外的處理
02   public class Ch13_2{
03     public static void main(String[] args){
04        try{                  // 檢查這個程式區塊的程式碼
05           int arr[]=new int[5];
06           arr[10]=7;
07        }
08        catch(ArrayIndexOutOfBoundsException e){
09           System.out.println("index out of bound!!");
10        }
11        finally{              // 這個區塊的程式碼一定會執行
12           System.out.println("this line is always executed!!");
13        }
14        System.out.println("end of main()!!");
15     }
16  }
```

如果拋出例外，便執行此區塊的程式碼

執行結果：

```
index out of bound!!
this line is always executed!!
end of main()!!
```



例外類別的變數

- 捕捉到例外時，例外類別會建立一個物件 e：

```
01  // Ch13_3, 例外訊息的擷取
02  public class Ch13_3{
03     public static void main(String[] args){
04        try{
05           int arr[]=new int[5];
06           arr[10]=7;
07        }
08        catch(ArrayIndexOutOfBoundsException e){
09           System.out.println("index out of bound!!");
10           System.out.println("Exception="+e);    // 顯示例外訊息
11        }
12        System.out.println("end of main()!!");
13     }
14  }
```

補捉到例外時所建立的類別變數

省略finally區塊程式依然可以運作

執行結果：

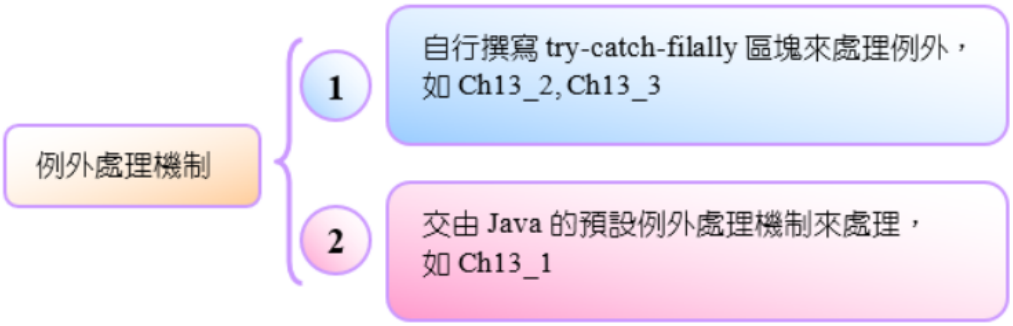
```
index out of bound!!
Exception=java.lang.ArrayIndexOutOfBoundsException: Index 10 out of
bounds for length 5
end of main()!!
```

例外處理機制的回顧

- 例外發生時，通常有二種方法來處理
  - 一種是交由預設的例外處理機制做處理，如

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 10 out
of bounds for length 5
        at Ch13_1.main(Ch13_1.java:5)
```

  - 另一種方式是自行撰寫try-catch-finally區塊來捕捉例外
- 下圖繪出例外處理機制的選擇流程：

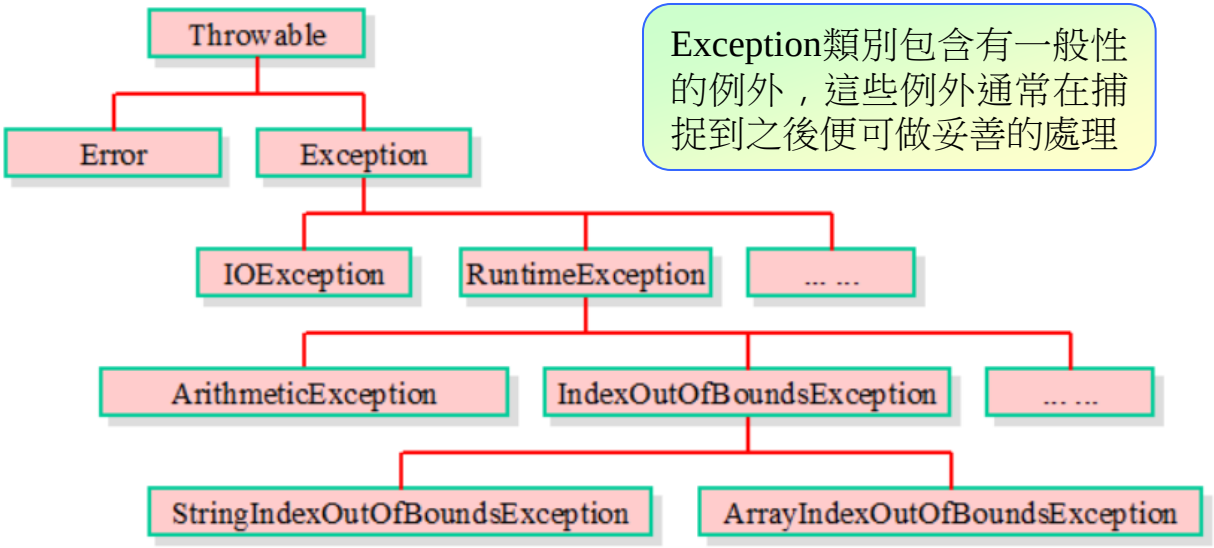


Throwable類別

- 例外類別可分為兩大類：
  - java.lang.Exception類別
  - java.lang.Error類別

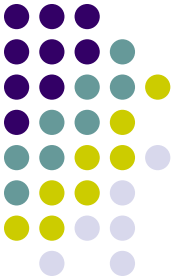
它們均繼承自
java.lang.Throwable類別

- 下圖為Throwable類別的繼承關係圖



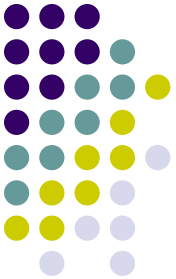
Exception類別包含有一般性的例外，這些例外通常在捕捉到之後便可做妥善的處理

Error類別專門用來處理嚴重影響程式執行的錯誤，通常不會設計程式碼去捕捉這類的錯誤



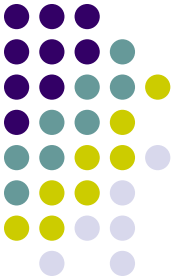
不受檢查 (unchecked) 的例外

- 「不受檢查 (unchecked) 的例外」:
  - RuntimeException
  - Error java.lang.Error類別
- 由程式錯誤或系統錯誤引起的，不建議在程式中捕捉並處理它們



受檢查 (checked) 的例外

- 其他繼承自 Exception 類別的例外在程式碼中必須明確地處理
  - 透過 try-catch 區塊
  - 使用 throws 拋出例外
- IOException 例外屬於受檢查的例外
  - 在進行輸入或輸出等工作時，因錯誤而拋出的例外，例如檔案存取錯誤

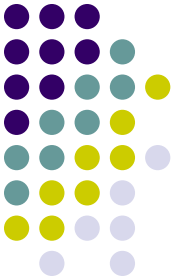


catch() 括號的限制

- 例外發生時，catch() 只能接收由Throwable類別的子類別所產生的物件

只能接收由 Throwable 類別的子類別所產生的物件

```
catch( ArrayIndexOutOfBoundsException e )
{
   System.out.println("index out of bound!!");
   System.out.println("Exception="+e);       // 顯示例外訊息
}
```



捕捉例外(1/2)

- 想捕捉一種以上的例外，可針對它們撰寫catch()：

```
01  try{
02     // try 區塊的程式碼
03  }
04  catch(ArrayIndexOutOfBoundsException e){
05      // 捕捉到 ArrayIndexOutOfBoundsException 例外所執行的程式碼
06  }
07  catch(ArithmeticException e){
08       // 捕捉到 ArithmeticException 例外所執行的程式碼
09  }
```

- 想捕捉所有的例外，可以利用Exception例外，如：

```
01  catch(Exception e){
02      // 捕捉任何例外所執行的程式碼
03  }
```

捕捉例外(2/2)

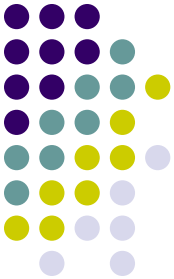
- 想捕捉多個例外時，範圍小的例外要排放在前面的catch()區塊，範圍大的例外要排放在後面的catch()區塊

```
01  try{
02     // try 區塊的程式碼
03  }
04  catch(ArrayIndexOutOfBoundsException e){
05      // 捕捉到 ArrayIndexOutOfBoundsException 例外所執行的程式碼
06  }
07  catch(Exception e){
08      // 捕捉到 Exception 例外所執行的程式碼
09  }
```

範圍較小的例外要排在前面的 catch()區塊裡

範圍較大的例外要排在後面的 catch()區塊裡

如果第7行與第4行互換，不管遇到何種例外都會直接被catch(Exception e) 補捉



例外的拋出

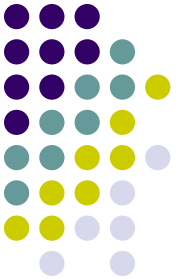
- 拋出例外有下列兩種方式：

  (1) 於程式中拋出例外

  (2) 指定函數拋出例外

- 於程式中拋出例外時的語法如下：

拋出例外處理的語法
throw 由例外類別所產生的物件;



於程式中拋出例外

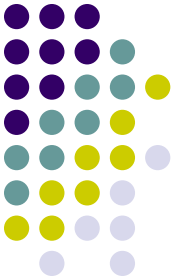
- Ch13_4是於程式中拋出例外的範例：

```
01  // Ch13_4, 於程式中拋出例外
02  public class Ch13_4{
03     public static void main(String[] args){
04        int a=4,b=0;
05
06        try{
07          if(b==0)
08             throw new ArithmeticException();     // 拋出例外
09          else
10             System.out.println(a+"/"+b+"="+a/b);// 若無拋出例外，則執行此行
11        }
12        catch(ArithmeticException e){
13           System.out.println(e+" throwed");
14        }
15     }
16  }
```

throw關鍵字所接的是「由例外類別所產生的物件」，因此必須使用new關鍵字產生物件

執行結果：

```
java.lang.ArithmeticException throwed
```



系統自動拋出例外

- Ch13_5是讓系統自動拋出例外的驗證：

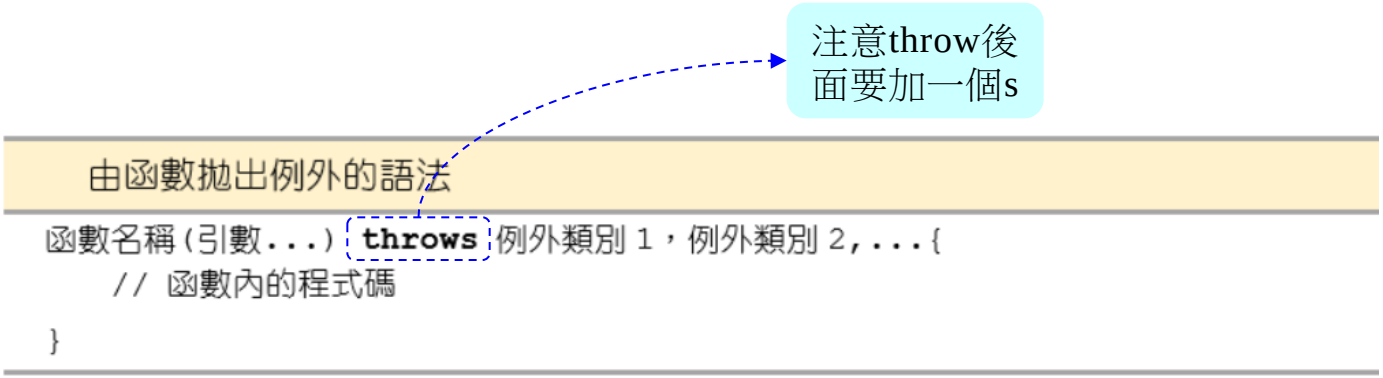
```
01  // Ch13_5, 讓系統自動拋出例外
02  public class Ch13_5{
03     public static void main(String[] args){
04        int a=4,b=0;
05
06        try{
07            System.out.println(a+"/"+b+"="+a/b);
08        }
09        catch(ArithmeticException e){
10           System.out.println(e+" throwed ");
11        }
12     }
13  }
```

執行結果：

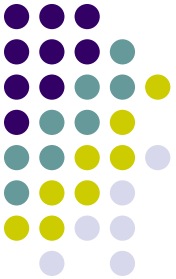
```
java.lang.ArithmeticException throwed: / by zero throwed
```

由函數拋出例外

- 由函數拋出例外的語法：



- 在函數的內部拋出例外，是使用關鍵字「throw」
- 如果是指定要由函數拋出例外，則使用「throws」



指定函數拋出例外

- Ch13_6是指定由函數來拋出例外的範例

```
01  // Ch13_6, 指定函數拋出例外
02  public class Ch13_6{
03     public static void aaa(int a,int b) throws ArithmeticException{
04        int c;
05        c=a/b;
06        System.out.println(a+"/"+b+"="+c);
07     }
08
09     public static void main(String args[]){
10        try{
11           aaa(4,0);
12        }
13        catch(ArithmeticException e){
14           System.out.println(e+" throwed");
15        }
16     }
17  }
```

指定由函數aaa拋出例外

執行結果：

```
java.lang.ArithmeticException: / by zero throwed
```

不同類別的函數拋出例外

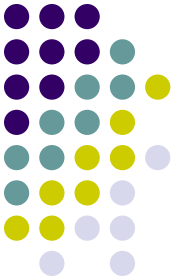
```
01  // Ch13_7, 從不同類別內的函數拋出例外
02  class Test{
03      public static void aaa(int a,int b) throws ArithmeticException{
04          int c=a/b;
05          System.out.println(a+"/"+b+"="+c);
06      }
07  }
08
09  public class Ch13_7{
10     public static void main(String args[]){
11        try{
12           test.aaa(4,0);
13        }
14        catch(ArithmeticException e){
15           System.out.println(e+" throwed");
16        }
17     }
18  }
```

從不同類別內的函數(aaa)拋出例外

Ch13_7說明如何從不同類別裡的函數裡拋出例外

執行結果：

```
java.lang.ArithmeticException: / by zero throwed
```

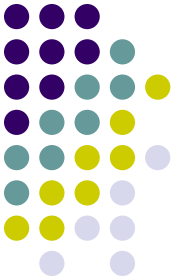


自行撰寫例外

- 自己設計的例外類別必須繼承Exception類別
  - 自行撰寫例外類別的語法如下：

撰寫自訂例外類別的語法

```
class  例外類別名稱 extends Exception{
   // 定義類別裡的各種成員
}
```



自行撰寫例外的範例 (1/2)

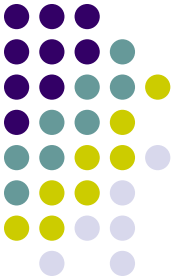
- 以一個範例來說明如何定義自己的例外類別：

```
01  // Ch13_8, 定義自己的例外類別
02  class CircleException extends Exception{  // 定義自己的例外類別
03  }
04
05  class Circle{              // 定義類別 Circle
06     private double radius;
07     public void setRadius(double r) throws CircleException{
08        if(r<0){
09           throw new CircleException();          // 拋出例外
10        }
11        else
12           radius=r;
13     }
14
15     public void show(){
16        System.out.println("area="+3.14*radius*radius);
17     }
18  }
19
```

必須繼承自Exception類別

由函數拋出例外

接續下一頁



自行撰寫例外的範例 (2/2)

```
20  public class Ch13_8{
21     public static void main(String[] args){
22        Circle c1=new Circle();
23        try{
24           c1.setRadius(-2.0);
25        }
26        catch(CircleException e){    // 捕捉由 setRadius()拋出的例外
27           System.out.println(e+" throwed");
28        }
29        c1.show();
30     }
31  }
```

執行結果：

```
CircleException throwed
area=0.0
```

拋出輸入型態不合的例外

- 程式中預設要讓使用者輸入整數，使用者卻輸入英文字母，會拋出InputMismatchException例外後中斷執行

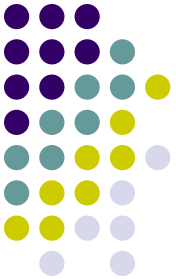
```
01  // Ch13_9, 捕捉 InputMismatchException 例外
02  import java.util.*;
03  public class Ch13_9{
04      public static void main(String[] args){
05          int num;
06          Scanner scn=new Scanner(System.in);
07          try{
08              System.out.print("請輸入一個整數: ");   // 輸入整數
09              num=scn.nextInt();
10              System.out.println("num="+num);
11          }
12          catch(Exception e){                        // 捕捉所有的例外
13              System.out.println("拋出"+e+"例外");    // 印出例外的種類
14          }
15          scn.close();
16      }
17  }
```

執行結果：

```
請輸入一個整數: k
拋出 java.util.InputMismatchException 例外
```

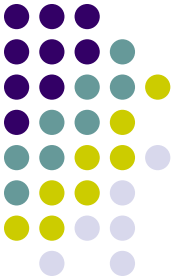
執行過程中刻意輸入英文字母，方便觀察拋出的例外種類

InputMismatchException的英文分解，即Input Mismatch Exception，表示輸入的資料型態不合之義



IOException例外類別

- IOException是處理有關輸入/輸出的例外，例如：
  - 讀取檔案未完成便被終止
  - 讀不到指定的檔案
- IOException例外的處理方式：
  - 直接由main() 拋出例外，讓Java預設的例外處理機制處理
  - 在程式碼內撰寫try-catch區塊來捕捉拋出的IOException例外

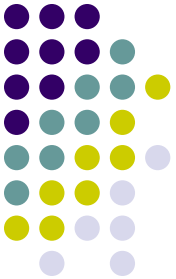


IOException例外的處理方式(1/2)

- 由main() 拋出IOException例外直接由main() 拋出例外，讓Java預設的例外處理機制處理

```
01  // 從鍵盤輸入字串
02  import java.io.*;
03  public class test
04  {
05     public static void main(String args[]) throws IOException{
..        ...
16     }
17  }
```

由 main() 拋出例外讓系統預設的例外處理機制來處理



IOException例外的處理方式(2/2)

- 撰寫try-catch區塊捕捉拋出的IOException例外

```
01  // Ch13_10, 撰寫 try-catch 區塊來捕捉 IOException 例外
02  import java.io.*;                        // 載入 java.io 類別庫裡的所有類別
03  public class Ch13_10{
04     public static void main(String[] args){
05        BufferedReader buf;
06        String str;
07
08        buf=new BufferedReader(new InputStreamReader(System.in));
09        try{
10           System.out.print("Input a string: ");
11           str=buf.readLine();
12           System.out.println("string= "+str);        // 印出字串
13        }
14        catch(IOException e){}
15     }
16  }
```

捕捉IOException例外

執行結果：

```
Input a string: Hello Java!
string= Hello Java!
```


-The End-