ITP4510 – Data Structures & Algorithms: Concepts and Implementation

Topic 2: Exception Handling (例外處理)

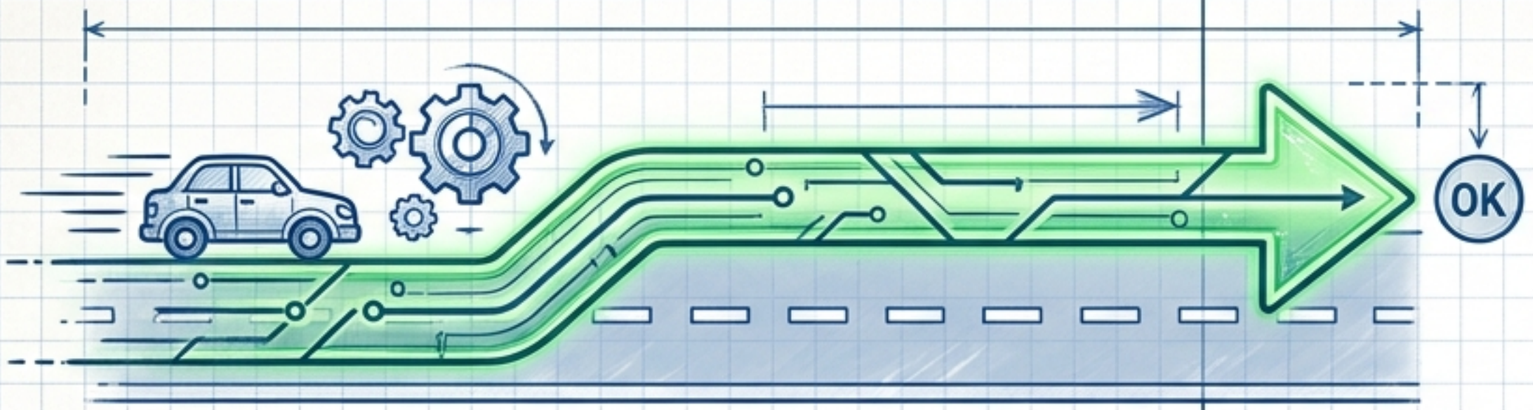
從零開始掌握 Java 例外處理機制

Mastering Java Exception Handling from Scratch

教授的私房筆記：為你的程式碼架設完美防護網

The Professor's Notes: Building the Perfect Safety Net for Your Code

Normal Flow (正常流程)

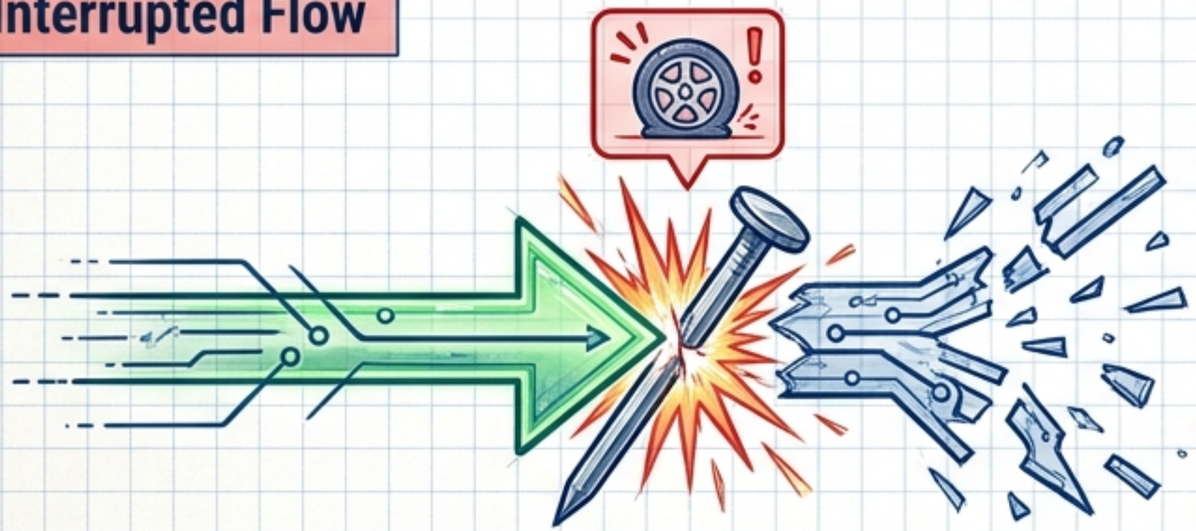


Normal Flow (正常流程)

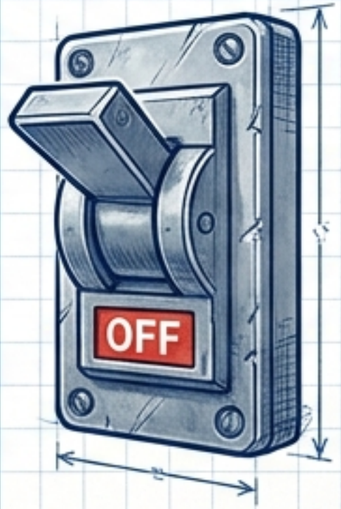
Feynman Analogy (費曼類比)

想像你在高速公路上開車，突然爆胎了！這個「爆胎」打斷了你原本的旅程，這在 Java 裡就是一個 Exception。

Interrupted Flow



Exception (例外)



Definition (定義)	Common Causes (考試常客)	Goal (最終目標)
An event during execution that interrupts the normal flow of instructions. (程式執行期間發生，打斷正常執行流程的事件)	• Divide a number by 0 (除以零) • Array index out of bounds (陣列索引超界) • Null reference (空指標參考)	Notify user (通知使用者) ⇨ Terminate gracefully (優雅結束) ⇩ Recover (從錯誤中恢復)

Exception Handling: try-catch-finally

Zone 1 (Top)

Zone 2 (Middle)

Zone 3 (Bottom)

```
try {
    // 包含可能出錯的程式碼
}
```

BANG!

```
catch (Exception e) {
    // 負責處理特定錯誤
}
```

```
finally {
    // 無論如何都會執行
}
```

The Danger Zone (試錯區)

Encloses statements that may generate an exception.
一旦出錯，立刻停止並跳離！

The Safety Net (捕手區)

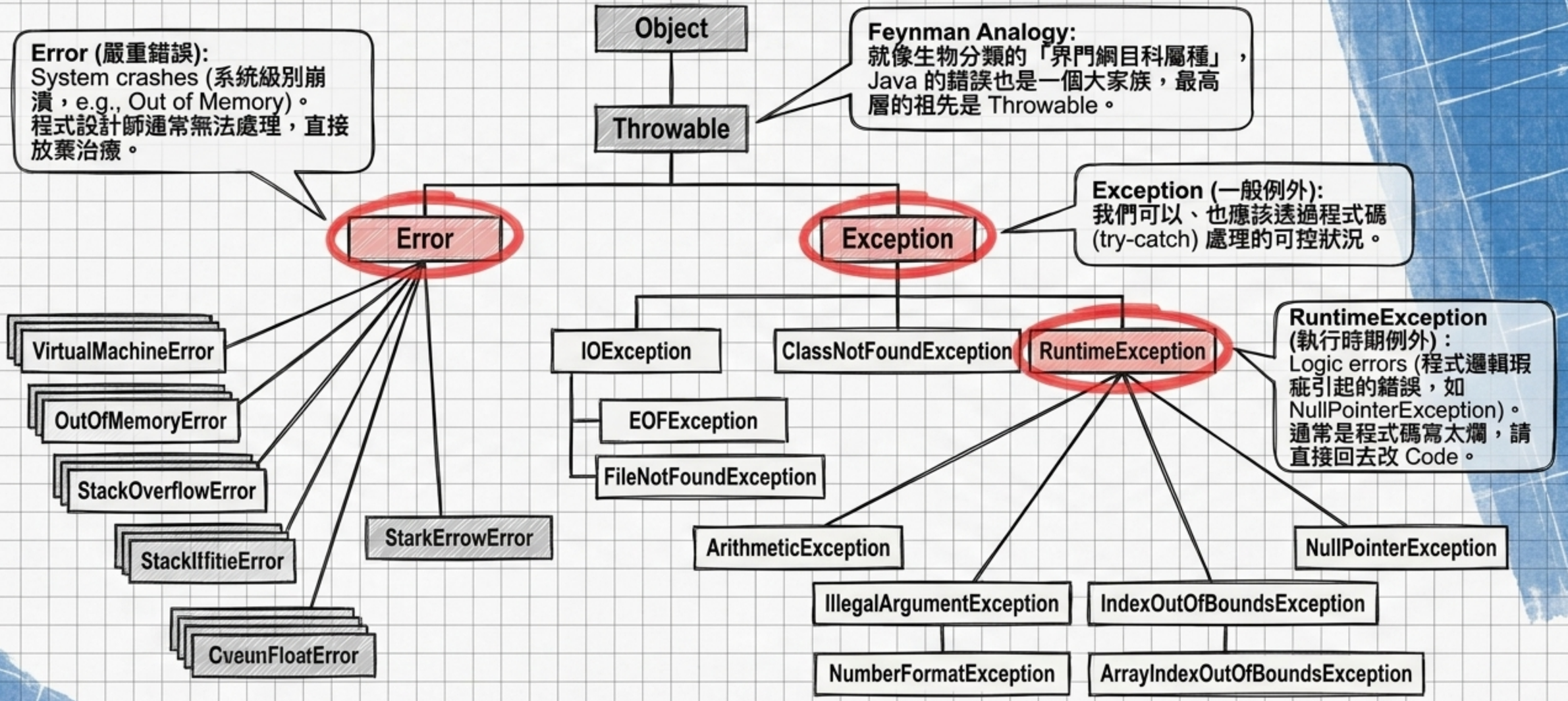
Contains the exception handler.
當例外發生，接住從 try 掉下來的錯誤。

The Cleanup Crew (善後區)

Always executes.
打掃戰場，釋放資源。

Feynman Analogy:
try 是你在走鋼索，一旦踩空掉下來，catch 就是接住你的安全網。

Java Exception Hierarchy



Checked Exceptions (受檢例外)

Family: All non-RuntimeException subclasses (非 RuntimeException 的其他 Exception)。

Compiler Rule: MUST be caught (用 try-catch) OR declared (用 throws). 否則編譯失敗 (Compile Error)!

Why?: 外在不可控因素 (例如硬碟壞了、網路斷了)，編譯器強迫你必須先做好最壞打算。

Examples: IOException, FileNotFoundException

Unchecked Exceptions (非受檢例外)

Family: Derive from RuntimeException (繼承自 RuntimeException 的家族)。

Compiler Rule: NOT required to be listed in throws or caught. 編譯器不會強迫你處理。

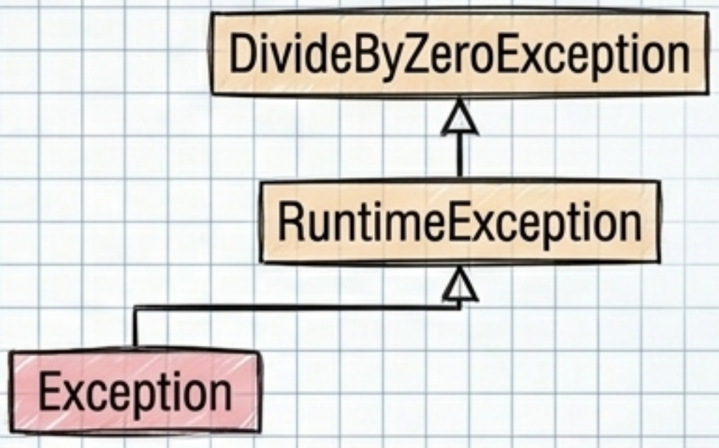
Why?: 這通常是程式邏輯錯誤 (Logic errors)，你應該直接修好 Bug，而不是寫 catch 掩蓋它。

Examples: ArrayIndexOutOfBoundsException, NullPointerException

Handling Multiple Exceptions

```
try {
    result = quotient(num, den);
} catch (DivideByZeroException e) {
    System.out.println("DivideByZeroException
    caught!");
} catch (Exception e) {
    System.out.println("Exception caught!");
}
```

Order Matters:
Specific FIRST, General LAST
(先抓子類別，最後抓父類別)



Exam Trap (危險陷阱)

```
catch (Exception e) { ... }
catch (DivideByZeroException e) { ... }
```

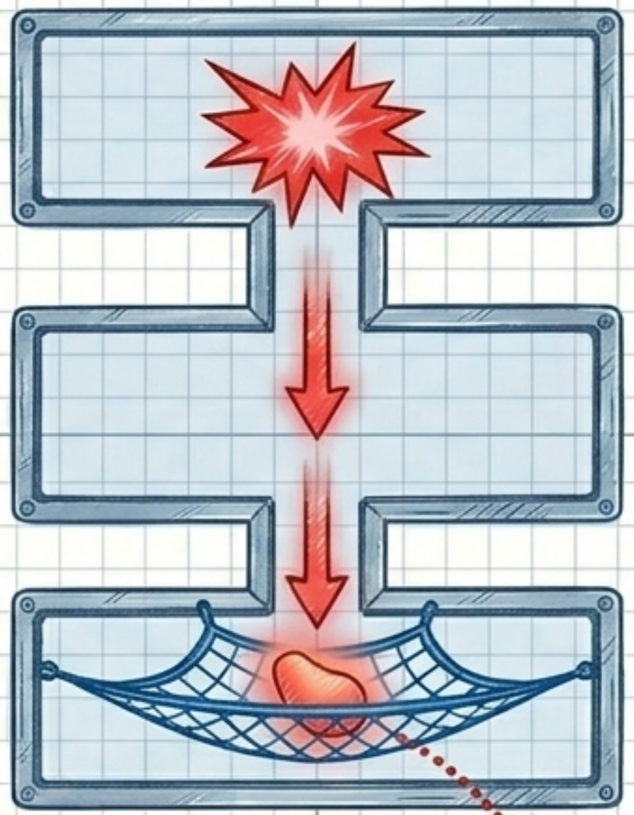
If you put Exception at the top, it acts like a black hole absorbing all errors!
Compiler Error: “exception has already been caught”.

Method Call Stack: The Hot Potato Game

Error sparks here!
No catch block.

Method B()

main()



Feynman Analogy: 踢皮球遊戲 (Passing the Hot Potato)。這層處理不了，就往上丟給呼叫者 (Caller)。

Falls to caller.
No catch block.

Safety Net catches it here!
Program survives.

Java Runtime / Program Crash

Ultimate Fate: 如果一路拋到 main 都沒人理，皮球掉到 Java runtime，程式直接當掉！

```
public static void processFile(String f)
        throws FileNotFoundException {
    ...
}
```

The Keyword

throws (注意有 ‘s’，名詞屬性，放在方法宣告的尾端)。

The Purpose

宣告方法可能發生的 Checked Exception。這是一張警告標籤，告訴呼叫者：「用我的方法可能會發生這些危險，你要準備好處理！」

The Chain Reaction

呼叫帶有 throws 的方法時，你必須 try-catch 它，或是繼續往上加 throws 踢皮球！(Must catch or declare to be thrown).

throws vs. throw: The Passive Warning vs. The Active Verb


```
public void process() throws IOException
```

throws (With ‘s’)

Passive warning label on a method signature. 警告標籤。


```
try {
   ...
} catch (Exception e) {
   throw e;
}
```

Rethrowing (再次抛出)

Rethrow!
Pass up the chain to the next caller.
(再次抛出！傳給上層呼叫者)


```
if (den == 0)
    throw new DivideByZeroException();
```

throw (No ‘s’)

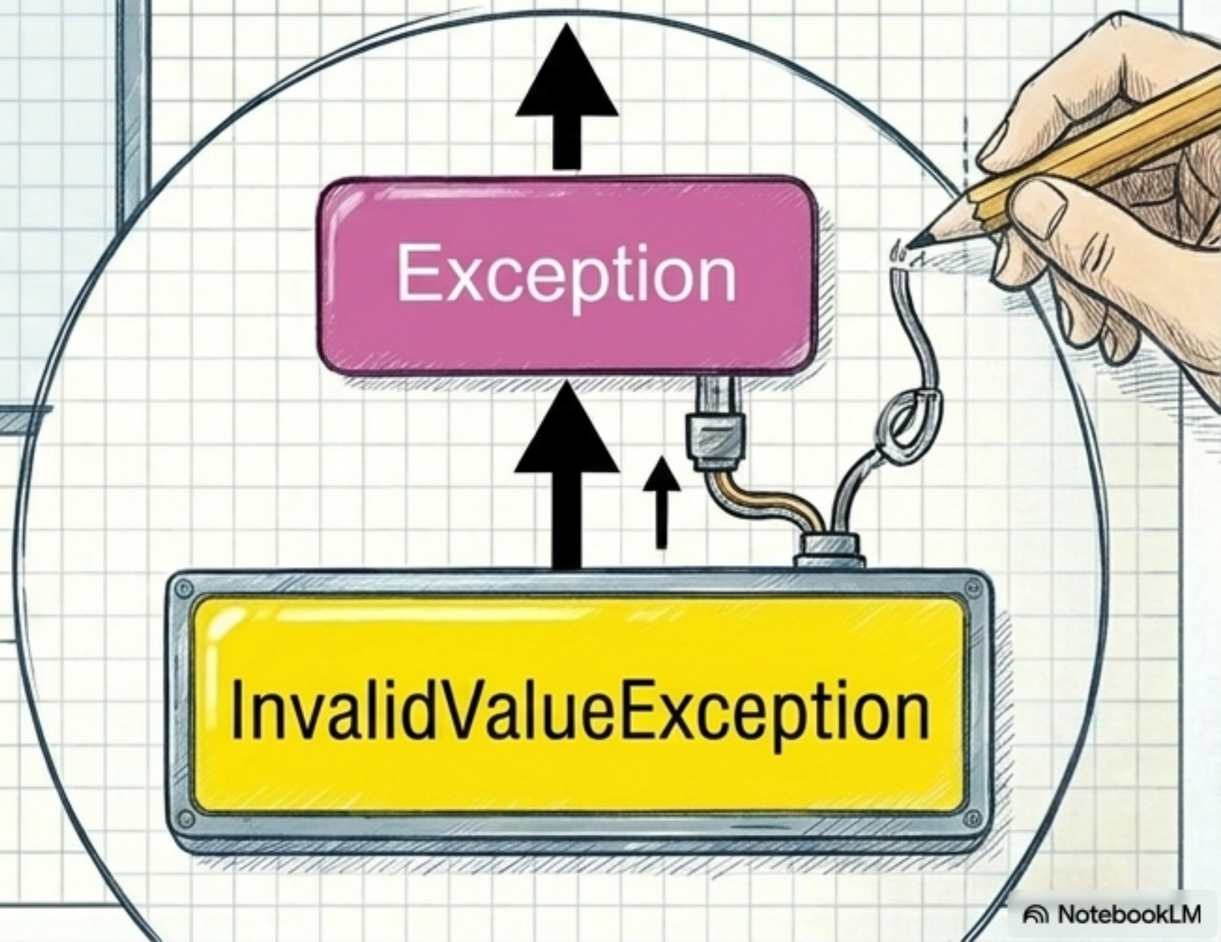
Active verb. 主動創造並引爆一個例外物件 (Explicitly trigger an exception).

Creating Your Own Custom Exceptions

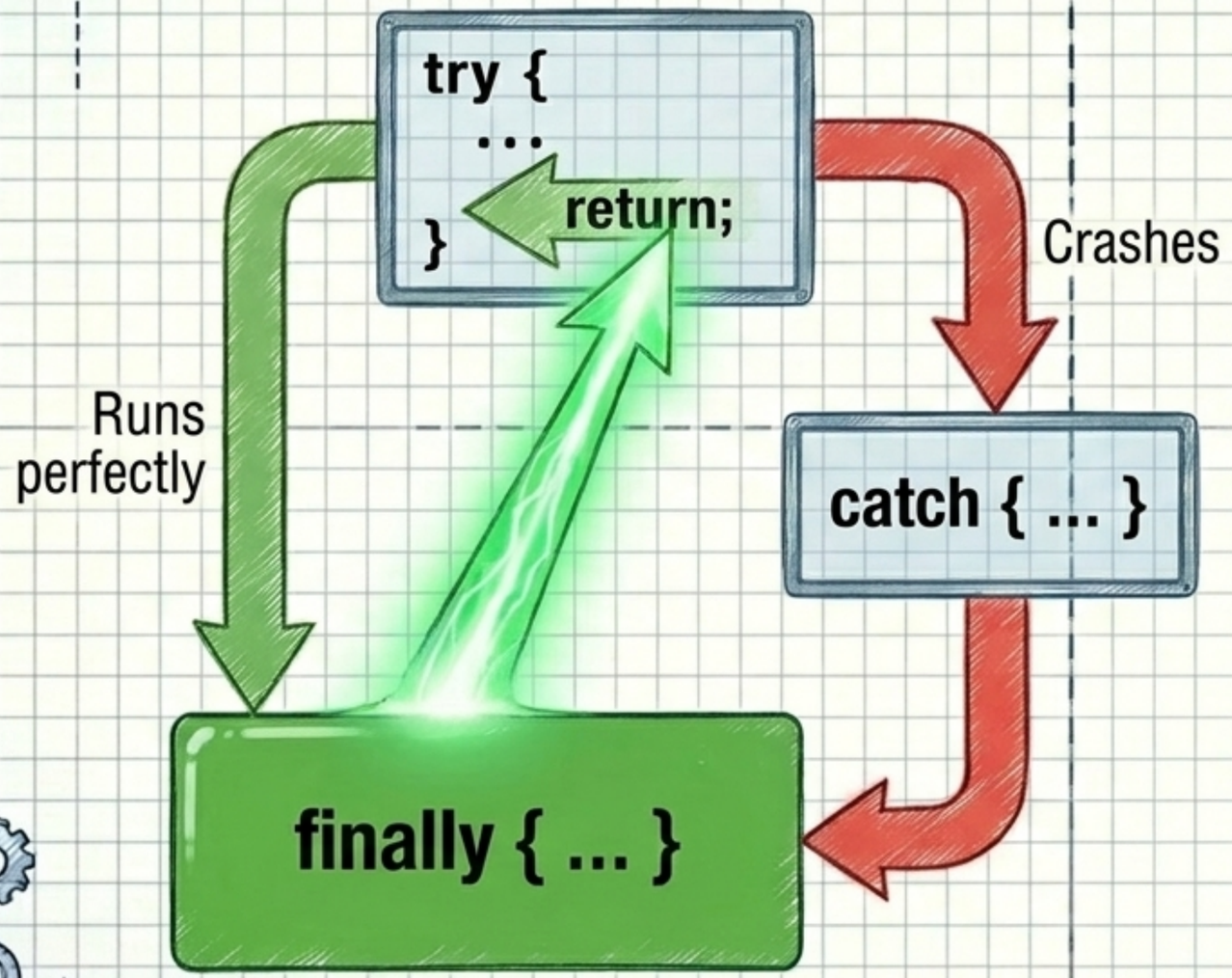
```
class InvalidValueException extends Exception {

    public InvalidValueException(double v) {
        super("Invalid radius: " + v);
    }

}
```

Inheritance Relationship (Is-A)。繼承現有的例外類別來創造專屬的錯誤類型。

Usually just call the parent constructor.



The Finally Block: The Absolute Execution Guarantee



The Absolute Rule: finally 提供無論如何都會執行的程式碼 (Always executes).

Feynman Analogy: 最盡責的清潔工。不管派對辦得很成功 (try) 還是搞砸了 (catch)，最後都得由清潔工負責打掃關燈，沒做完誰都不准走。

Exam Trap (必考陷阱) :
The **finally**-block is executed even if the method returns within **try** or **catch**!
(就算寫了 return，還是會被硬拉回來把 finally 執行完！)

Step-by-Step Execution Trace (實戰演練：程式碼追蹤)

```
public class TestException {
  public static void main(String[] args) {
    try {
      // Code Segment 2
      Circle c1 = new Circle(8);
      Circle c2 = new Circle(-5);
      c2.setRadius(-7);
    }
    catch (InvalidValueException ex) {
      System.out.println("Exception in main thrown!");
    }
    finally {
      System.out.println("Done!");
    }
    System.out.println("Number of objects created: " +
      Circle.getNumberOfObjects());
  }
} // end class TestException

class Circle {
  private double radius;
  private static int numberOfObjects = 0;

  public Circle(double newRadius) {
    try {
      setRadius(newRadius);
      System.out.println("Circle with radius " + newRadius
                         + " set!");
      numberOfObjects++;
  }
```

Step-by-Step Execution Trace (實戰演練：程式碼追蹤)

1. c1 = new Circle(8);
→ Normal. numberOfObjects = 1.

2. c2 = new Circle(-5);
Trap!
→ throw InvalidValueException in setRadius.
Caught *inside* Circle constructor!
Prints “Some exception!”.
numberOfObjects stays 1.
(Trap: main didn’t crash!)

3. c2.setRadius(-7);
→ Exception thrown directly to main!
Caught in main.
Prints “Exception in main thrown!”.

4. finally in main runs
→ Prints “Done!”.

5. End
→ Prints “Number of objects created: 1”.

實戰演練：Rethrow 與 Finally 追蹤
(Exam Application: Rethrow & Finally)

1. methodC(48, 0)
calls methodB
➔ 48/0 triggers ArithmeticException.

2. Catch in methodC
methodC catches it
➔ Prints “ari exp”.

3. Rethrow!
throw e;
➔ Rethrows exception back to main!

!!

Trap!
Always runs before leaving!

4. Rethrow!
必須先執行清潔工！
finally runs
➔ Prints “fin”.

!!

5. Reach main()
Exception reaches main
catch (Exception e) triggers
➔ Prints “ari or cus a”.

6. Main ends
main finally runs
➔ Prints “main fin”.

四大神器總結 (The 4 Pillars Summary)

try-catch-finally

處理例外的防護網
(Block for handling exceptions).

throw

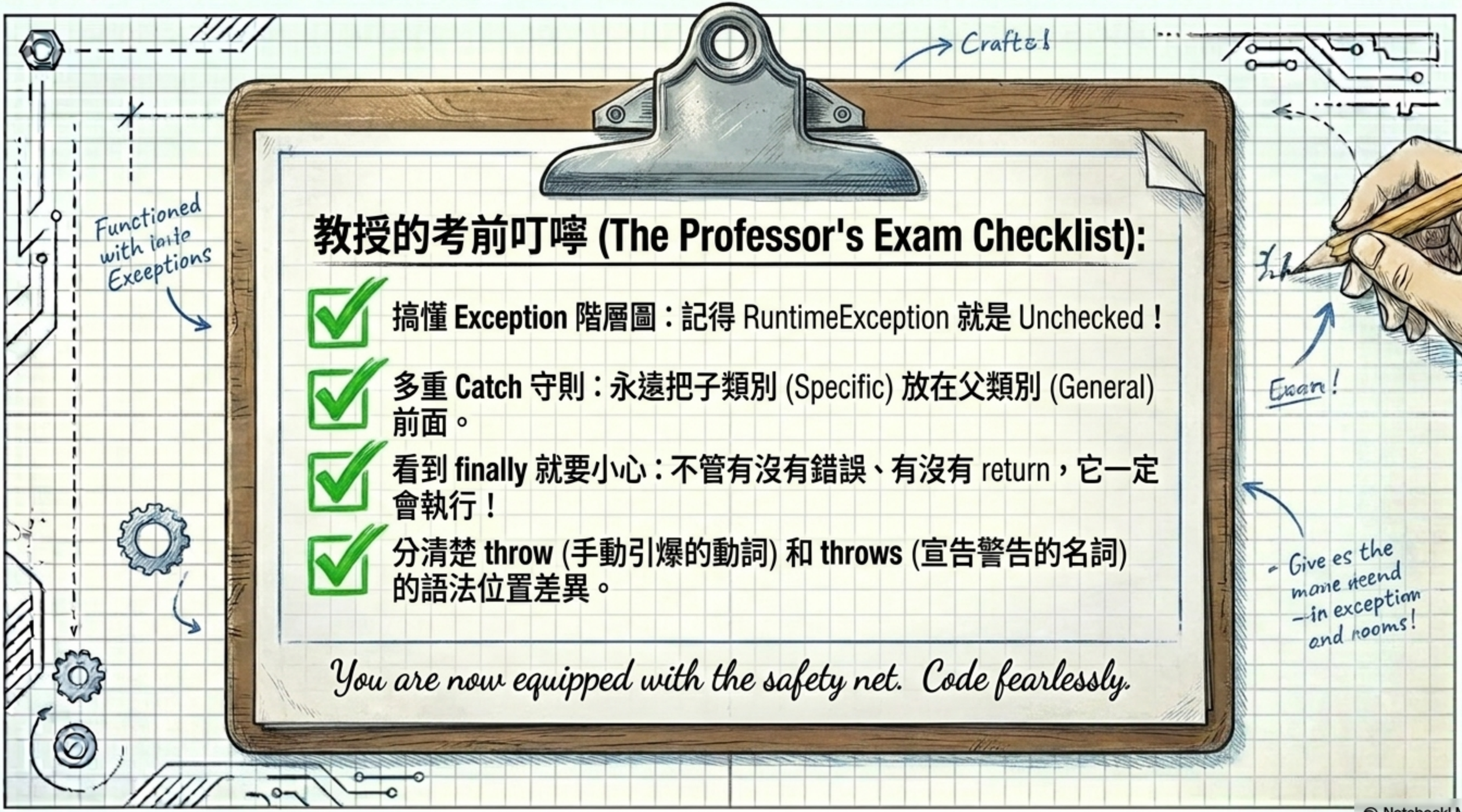
動詞：主動創造並拋出例外物件
(Action: explicitly throwing an object).

throws

名詞宣告：警告方法可能發生的危險
(Declaration: method signature warning).

Checked vs Unchecked

編譯器強迫處理 (IOException) vs 程式碼邏輯瑕疵 (RuntimeException).


Crafted
Functioned with late Exceptions
Exam!
Give es the mame neend —in exceptiom and rooms!
教授的考前叮嚀 (The Professor's Exam Checklist):
搞懂 Exception 階層圖：記得 RuntimeException 就是 Unchecked！
多重 Catch 守則：永遠把子類別 (Specific) 放在父類別 (General) 前面。
看到 finally 就要小心：不管有沒有錯誤、有沒有 return，它一定會執行！
分清楚 throw (手動引爆的動詞) 和 throws (宣告警告的名詞) 的語法位置差異。
You are now equipped with the safety net. Code fearlessly.