Java程式設計進階
例外處理

鄭安翔
ansel_cheng@hotmail.com

課程大綱

1) **例外機制**
   - **錯誤回報**
   - **例外分類**
   - **例外處理機制**
2) 例外處理機制
3) 例外處理進階

錯誤回報

- 錯誤回報
  - 當程式發生無法執行的狀態，系統停止執行，並於命令提示字元顯示錯誤訊息

```
public class AddArguments{
    public static void main(String[] args)
        int sum = 0;
        for(int i=0;
            sum += Inte
        }
        System.out.pr
    }
}
```

系統管理員: 命令提示字元

```
C:\JavaClass>java AddArguments 1 2 3 4
sum = 10

C:\JavaClass>
```

系統管理員: 命令提示字元

```
C:\JavaClass>java AddArguments 1 2 3
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
        at AddArguments.main(AddArguments.java:5)

C:\JavaClass>
```

系統管理員: 命令提示字元

```
C:\JavaClass>java AddArguments 1 two 3 4
Exception in thread "main" java.lang.NumberFormatException: For input string: "t
wo"
        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 AddArguments.main(AddArguments.java:5)

C:\JavaClass>
```

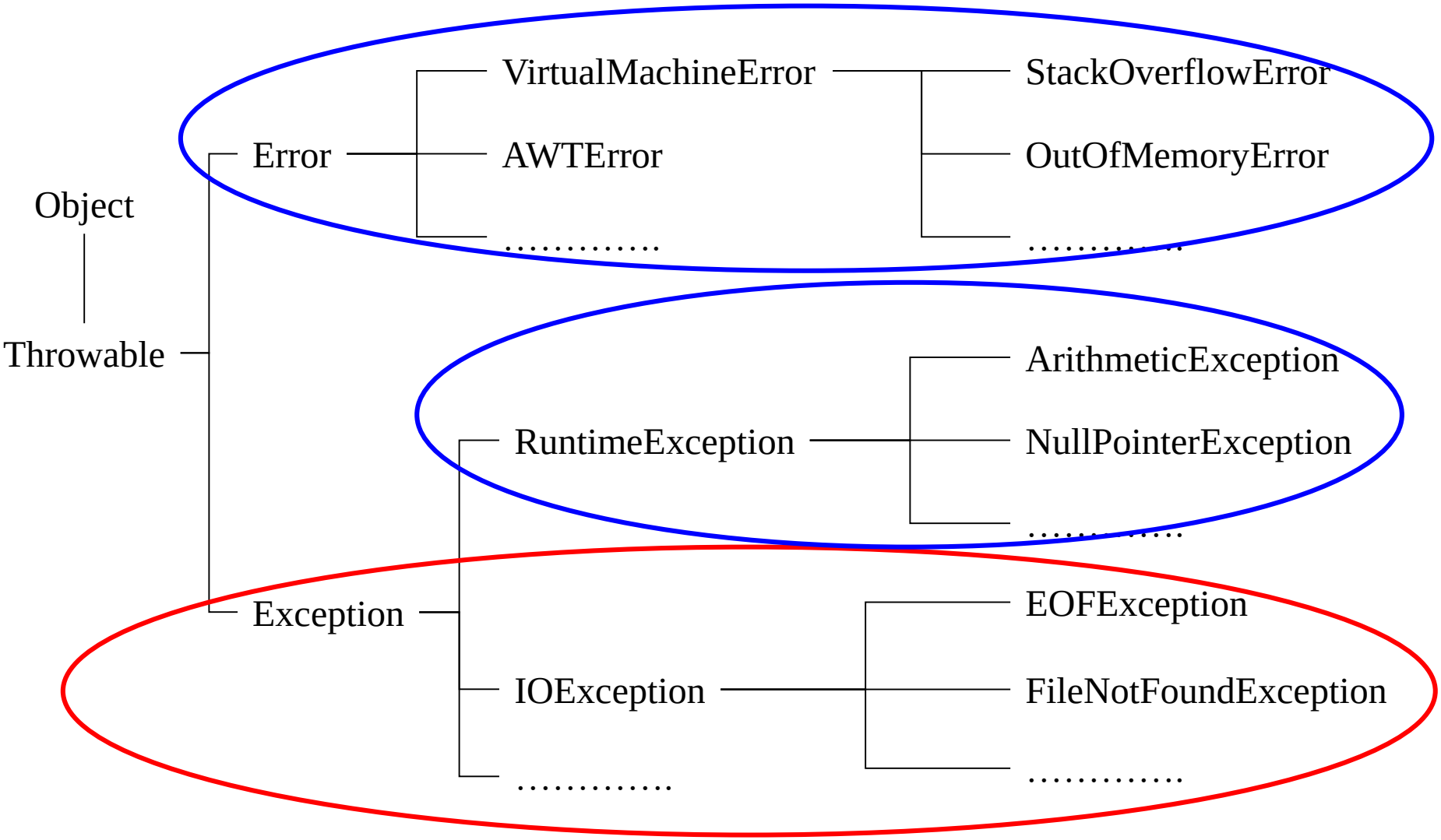
例外分類

- 例外類別
  - java.lang.Throwable的子類別
    - 錯誤：java.lang.Error
    - 例外：java.lang.Exception
  - 可應用Java例外處理機制來捕捉

例外分類

- 例外分類
  - 不需檢查的例外 (Unchecked Exception)
    - java.lang.Error
    - java.lang.RuntimeException
  - 必需檢查的例外 (Checked Exception)
    - java.lang.Exception

例外分類


Object
Throwable
Error
VirtualMachineError
AWTError
……
StackOverflowError
OutOfMemoryError
……
Exception
RuntimeException
ArithmeticException
NullPointerException
……
IOException
EOFException
FileNotFoundException
……
……

例外分類

- 不需檢查的例外 Unchecked Exception
  - Error
    - 難以修正回復的例外
    - 不需以例外處理機制保護
      - 記憶體不足→OutOfMemoryError
      - 執行緒死結→ThreadDeath

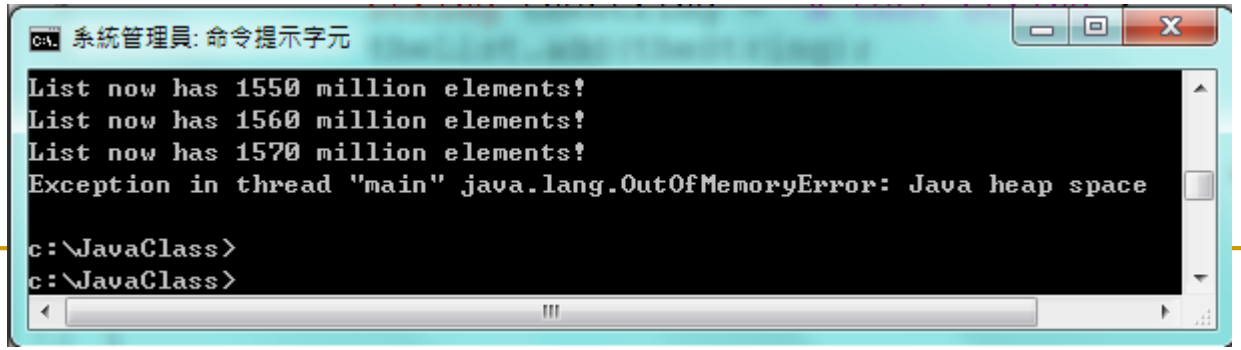
例外分類

- 不需檢查的例外 Unchecked Exception
  - RuntimeException
    - 程式設計上疏忽造成
    - 此類例外雖可用例外處理機制保護, 但不建議這麼做
    - 應該修正程式中的Bug

Error範例

```
public class ThrowError {
    public static void main (String args[]) {
        java.util.ArrayList theList = new java.util.ArrayList();
        while(true) {
            String theString = "A test String";
            theList.add(theString);

            if (theList.size()% 1000000 == 0) {
                System.out.println("List now has " + theList.size()/100000 + " million elements!");
            }
        }
    }
}
```



常見執行期例外 RuntimeException

執行期例外	說明
ArithmeticException	數學運算時的例外。例如：某數除以0。
ArrayIndexOutOfBoundsException	陣列索引值超出範圍。
NegativeArraySizeException	陣列的大小為負數。
NullPointerException	物件參照為null，並使用物件成員時所產生的例外。
NumberFormatException	數值格式不符所產生的例外。

需檢查的例外 (Checked Exception)

- Checked exception
  - Exception的子類別, 但不是RuntimeException的子類別
  - 可預期的外部因素造成的例外
    - 檔案不存在錯誤→FileNotFoundException
    - 輸出入處理錯誤→IOException
    - 資料庫處理錯誤→SQLException
    - 網路連結錯誤→SocketException
  - 系統強制程式中一定要作例外處理,否則編譯失敗

Java API常見 Checked Exception

Constructor Detail

File

```
public File(String pathname)
```

Creates a new `File` instance by converting the given pathname string into an abstract pathname. If the given string is the empty string, then the result is the empty abstract pathname.

Parameters:

`pathname` - A pathname string

Throws:

`NullPointerException` - If the `pathname` argument is `null`

Method Detail

createNewFile

```
public boolean createNewFile()
                        throws IOException
```

Atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist. The check for the existence of the file and the creation of the file if it does not exist are a single operation that is atomic with respect to all other filesystem activities that might affect the file.

Note: this method should *not* be used for file-locking, as the resulting protocol cannot be made to work reliably. The `FileLock` facility should be used instead.

Returns:

`true` if the named file does not exist and was successfully created; `false` if the named file already exists

Throws:

`IOException` - If an I/O error occurred

`SecurityException` - If a security manager exists and its `SecurityManager.checkWrite(java.lang.String)` method denies write access to the file

Since:

1.2

Java API常見 Checked Exception

Constructor Detail

FileReader

```
public FileReader(String fileName)
           throws FileNotFoundException
```

Creates a new `FileReader`, given the name of the file to read from.

Parameters:

`fileName` - the name of the file to read from

Throws:

`FileNotFoundException` - if the named file does not exist, is a directory rather than a regular file, or for some other reason cannot be opened for reading.

Method Detail

read

```
public int read()
         throws IOException
```

Reads a single character. This method will block until a character is available, an I/O error occurs, or the end of the stream is reached.

Subclasses that intend to support efficient single-character input should override this method.

Returns:

The character read, as an integer in the range 0 to 65535 (`0x00-0xffff`), or -1 if the end of the stream has been reached

Throws:

`IOException` - If an I/O error occurs

Checked Exception 範例

```
public class CheckedException1 {
        public static void main(String[] args) {
                java.io.File testFile = new java.io.File(“test.txt");
                testFile.createNewFile();

                System.out.println("File exists: " + testFile.exists());
                testFile.delete();
                System.out.println("File exists: " + testFile.exists());
        }
}
```

系統管理員: 命令提示字元

```
c:\JavaClass>javac CheckedException1.java
CheckedException1.java:4: error: unreported exception IOException; must be caugh
t or declared to be thrown
        testFile.createNewFile();
                                ^
1 error

c:\JavaClass>
```

課程大綱

1) 例外機制
2) **例外處理機制**
   - **捕捉例外**
   - 例外傳遞
3) 例外處理進階

例外處理機制

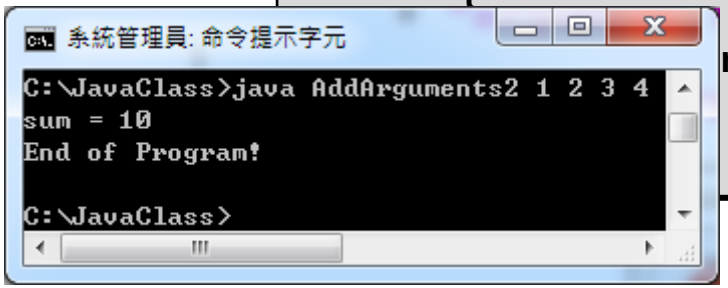
- 例外處理機制
  - 定義程式執行時，發生例外狀況時應如何處理
    - Ex: 網路連線失敗、欲開啟檔案不存在、傳入參數值錯誤
  - 確保系統在例外狀況發生時，仍能運行不會中斷
- 例外處理方法
  - 捕捉例外
  - 宣告丟出+例外傳遞

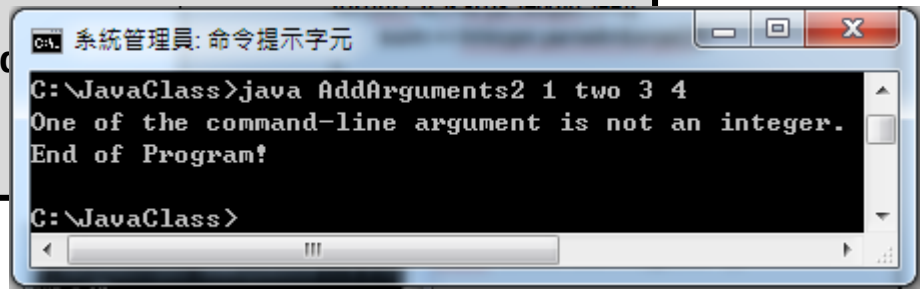
捕捉例外 try-catch 敘述

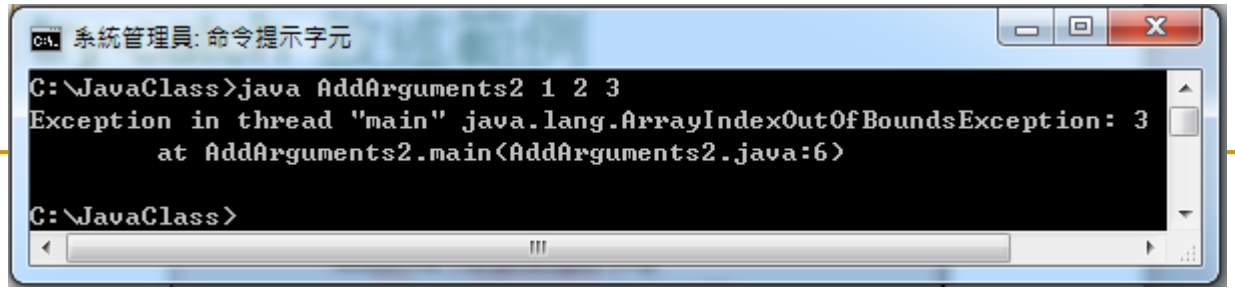
```
try {
    // 保護區塊
    .......
    .......
} catch (ExceptionType e) {
    // 錯誤處理
    ......
    ......
}
.......
.......
```

try-catch 敘述範例

```
public class AddArguments2{
    public static void main(String[] args) {
        try {
            int sum = 0;
            for(int i=0; i<4; i++){
                sum += Integer.parseInt(args[i]);
            }
            System.out.println("sum = " + sum);
        } catch (NumberFormatException nfe) {
            System.err.println("One of the command-line "
                    + "argument is not an integer.");
        }
```

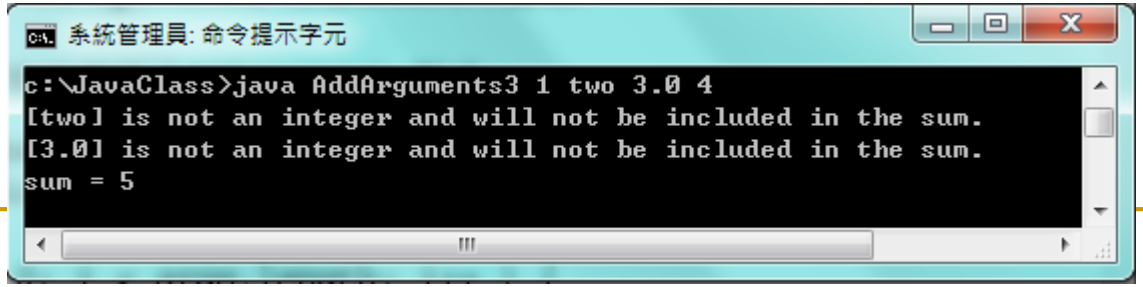






try-catch 敘述範例

```
public class AddArguments3{
    public static void main(String[] args) {
        int sum = 0;
        for(int i=0; i<4; i++){
            try {
                sum += Integer.parseInt(args[i]);
            } catch (NumberFormatException nfe) {
                System.err.println("["+args[i]+"] is not an integer"
                        + " and will not be included in the sum.");
            }
        }
        System.out.println("sum = " + sum);
    }
}
```



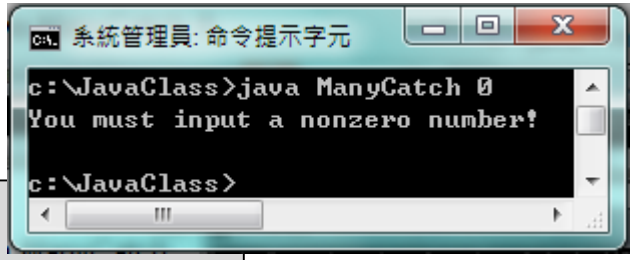
捕捉例外 try-catch 敘述

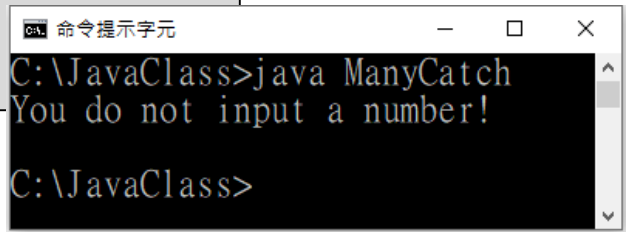
```
try {
    // 保護區塊
} catch (Specialize_Exc e) {
    // 錯誤處理
} catch (Normalize_Exc e) {
    // 錯誤處理
} finally {
    // 一定要執行的動作
}
```

可以有一個以上的 catch blocks
但是要注意 catch 的順序
例外也符合自動轉型
特定性的例外在前
一般性的例外在後

多重 catch 區段

```
public class ManyCatch {
    public static void main(String argv[]) {
        try {
            int i = Integer.parseInt(argv[0]);
            int ans = 10 / i;
        } catch (ArithmeticException ae) {
            System.err.println("You must input a nonzero number!");
        } catch (NumberFormatException ne) {
            System.err.println("You must input a integer number!");
        } catch (ArrayIndexOutOfBoundsException ae) {
            System.err.println("You do not input a number!");
        }
    }
}
```





多重 catch 區段

```
C:\JavaClass\Examples\Ch15>java ManyCatch1 0
You must input a nonzero number!
End of program!
```

```
public class ManyCatch1 {
    public static void main(String argv[]) {
        try {
                int i = Integer.parseInt(argv[0]);
            int ans = 10 / i;
        } catch (ArithmeticException ae) {
            System.err.println("You must input a nonzero number!");
        } catch (RuntimeException re) {
            System.err.println("RuntimeException: "+re);
        }
        System.out.println( "End of Program! " );
    }
}
```

```
C:\JavaClass\Examples\Ch15>java ManyCatch1
RuntimeException:java.lang.ArrayIndexOutOfBoundsException: 0
End of program!
```

```
C:\JavaClass\Examples\Ch15>java ManyCatch1 two
RuntimeException:java.lang.NumberFormatException: For input string: "two"
End of program!
```

catch 區段順序

```
public class ManyCatch2 {
    public static void main(String argv[]) {
        try {
            int i = Integer.parseInt(argv[0]);
            int ans = 10 / i;
        } catch (RuntimeException re) {
            System.err.println("RuntimeException: "+re);
        } catch (ArithmeticException ae) {
            System.err.println("You must input a nonzero number!");
        } catch (NumberFormatException ne) {
            System.err.println("You must input a integer number!");
        }
    }
}
```

系統管理員: 命令提示字元

```
c:\JavaClass>javac ManyCatch2.java
ManyCatch2.java:8: error: exception ArithmeticException has already been caught
                } catch (ArithmeticException ae) {
                  ^
ManyCatch2.java:10: error: exception NumberFormatException has already been caug
ht
                } catch (NumberFormatException ne) {
                  ^
2 errors

c:\JavaClass>
```

以父類別捕捉多組例外

- 多個例外的處理邏輯相同
  - 例外之間有繼承關係 vs. 例外之間沒有繼承關係

```
import java.io.*;
public class MultipleCatchExample {
    public static void main(String[] args) {
        try {
            ObjectInputStream in=
                new ObjectInputStream(
                new FileInputStream("cart.txt"));
            Object obj = in.readObject();
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }


    }
}
```

```
import java.io.*;
public class MultipleCatchExample {
    public static void main(String[] args) {
        ShoppingCart cart;
        try {ObjectInputStream in=
                new ObjectInputStream(
                new FileInputStream("cart.txt"));
            cart = (ShoppingCart) in.readObject();
        } catch (Exception e) {

        System.out.println(e.getMessage());
        }


    }
}
```

捕捉多組例外

- 捕捉多組例外
  - Java SE 7 新增
  - 多個例外的處理邏輯相同
  - 使用 “|” 串聯
    - 例外間不可有繼承關係
  - 減少重複內容的catch()

```
import java.io.*;
public class MultipleCatchExample {
    public static void main(String[] args) {
        ShoppingCart cart;
        try (ObjectInputStream in=
                new ObjectInputStream(
                new FileInputStream(“cart.txt") )){
            cart = (ShoppingCart) in.readObject();
        } catch (ClassNotFoundException |
                IOException e) {
            System.out.println(e.getMessage());
        }

    }
}
```

捕捉例外 try-catch 敘述

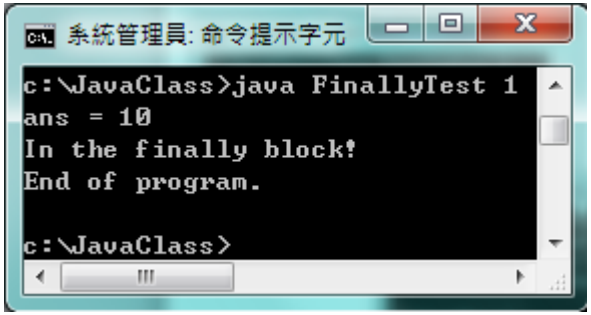
```
try {
    // 保護區塊
    .......
} catch (ExceptionType e) {
    // 錯誤處理
    ......
} finally {
    // 一定要執行的動作
    ......
}
.......
```

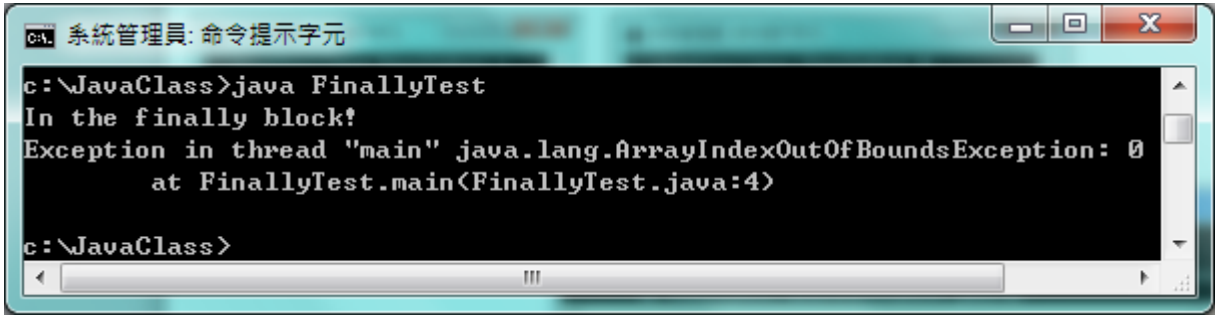
不論是否有例外產生都
一定會執行的區塊
除非在保護區快中遇到
System.exit() 方法

finally 區段

```
public class FinallyTest {
    public static void main(String argv[]) {
        try {
            int i = Integer.parseInt(argv[0]);
            int ans = 10 / i;
            System.out.println("ans = " + ans);
        } catch (ArithmeticException e) {
            System.err.println(e);
        } finally {
            System.out.println("In the finally block!");
        }

        System.out.println("End of program.");
    }
}
```

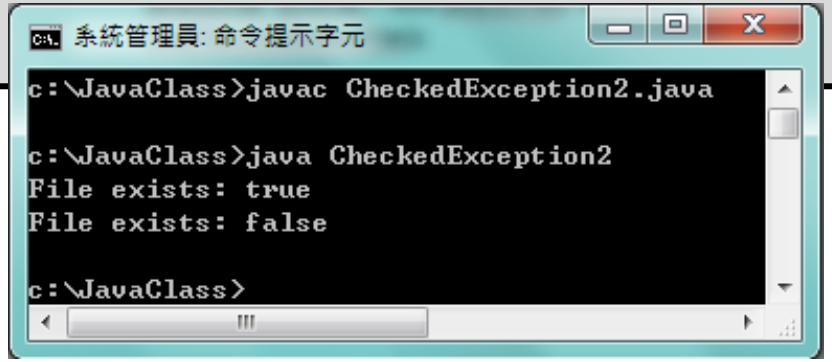




Checked Exception 範例

```
public class CheckedException2 {
    public static void main(String[] args) {
        try{
            java.io.File testFile = new java.io.File("test.txt");
            testFile.createNewFile();

            System.out.println("File exists: " + testFile.exists());
            testFile.delete();
            System.out.println("File exists: " + testFile.exists());
        } catch (java.io.IOException ioe){
            System.err.println(ioe);
        }
    }
}
```



處理例外相關方法

- java.lang.Throwable

  public String getMessage()

  public String getLocalizedMessage()

  public String toString()

  public void printStackTrace()

處理例外相關方法

```
public class ThrowableTest {
	public static void main(String argv[]) {
		try {
			java.io.FileReader f = new java.io.FileReader("test.txt");
		} catch (FileNotFoundException e) {
			System.out.println("=== getLocalizedMessage() ===");
			System.err.println(e.getLocalizedMessage());

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

			System.out.println("=== toString() ===");
			System.err.println(e);

			System.out.println("=== printStackTrace() ===");
			e.printStackTrace();
		}
	}
}
```

命令提示字元

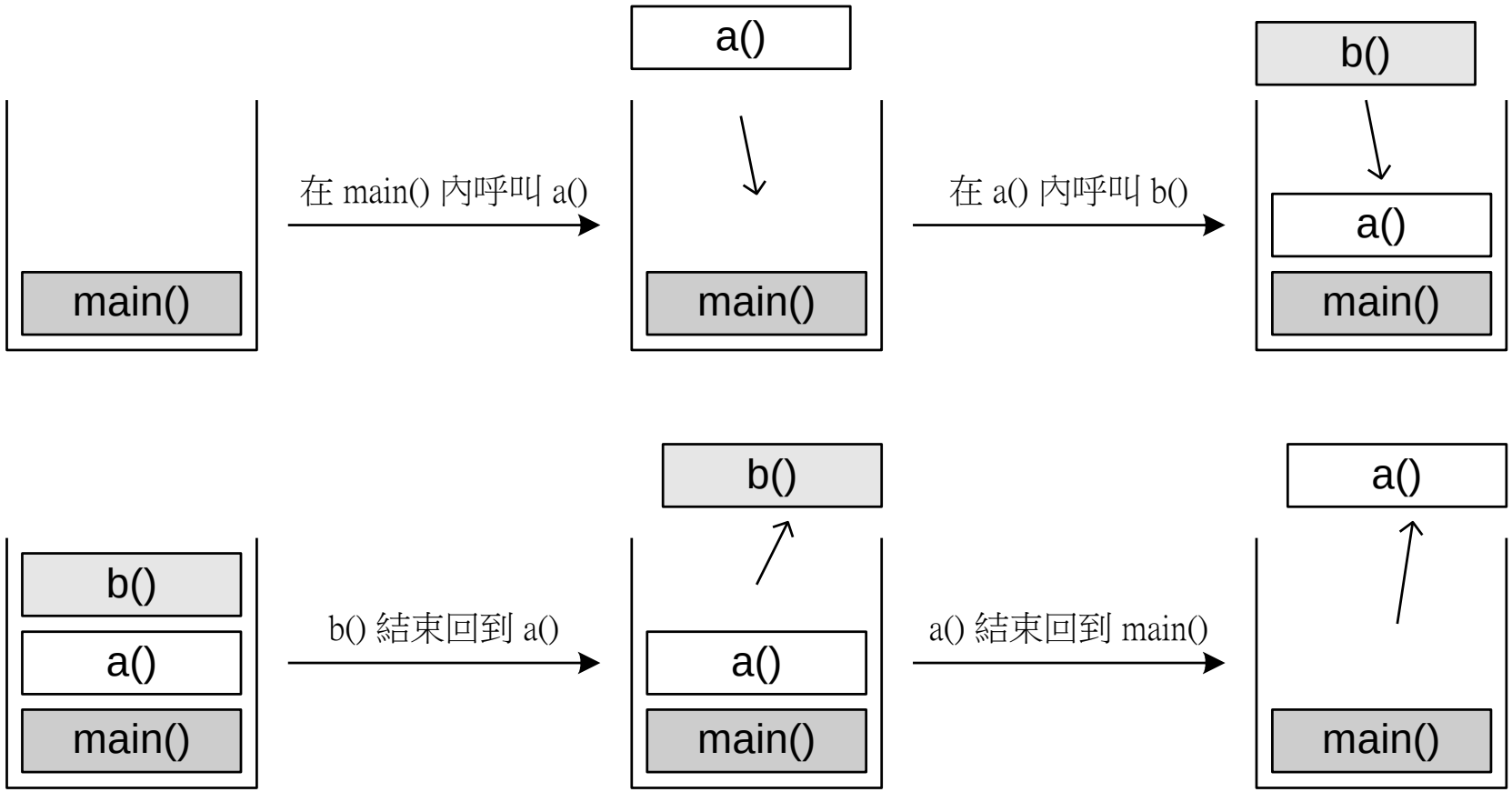
```
c:\JavaClass>java ThrowableTest
=== getLocalizedMessage() ===
test.txt (系統找不到指定的檔案。)
=== getMessage() ===
test.txt (系統找不到指定的檔案。)
=== toString() ===
java.io.FileNotFoundException: test.txt (系統找不到指定的檔案。)
=== printStackTrace() ===
java.io.FileNotFoundException: test.txt (系統找不到指定的檔案。)
        at java.io.FileInputStream.open0(Native Method)
        at java.io.FileInputStream.open(FileInputStream.java:195)
        at java.io.FileInputStream.<init>(FileInputStream.java:138)
        at java.io.FileInputStream.<init>(FileInputStream.java:93)
        at ThrowableTest.main(ThrowableTest.java:6)

c:\JavaClass>
```

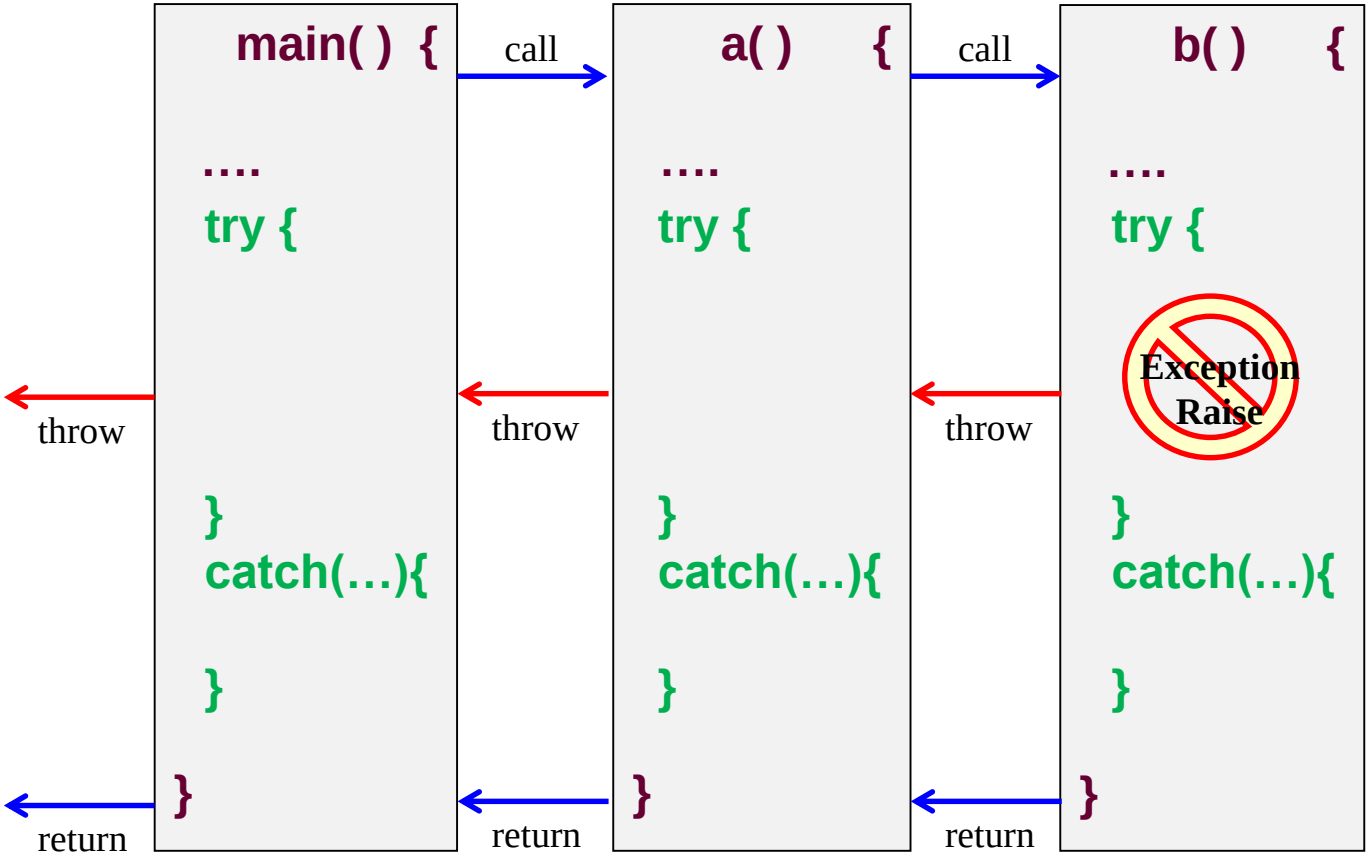
課程大綱

1) 例外機制
2) **例外處理機制**
   - 捕捉例外
   - **例外傳遞**
3) 例外處理進階

方法呼叫堆疊


main()
在 main() 內呼叫 a()
a()
main()
在 a() 內呼叫 b()
b()
a()
main()
b()
a()
main()
b() 結束回到 a()
b()
a()
main()
a() 結束回到 main()
a()
main()

堆疊與例外處理


main( ) {
....
try {
}
catch(…){
}
}
call
a( ) {
....
try {
}
catch(…){
}
}
call
b( ) {
....
try {
Exception
Raise
}
catch(…){
}
}
throw
throw
throw
return
return
return

```
public class TestCheckedCallStack {
    public static void main(String[] args) {
        CheckedCallStack cs = new CheckedCallStack();
        cs.a();
    }
}
```

```
public class CheckedCallStack {
    public void a(){
        b();
    }

    public void b() {
        try{
            java.io.File testFile = new java.io.File("test.txt");
            testFile.createNewFile();

            System.out.println("File exists: " + testFile.exists());
            testFile.delete();
            System.out.println("File exists: " + testFile.exists());
        } catch (java.io.IOException ioe){
            System.err.println(ioe);
        }
    }
}
```

Exception

throws

- 當例外發生時,可以不立即處理,而將例外丟給呼叫的方法處理
  - 被呼叫的方法內不作try/catch
  - 在方法宣告之後加上throws宣告
  - 宣告可能丟出CheckedException：系統強制方法呼叫者一定要作例外處理,否則編譯失敗
  - 宣告可能丟出UncheckedException,則無強制機制

- 例外處理 Handle or Declare Rule
  - try-catch-finally敘述處理
  - throws宣告丟出。

throws

- 語法

```
public void methodA(String str) throws 例外類別,… {
        //方法敘述
}
```

  - throws之後可以接多個例外型別，表示方法執行過程中可能丟出屬於這些例外型別的物件。

```
public class TestCheckedCallStack2 {
    public static void main(String[] args) {
        CheckedCallStack2 cs = new CheckedCallStack2();
        cs.a();
    }
}
```

```
public class CheckedCallStack2 {
    public void a(){
        try{
            b();
        } catch (java.io.IOException ioe){
            System.err.println(ioe);
        }
    }

    public void b() throws java.io.IOException{
        java.io.File testFile = new java.io.File("test.txt");
        testFile.createNewFile();

        System.out.println("File exists: " + testFile.exists());
        testFile.delete();
        System.out.println("File exists: " + testFile.exists());
    }
}
```

Exception

```
public class TestCheckedCallStack3 {
    public static void main(String[] args) {
        CheckedCallStack3 cs = new CheckedCallStack3();
        try{
            cs.a();
        } catch (java.io.IOException ioe){
            System.err.println(ioe);
        }
    }
}
```

```
public class CheckedCallStack3 {
    public void a() throws java.io.IOException{
        b();
    }

    public void b() throws java.io.IOException{
        java.io.File testFile = new java.io.File("test.txt");
Exception   testFile.createNewFile();

        System.out.println("File exists: " + testFile.exists());
        testFile.delete();
        System.out.println("File exists: " + testFile.exists());
    }
}
```

throw

- 程式內部自行產生要丟出的例外物件
  - throw 所丟出的例外物件同樣可以使用try-catch敘述處理。

- 語法

```
throw new ExceptionType();

throw new ExceptionType("錯誤訊息");
```

throw vs. throws

```
public void method() throws XXXException {
        throw new XXXException();
}
```

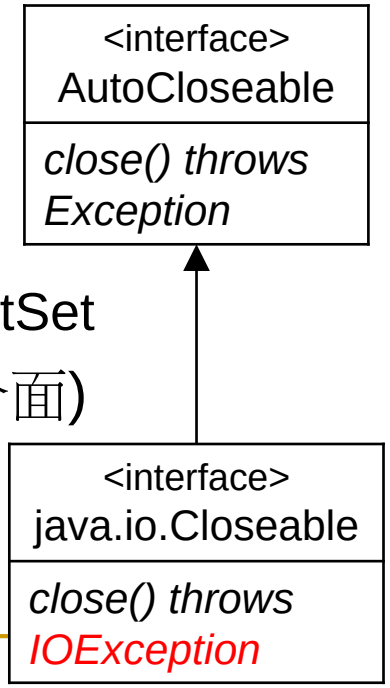
- Checked vs. Unchecked Exception
  - 方法內丟出(thorw) CheckedException,必須在方法簽章宣告throws此例外型別
  - 方法內丟出(thorw) Unchecked Exception (Error/RuntimeException),可以不在方法簽章宣告throws此例外型別

課程大綱

1) 例外機制
2) 例外處理機制
3) **例外處理進階**
   - **資源自動關閉**
   - 自訂例外
   - 例外與方法覆寫

資源自動關閉 try with Resource

- 資源自動關閉
  - Java SE 7 提供
  - 確保資源物件在try-catch區段執行完畢後,自動關閉
  - 資源於try命令後的括號中()中開啟
  - 資源物件需實作
    - java.lang.AutoCloseable 介面
      - java.sql套件中Connection、Statement、ResultSet
    - java.io.Closeable 介面 (AutoCloseable的子介面)
      - java.io套件中 FileReader、FileWriter、FileInputStream、FileOutputStream



資源自動關閉

```
import java.io.*;
public class FinallyExampleMain {
    public static void main(String[] args) {

        try (InputStream in=
                new FileInputStream("missingfile.txt") ){
            System.out.println("File open");
            int data = in.read();
        } catch (FileNotFoundException e) {
            System.out.println(e.getMessage());
        } catch (IOException e) {
            System.out.println(e.getMessage());
        } finally {
            try {
                if (in != null) {    in.close();    }
            } catch (IOException e) {
                System.out.println("Failed to close file");
            }
        }
    }
}
```

在 try with resource中

- catch 與 finally區段都是optional
- catch及finally區段,將在宣告的資源關閉後執行

多個資源自動關閉

```
import java.io.*;
import java.util.*;
public class MultipleResource {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("輸入資料來源檔:");
        String src = sc.next();
        System.out.print("輸入複製目標檔:");
        String dest = sc.next();
        try (BufferedReader in = new BufferedReader(new FileReader(src));
            PrintWriter out = new PrintWriter(new FileWriter(dest))) {
            String line;
            while((line = in.readLine()) != null) {
                out.println(line);
            }
        } catch(IOException e){
            System.out.println(e.getMessage());
        }// No need to close resources in a "finally"
    }
}
```

資源關閉的順序與建構順序相反

Suppressed Exception例外的壓制

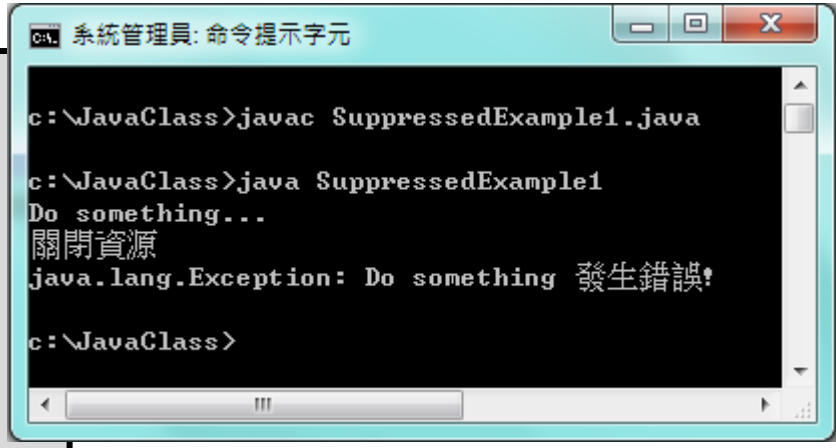
```
public class MyResource implements AutoCloseable{
    @Override
    public void close() throws Exception {
        System.out.println("關閉資源");
        throw new Exception("關閉資源發生錯誤!");
    }
}
```

Exception

```
public class SuppressedExample1 {
    public static void main(String[] args) {
        try {
            doSomething();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
    public static void doSomething() throws Exception {
        try (MyResource mr = new MyResource()) {
            System.out.println("Do something...");
            throw new Exception("Do something 發生錯誤!");
        }
    }
}
```

Exception

try 區塊裡丟出例外時,Java會先關閉資源,再處理catch區段
萬一關閉資源時,close()方法丟出另一個例外,catch區段捕捉到哪個例外？



catch 陳述式只能補捉一個例外!

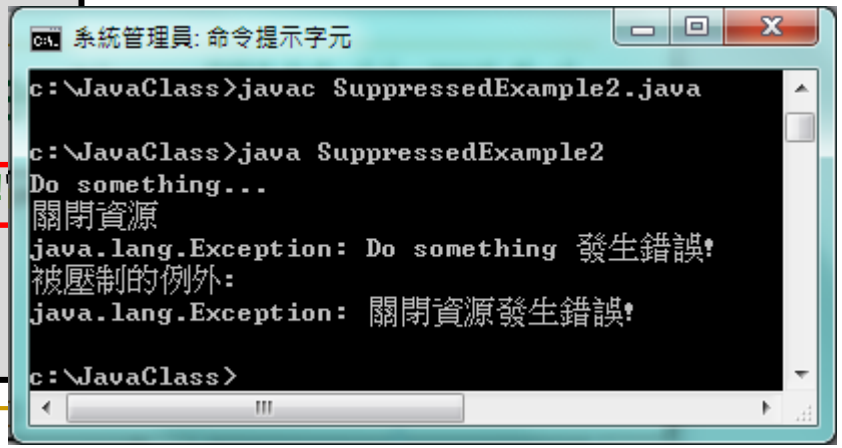
- **try 區塊的程式碼是主要的,丟出的例外，會被優先補捉**
- **close() 丟出來的例外是次要的，會被壓制(Suppressed)**

Suppressed Exception例外的壓制

```
public class SuppressedExample2 {
    public static void main(String[] args) {
        try {
            doSomething();
        } catch (Exception e) {
            System.out.println(e);
            System.out.println("被壓制的例外: ");
            Throwable[] th = e.getSuppressed();
            for(Throwable t : th)
                System.out.println(t);
        }
    }
    public static void doSomething() throws Exception {
        try (MyResource mr = new MyResource()) {
            System.out.println("Do something...");
            throw new Exception("Do something 發生錯誤!
        }
    }
}
```

```
public class MyResource implements AutoCloseable{
    @Override
    public void close() throws Exception {
        System.out.println("關閉資源");
        throw new Exception("關閉資源發生錯誤!");
    }
}
```

取出所有被壓制的例外：
Throwable 介面的
getSuppressed():Throwable[ ]



自訂例外

- 使用者可自訂自己需要的例外類別
- 使用時機
  - 所需例外類別，類別函式庫沒有提供
  - 由某個程式所產生的例外要和其他程式所產生的例外作區分

定義自訂Exception類別

- 自訂Exception類別
  - 繼承某個原有Exception類別
  - 加上傳入字串參數的建構子
  - 加上無參數的建構子
  - 自行定義新增或覆寫的方法

包覆例外 Wrapper Exception

- 將捕捉到JVM丟出的例外,包覆成自訂例外
  - 增加以捕捉到的Throwable物件為參數的建構子
  - 增加一個字串及捕捉到的Throwable物件為傳入參數的建構子
  - Throwable類別的getCause()方法可取得此被包覆的例外

自訂Exception類別範例

```
public class DAOException extends Exception {

    public DAOException() {
        super();
    }

    public DAOException(String message) {
        super(message);
    }

    public DAOException(Throwable cause) {
        super(cause);
    }

    public DAOException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

```
try {
    //……
} catch (DAOException e) {
    Throwable t = e.getCause();
}
```

Wrapper Exception in DAO Pattern

```
public interface EmployeeDAO extends AutoCloseable {

    public void add(Employee emp) throws DAOException;

    public void update(Employee emp) throws DAOException;

    public void delete(int id) throws DAOException;

    public Employee findById(int id) throws DAOException;

    public Employee[] getAllEmployees() throws DAOException;

}
```

```
public Employee findById(int id) throws DAOException {
        try {
            return employeeArray[id];
        } catch (ArrayIndexOutOfBoundsException e) {
            throw new DAOException("Error finding employee in DAO", e);
        }
}
```

重拋例外 Java SE 6

```
public class FirstException extends Exception {
    // ...
}
```

```
public class SecondException extends Exception {
    // ...
}
```

```
public void rethrowException(String exceptionName) throws Exception {
    try {
        if (exceptionName.equals("First")) {
            throw new FirstException();
        } else {
            throw new SecondException();
        }
    } catch (Exception e) {
            throw e;
    }
}
```

重拋例外 Java SE 7

```
public class FirstException extends Exception {
    // ...
}
```

```
public class SecondException extends Exception {
    // ...
}
```

```
public void rethrowException(String exceptionName) throws  FirstException, SecondException {
    try {
        if (exceptionName.equals("First")) {
            throw new FirstException();
        } else {
            throw new SecondException();
        }
    } catch (Exception e) {
            throw e;
    }
}
```

例外與方法覆寫

- 方法名稱一定要一樣,否則是不同的方法
- 傳入參數列要一樣,否則是overloading
- 傳回值資料要一樣,否則會有compile error
- 修飾子權限不能變小
- 方法覆寫時,丟出例外的規則
  - 子類別覆寫的方法,不可丟出父類別方法無法丟出的Check Exception類別

Exception

↑

IOExcption

↑

EOFException

```
public class Parent {
    public void m1() throws IOException{
    }
}
```

```
public class Child1 extends Parent {
    public void ml() throws EOFException {
    }
}
```

```
public class Child2 extends Parent {
    public void ml() throws Exception {
    }
}
```

```
public class Test {
    public static void main(String [] args) {
        Parent p1 = new Child1();
        try {
            p1.m1();
        } catch (IOException e) {

        }
        Parent p2 = new Child2();
        try {
            p2.m1();
        } catch (IOException e) {

        }
    }
}
```