



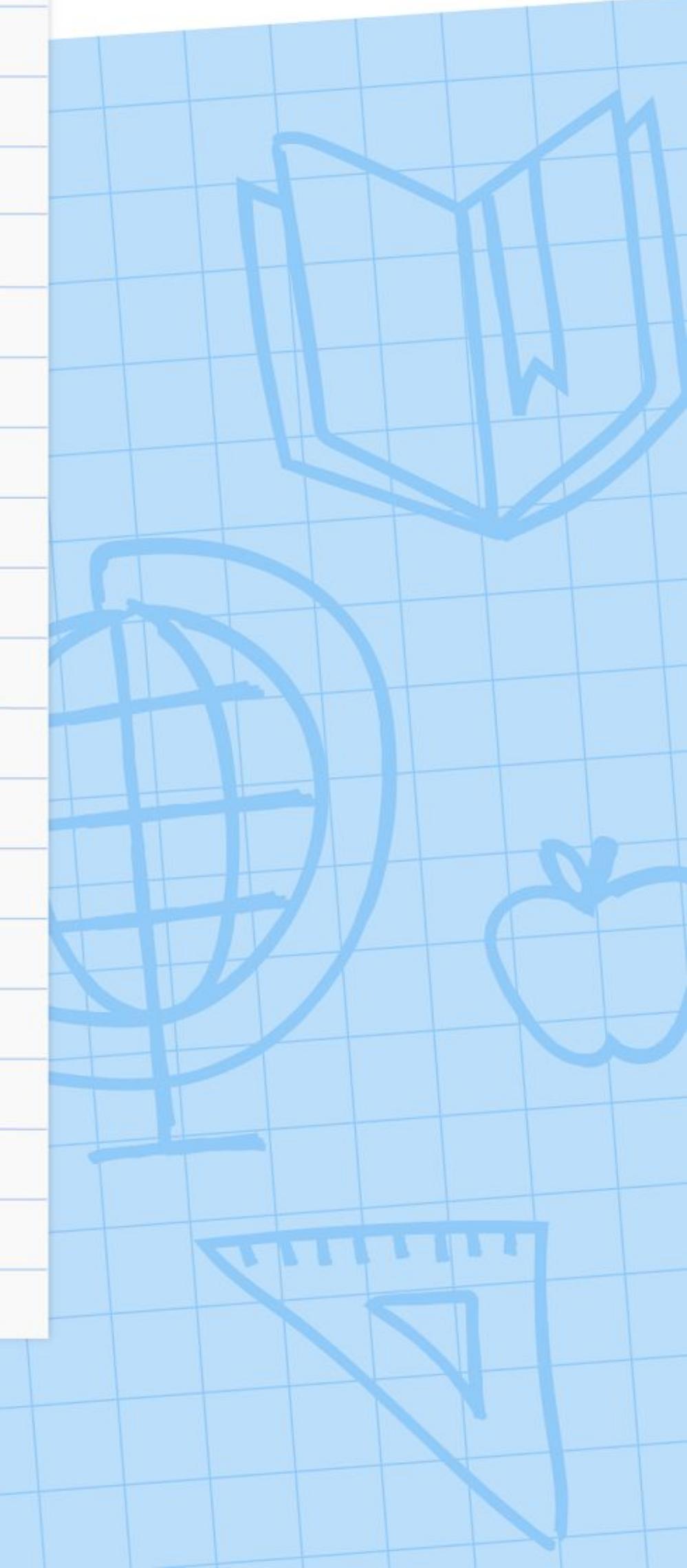
Developer Student Clubs
National Kaohsiung Normal University

視窗程式設計 從介面開始的軟體工程 ~

讀書會主持人：顏榕嶙(Bernie)、吳傢澂

日期：3/18 燕巢場、4/1 燕巢場、5/20 燕巢場、6/3燕巢場

```
filterByOrg = filterByOrg ? study.team_organization === filterByOrg : true
filterStatus = filterByStatus ? study.status === filterByStatus : true
const filterStudies = (studies, filterByOrg, filterByStatus) =>
  studies.filter(study => filterByOrg && filterStatus || !filterByOrg && !filterStatus)
```





目錄

1. 在那之前?
 2. 一個前端介面的基礎
 3. 必先苦其心志
 4. 元件之間的羈絆
 5. 跳脫思想的關鍵
 6. 更大的包容性
 7. 就像一棵樹一樣
 8. 美麗的視窗程式設計
 9. 所以, 然後呢?
- ★解鎖隱藏章節!!





Developer Student Clubs
National Kaohsiung Normal University

1

在那之前？

工欲善其事，必先利其器

```
const filterByOrg = study => study.lead_organization === filterByOrg;
const filterStatus = filterByStatus ? study.status === filterByStatus : true;
const filterPatchStatus = filterPatchStatus ? study.patch_status === filterPatchStatus : true;

function filterStudies({ studies, filterByOrg = false, filterStatus = false, filterPatchStatus = false }) {
    return studies.filter(study => filterByOrg || filterStatus || filterPatchStatus);
}
```



這是凱奇

他想要自己寫一個漂亮的應用程式
不過很可惜的，他完全沒有相關經驗

再加上，他的程式能力也算不上太好
在這種情況下，可以做些什麼呢？



工具和開發環境

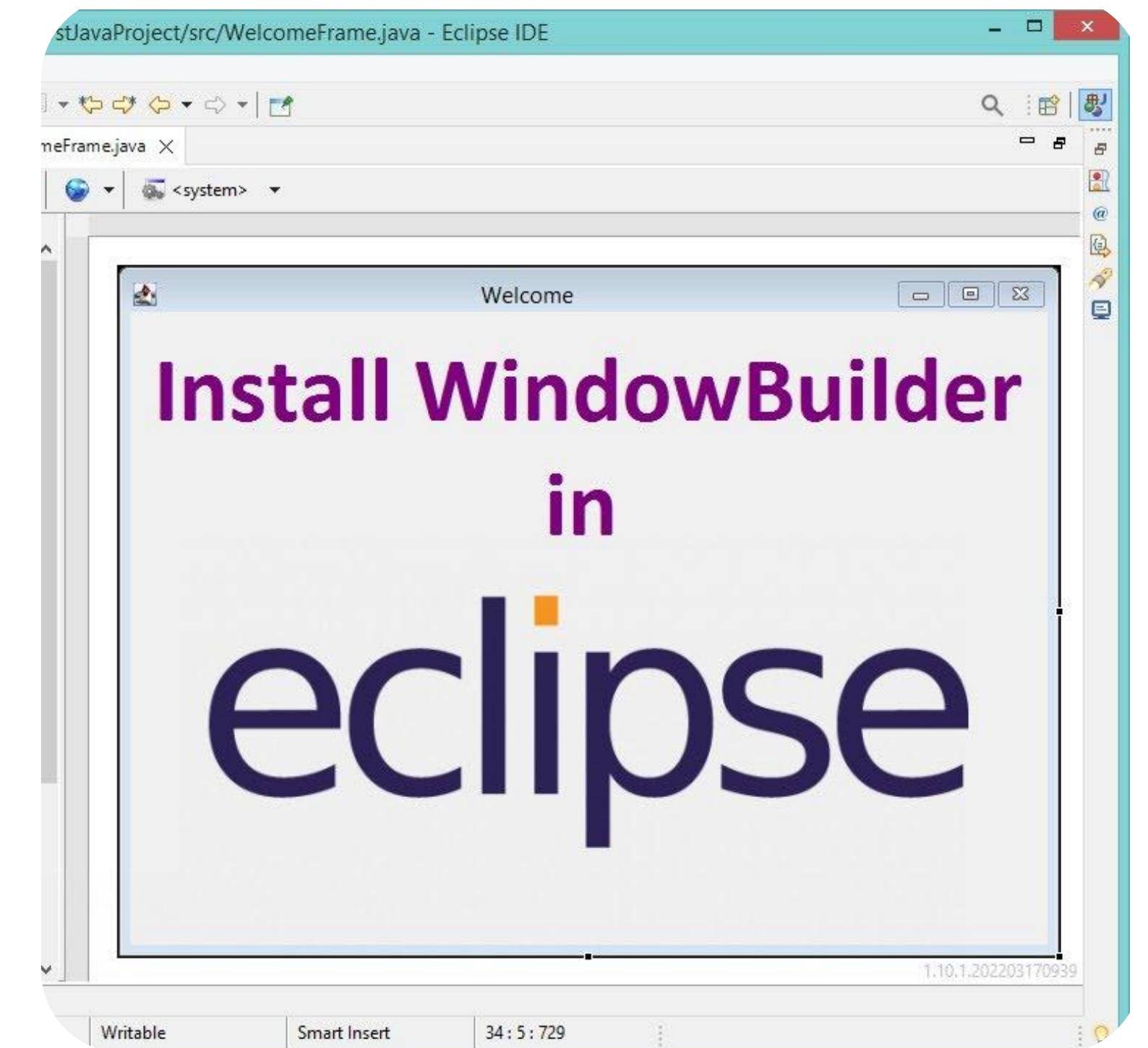
Eclipse和其提供的WindowBuilder

再突出的匠人也不能沒有自己的工具箱

接下來的課程當中，我們會用Java做教學

使用的軟體是安裝快速且常用的Eclipse
在Java Swing的使用上，他有提供設計界面

這一點在未來的開發上會很有用



安裝過程

交給螢幕分享和助教來一步一步帶著做

- 去官網上下載執行檔，走！
- 安裝完畢後開啟一個新的專案
- 在市集中安裝WindowBuilder插件
- 創建一個新的JFrame類別
- 然後就~大功告成！



eclipse



接下來要學什麼？

給接下來的課程帶來簡單的介紹

沒有人喜歡寫程式，實際上，我們會試著避開無聊的語法Blablabla

基於這是一堂視窗設計課程，培養對介面的基礎認知是必要的

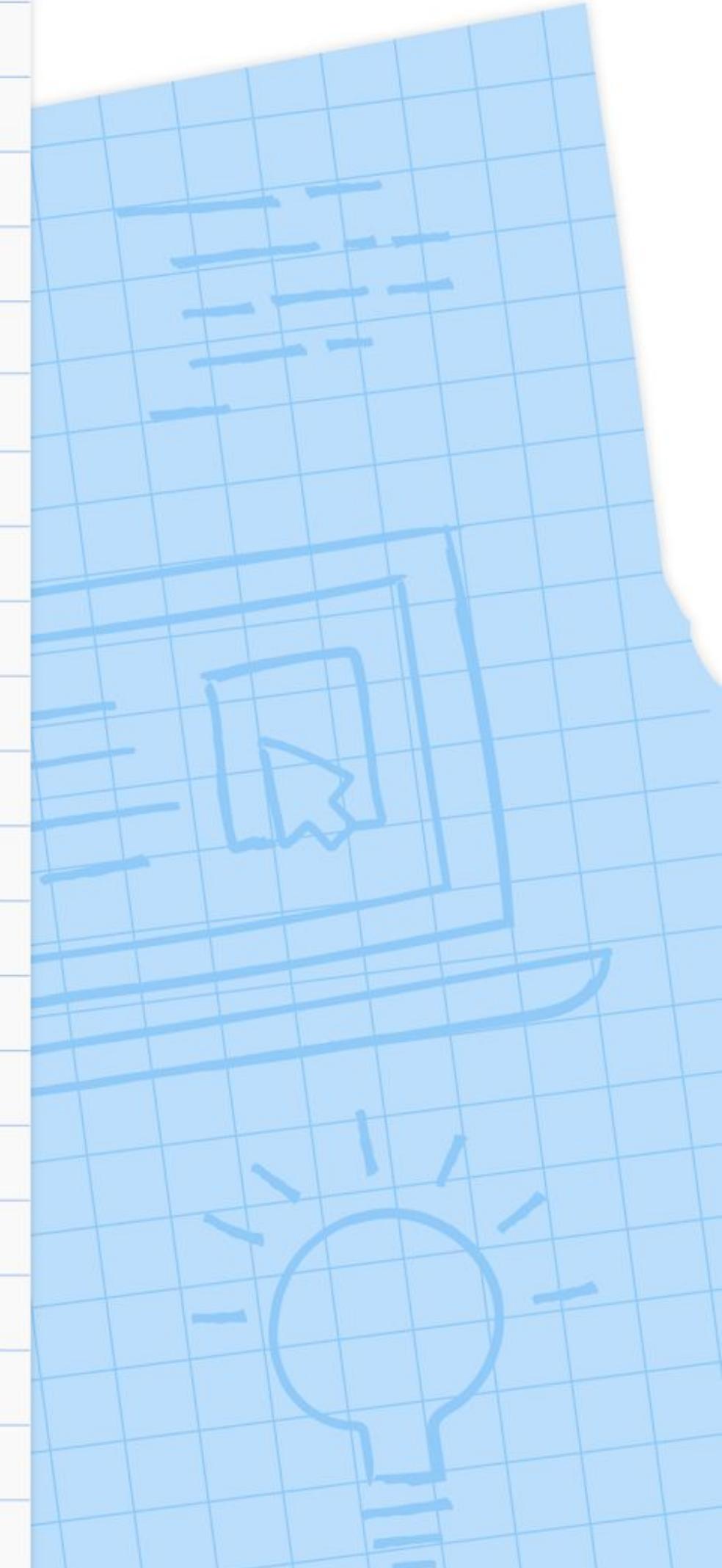
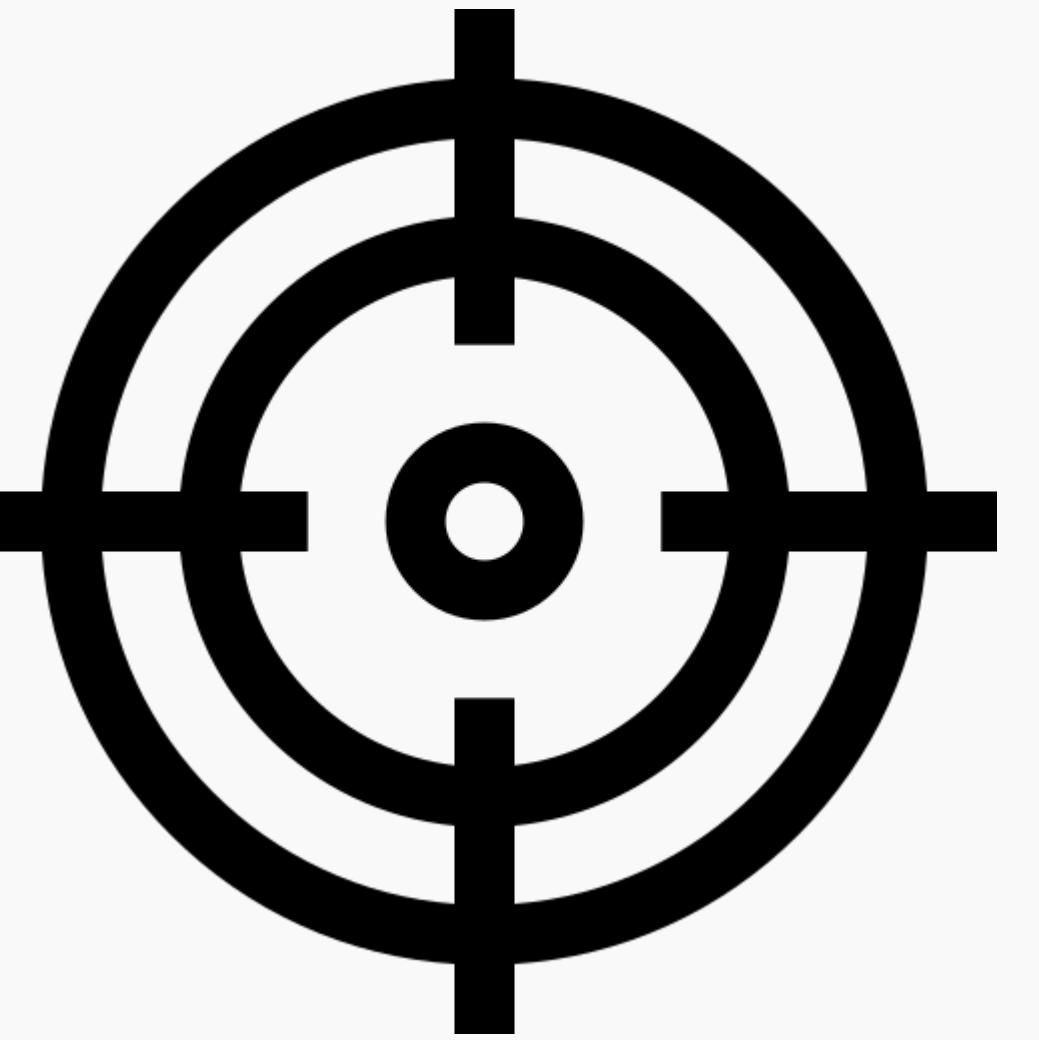
這堂讀書會總共有三個部分：認識元件、使用元件、應用元件

並期許在第三節課鐘聲響起之時，你們都能成功「拉」出獨一無二的視窗





那麼，讓我們進入重點吧。





Developer Student Clubs
National Kaohsiung Normal University

2

一個前端介面的基礎

如同原子一般，組成世間萬物

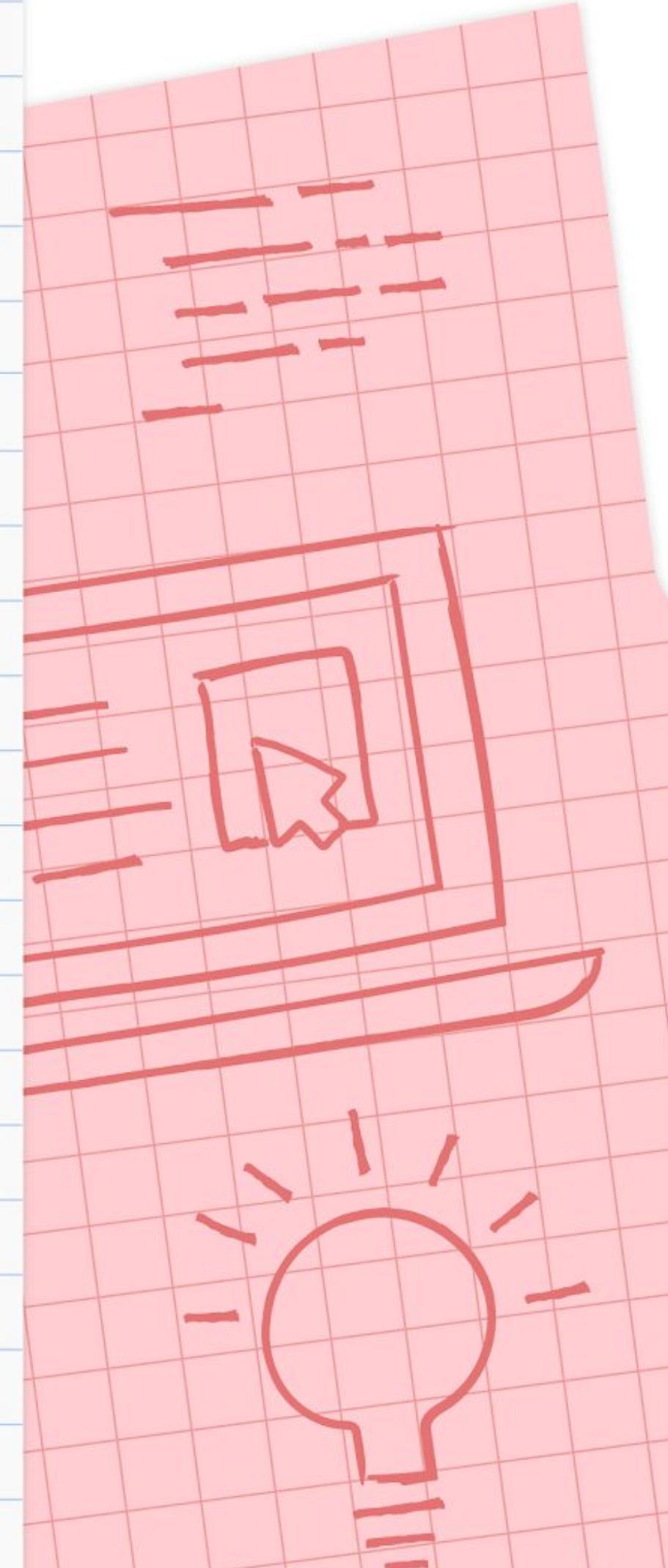
```
const filterByOrg = study => study.lead_organization === filterByOrg;
const filterStatus = filterByStatus ? study.status === filterByStatus : true;
const matchStatus = filterStatus || !filterStatus;

function filterStudies({ studies, filterByOrg, filterByStatus }) {
  return studies.filter(study => filterByOrg(study) && matchStatus);
}
```





所以，視窗界面的意義何在？



那又是什麼讓一個視窗「好看」的

當然排版也是一門技術，但終究還是歸功於元件

「元件」(Component) 聽起來是個陌生的詞

那我們直接這麼說好了。

訂閱「按鈕」

他是一個元件



難道什麼東西都是元件？

相信我，對，你看到的所有東西都是元件



他是一個元件

Google「標籤」



他是一個元件

單登的「輸入框」

是否為高師大在校學生 *

是
 否

想不到吧，他是一個元件

表單的「單選方塊」

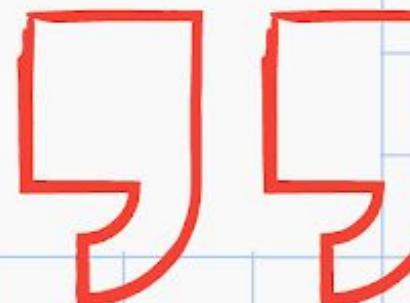
懂了，然後呢？

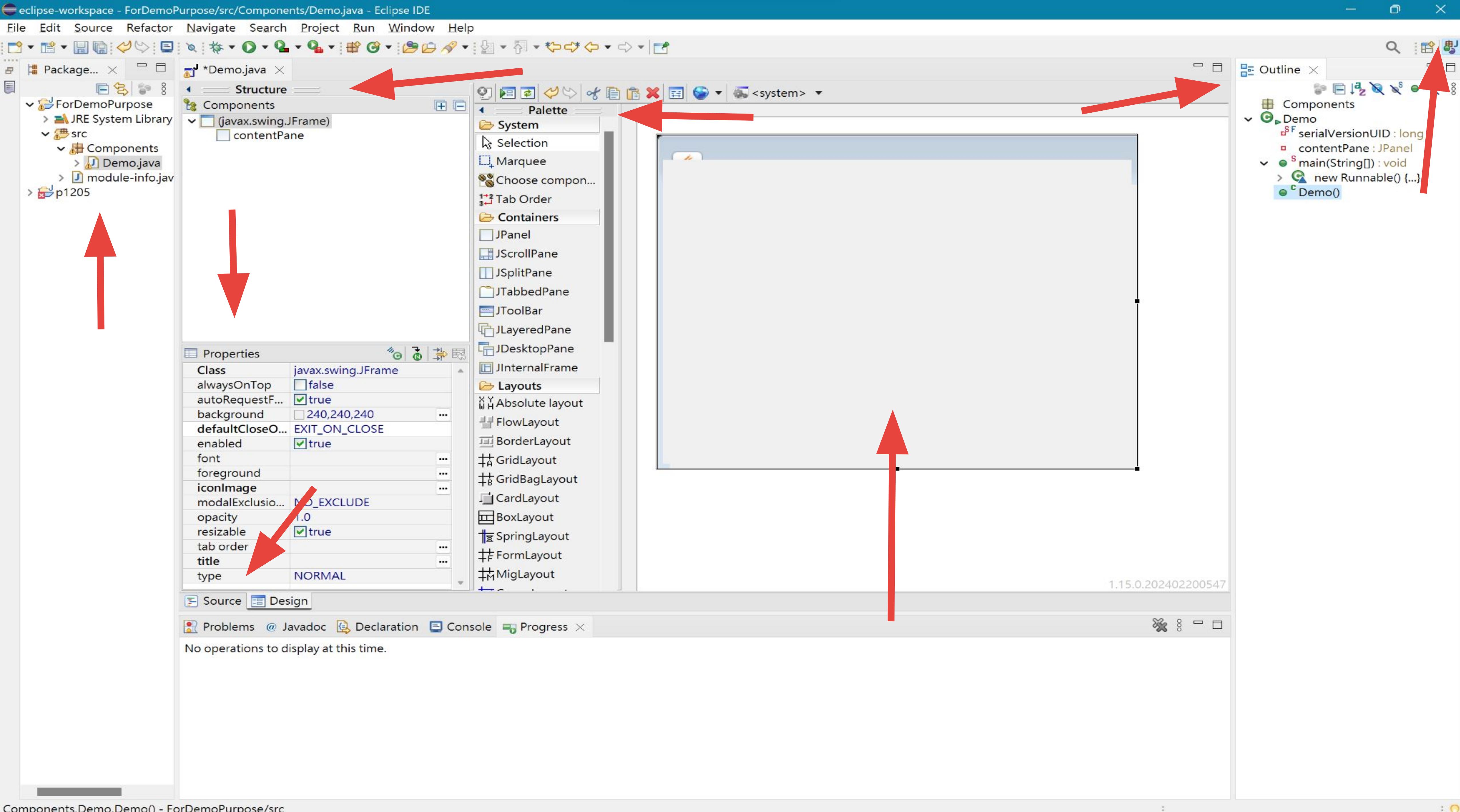


放心，我們沒有要把全部好幾百種元件都詳細介紹

不過在實際踏入實作之前，還有兩三個部份要提一下

相信我，這些都是必要的





喔不，頭昏眼花

不過說實話，他已經提供了一個很好開發的框架

實際的開發流程其實意外的簡單：

- 設定視窗佈局模式
- 從工具箱中拿出你要用的元件，把它定位在視窗之中
- 進到編輯器界面去連結功能函式 

 Google Developer Student Clubs



可以做到甚麼地步

我跟我們系的同學非常有禮貌的借了他的期末專題

分享螢幕，請！



其他需要注意的名詞介紹

快了，這個講完就可以下去實作了

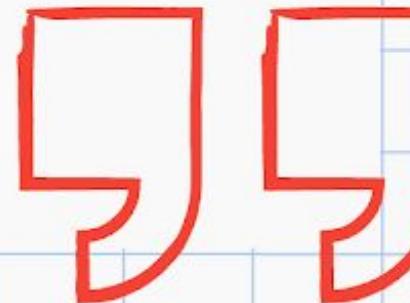
- 設計介面(Designer)、編輯器介面(Source): 元件的設計以及程式的實作區域
- 佈局(Layout): 決定元件將以甚麼形式去放在視窗當中
- 容器(Container): 也是一種元件，可以在其中放置更多元件
- 事件(Event): 決定一個元件會以甚麼形式去反應、接收/傳送訊號
- 屬性(Attribute): 一個元件的相關訊息，包含他的位置、功能等
- 建構式(Constructor): 元件的核心，讓一個實體能夠被創建



小小的休息



給你們時間去四處嘗試吧，現在是休息時間！



```
function filterStudies({ studies, filterByOrg = false, filter }) {
  if (!studies) return []
  if (filterByOrg) {
    return studies.filter(study => {
      const org = study.organizations[0]
      return org ? org.type === 'Organization' : true
    })
  }
  return studies
}
```



Developer Student Clubs
National Kaohsiung Normal University

3

必先苦其心志

成果是一步步累積出來的，而背後
富含著艱辛的歷程

```
filterByOrg = filterByOrg ? study.lead_organization === filterByOrg : true
filterStatus = filterByStatus ? study.status === filterByStatus : true
const filterStudies = ({ studies, filterByOrg, filterByStatus }) =>
  studies.filter(study => filterByOrg && filterStatus ? study.lead_organization === filterByOrg && study.status === filterByStatus : true)
```



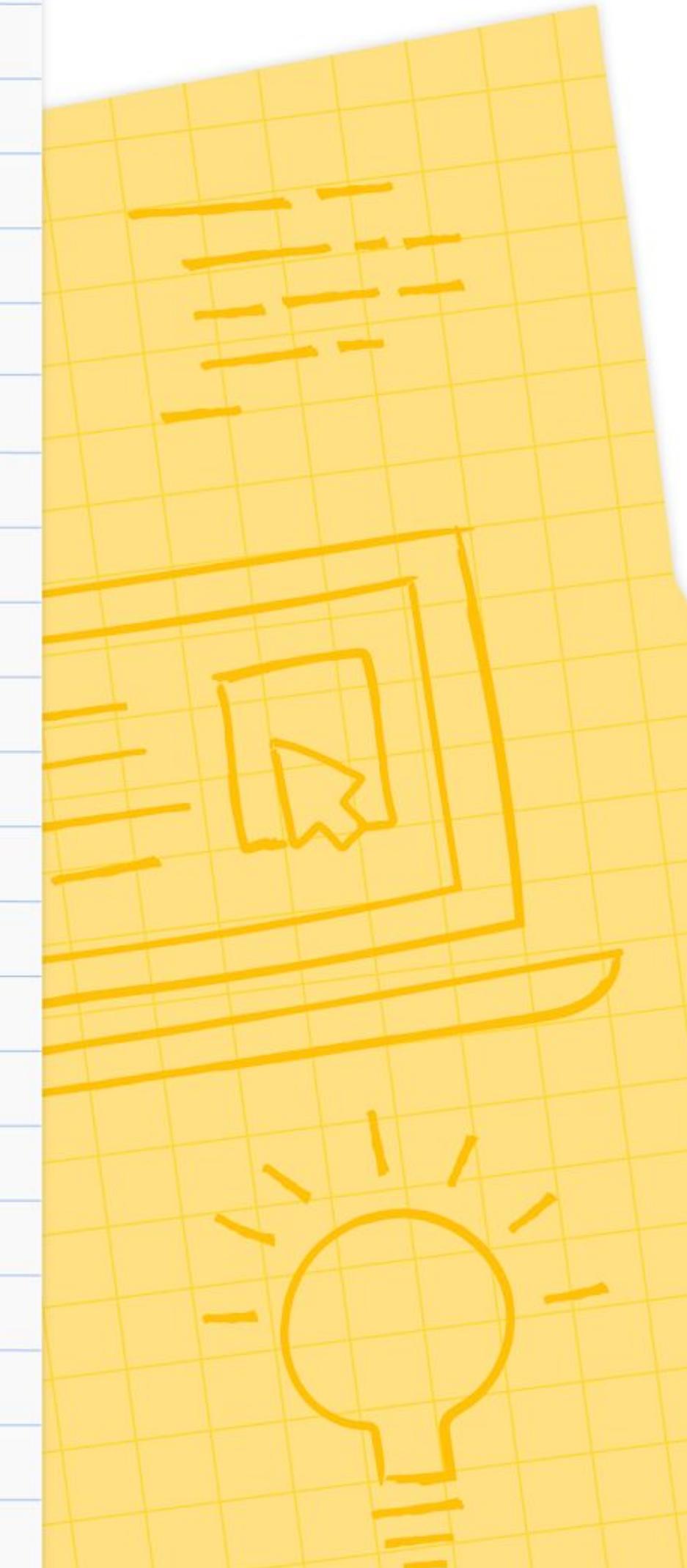


先有心理準備

我們現在有了元件們的基礎，但總不能白乾乾的看著他們被放在視窗上而已。



於是乎，是時候來把靈魂給刻進每一個元鍵當中，讓他們跟使用者能夠「互動」吧！



佈局，你的空間管理大師

今天要歸屬一個元件，就要告訴他要怎麼動

目前Eclipse支援的佈局模式如下：

- BorderLayout: 將視窗劃分成 東、南、西、北、中五個區塊
- FlowLayout: 將物件依序排列，可以設定從左邊、右邊或是中間開始排
- GridLayout: 用表格的方式平均劃分整個 ContentPane
- AbsoluteLayout: 將整個ContentPane化作座標平面
- CardLayout: 允許在同一個空間中堆疊多個元素，但僅顯示其中一個元素
- BoxLayout: 按照水平或垂直方向排列元素，適用於沿著線性方向排列的情況

還有其他的我們就先暫時不提及



監聽器是一種隱形的翅膀

介於前端與後端之間不可或缺的角色

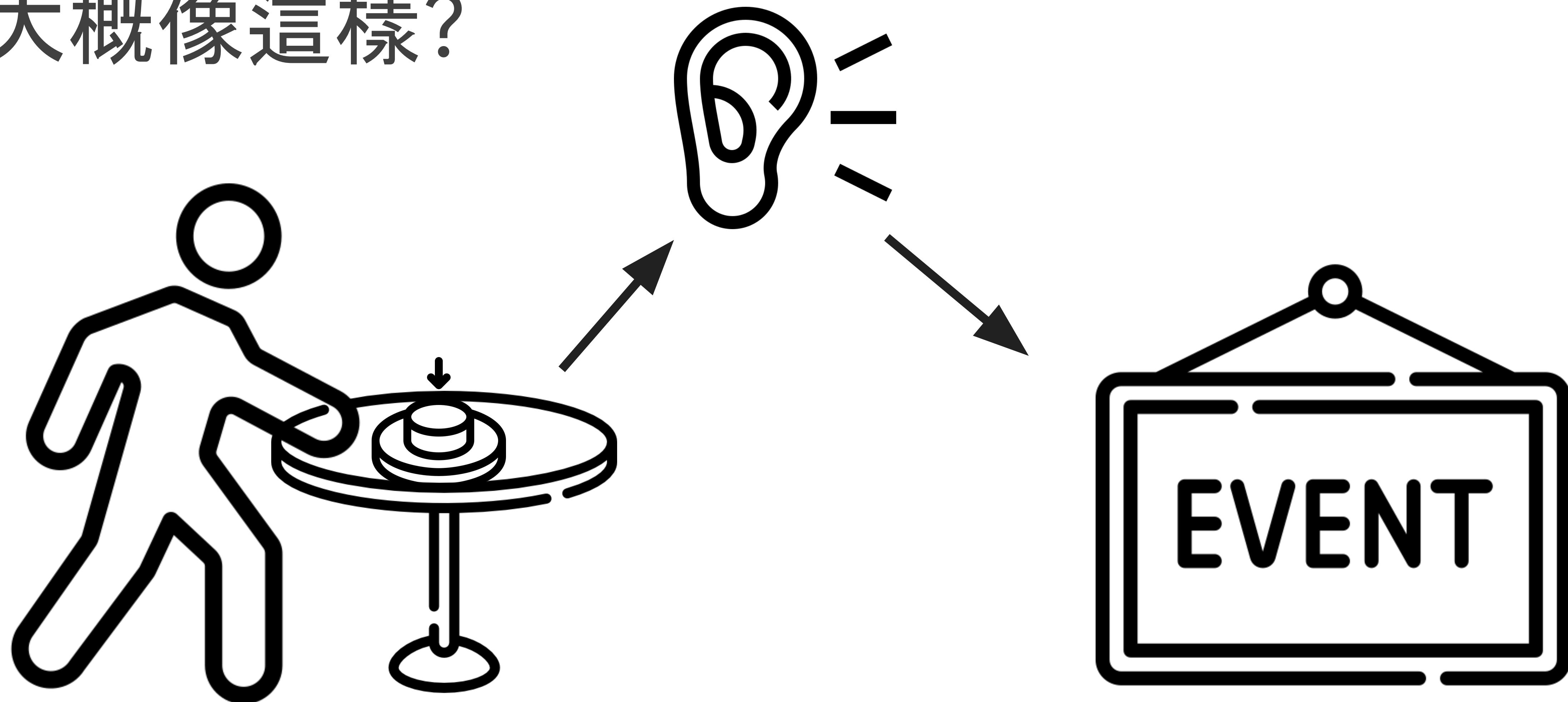
讓我們想象一個簡單的場景：你有一個按鈕，當你按下它時，會在應用程序中顯示一個消息框。

在這個操作當中，有一個你看不見的角色正在記錄著使用者的一舉一動，我們稱呼他為監聽器。

當你按下按鈕時，按鈕元件就會發出一個事件，這個事件被按鈕監聽器捕獲並處理，最終導致顯示一個消息框。這種關係就像是按鈕元件告訴監聽器：“嘿，我被按下了！”，而監聽器則回答：“好的，我知道了，現在我們該怎麼處理這個事件呢？”。



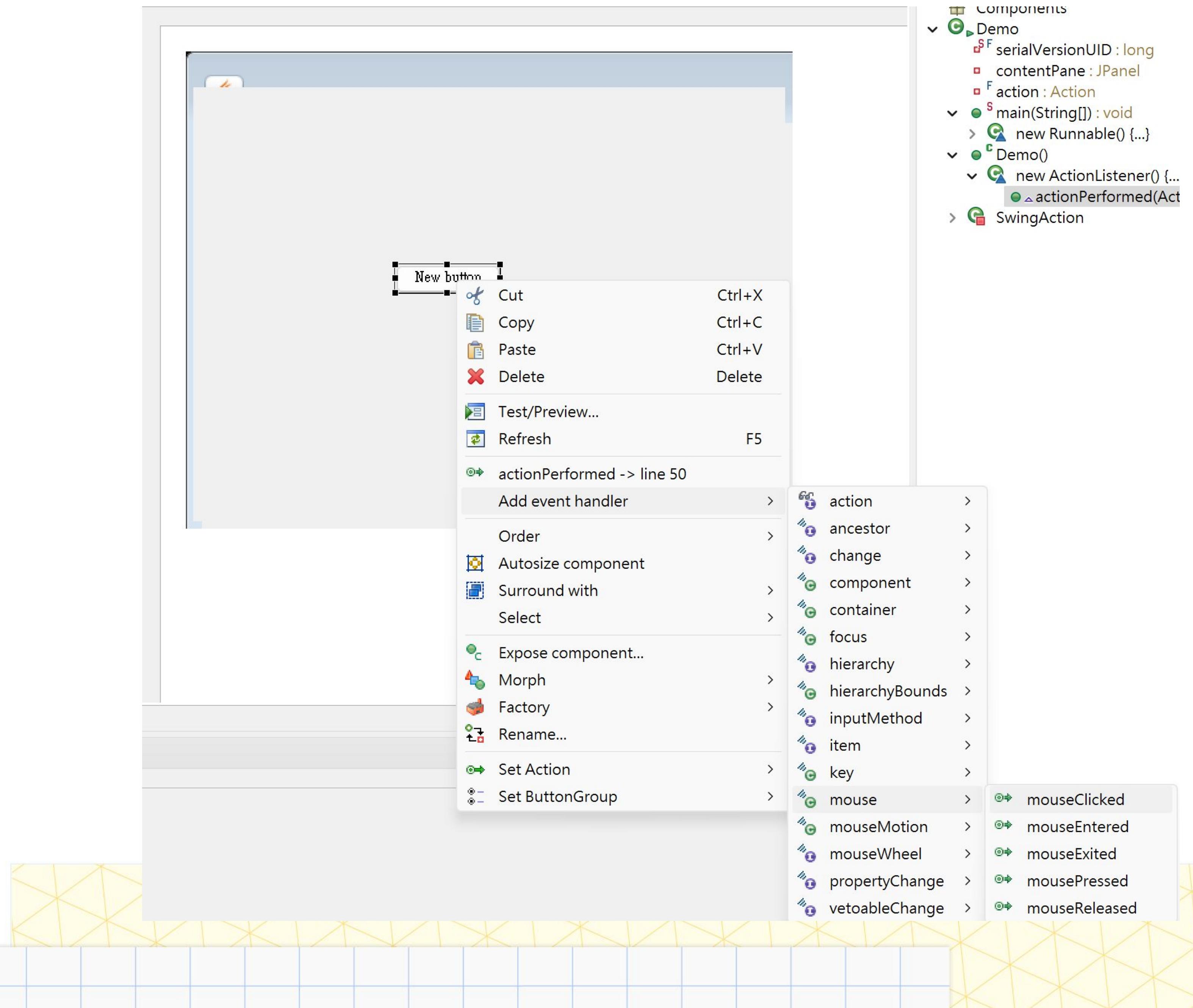
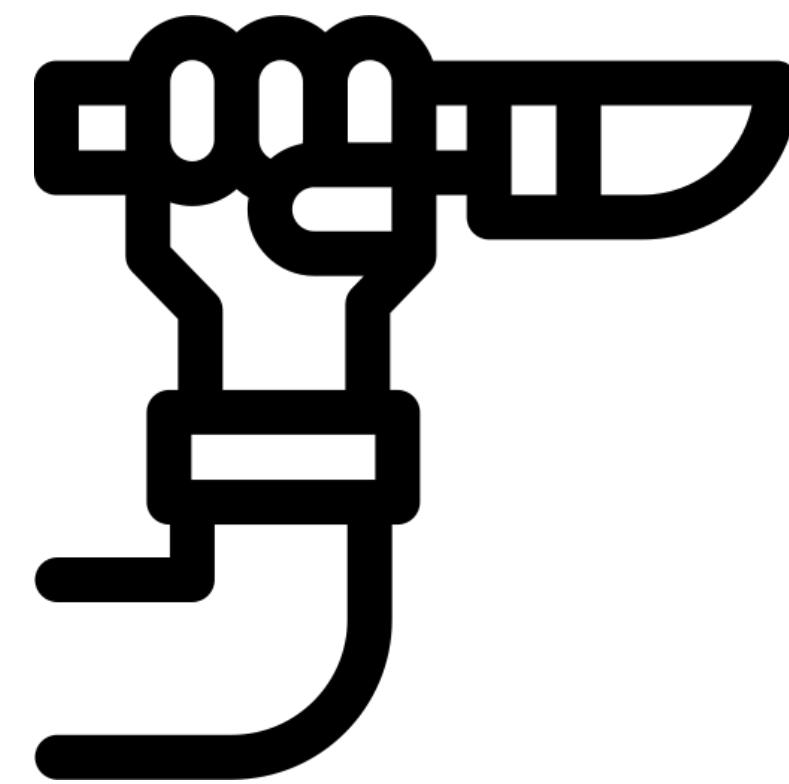
大概像這樣？



從這裡開始

設定一個你想要擷取的事件

然後就開始大開殺戒



The screenshot shows a Java Swing application window with a context menu open over a component labeled "New button". The menu includes standard options like Cut (Ctrl+X), Copy (Ctrl+C), Paste (Ctrl+V), Delete, Test/Preview..., Refresh (F5), and Add event handler. The "Add event handler" option is expanded, showing a list of event types. On the right side of the screen, there is a code editor window displaying Java code for a class named "Demo". The code includes a main method, a constructor, and an actionPerformed event handler. The "actionPerformed" method is highlighted in the code editor.

```
Components
C Demo
  SF serialVersionUID : long
  contentPane : JPanel
  action : Action
  S main(String[]) : void
    new Runnable() {...}
  C Demo()
    new ActionListener() {
      actionPerformed(Act
  G SwingAction
```

New button

- Cut Ctrl+X
- Copy Ctrl+C
- Paste Ctrl+V
- Delete Delete
- Test/Preview...
- Refresh F5
- actionPerformed -> line 50
- Add event handler >
- Order >
- Autosize component >
- Surround with >
- Select >
- Expose component... >
- Morph >
- Factory >
- Rename... >
- Set Action >
- Set ButtonGroup >

- action >
- ancestor >
- change >
- component >
- container >
- focus >
- hierarchy >
- hierarchyBounds >
- inputMethod >
- item >
- key >
- mouse >
- mouseClicked >
- mouseEntered >
- mouseExited >
- mousePressed >
- mouseReleased >

```
serialVersionUID : long
contentPane : JPanel
action : Action
main(String[]) : void
  new Runnable() {...}
Demo()
  new ActionListener() {
    actionPerformed(Act
SwingAction
```

稍等一下!!



我們可能會需要一點點的物件導向!

在進到最難的部分之前，請容許我們介紹一下所謂Java以及其他物件導向語言的核心概念！

(語法的部分可見附錄 2)



我們在一個魔法工廠裡面

所謂的物件導向就是給你藍圖，做出成品

在這個比喻中：

- **工廠** 就像是我們的程式，它是用來創造不同功能的。
- **機器和工具** 就像是物件，它們有各自的功能和特點。比如，在Java中，我們可以有不同的物件，比如按鈕、文本框等。
- **製造的魔法物品** 就像是我們的程式所創造的東西，比如一個GUI視窗，裡面有按鈕、文本框等元件。



元件，物件？

兩者相似卻又不盡相同

先前提到了Component元件這個概念，那物件又是甚麼呢？

元件是你能夠在一個視窗中見到的所有東西，例如按鈕之類的。

當然，你可以創建很多個按鈕，此時這些不同的按鈕就是不同的物件。

可以想像成，你根據一個藍圖做出了很多的成品。

而這就是物件導向程式語言的核心觀念。



喋喋不休的最後通牒

就是要你直接了解Java的本質

下一頁我們就要開始帶到程式的部分了，希望你們還沒頭昏腦脹。

讓我們來統整一下關鍵的概念：

- 監聽器可以拿來擷取事件，像是滑鼠按下、狀態變更時等等
- 讓元件被實作的方法就是建立一個物件(就是把他從工具箱拿出來的動作)
- 所有元件都有其自帶的屬性，有些甚至是獨特的 (在第四章會詳細介紹)

接下來，讓我們的魔法工廠開始運作吧！



程式的畫布

函式、方法，誰知道呢？

```
JButton BigB = new JButton("New button");
BigB.addMouseListener(new MouseAdapter() {
    @Override
    public void mousePressed(MouseEvent e) {
    });
BigB.setBounds(167, 116, 131, 47);
```

回到前面提到的。

在我們加入了一個監聽器之後就能夠擷取相對應的事件，而我們就會在這個地方加入程式，讓他能夠在接到事件發生時執行相對應的動作。

在大括號{}之間就是我們的畫布。



那要寫什麼？

這個地方可要熱起來了

所有人的第一支程式都是「你好，世界」，這裡當然也不意外。

在Java裡，輸出文字的方法是System.out.println("Your Text");

是的，跟Python的print("Your Text")、C++的std::cout << "Your Text"; 比起來可怕了許多，但是你們可以這樣理解：

System(系統) out(輸出) println(印出物件+換行) => 你的要印出來的東西



我好厲害！

就這樣，我們成功的讓按鈕不只是裝飾了

接下來，時間要加速了。

比起簡單的一加一，我們要往更加遠大的夢想而邁進。

要這麼做的話，我們要先從元件的屬性開始講起。

The screenshot shows an IDE interface with a Java code editor and a console window.

Java Code (Source tab):

```
43 public Demo() {  
44     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
45     setBounds(100, 100, 492, 387);  
46     contentPane = new JPanel();  
47     contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));  
48  
49     setContentPane(contentPane);  
50     contentPane.setLayout(null);  
51  
52     JButton BigB = new JButton("New button");  
53     BigB.addMouseListener(new MouseAdapter() {  
54         @Override  
55         public void mousePressed(MouseEvent e) {  
56             System.out.println("我好厲害!");  
57         }  
58     });  
59     BigB.setBounds(167, 116, 131, 47);  
60     contentPane.add(BigB);  
61 }  
62 }  
63 }
```

Console Output:

```
我好厲害!  
我好厲害!  
我好厲害!  
我好厲害!  
我好厲害!  
我好厲害!  
我好厲害!  
我好厲害!
```



4

元件之間的羈絆

元件之間的精神，又稱元神

```
const filterByOrg = study => study.lead_organization === filterByOrg;
const filterByStatus = filterByStatus ? study.status === filterByStatus : true;
const filterMatchStatus = matchStatus ? study.status === matchStatus : true;

function filterStudies({ studies, filterByOrg, filterByStatus, matchStatus }) {
  return studies.filter(study => filterByOrg(study) & filterByStatus(study) & filterMatchStatus(study));
}
```





人際互動，但在程式裡

當然，我們可以讓終端機幫我們完成大部分的事，不過在打包成應用程式之後哪來的終端機呢？

我們要獨立自主，就必須讓元件們聯合起來，一切都在使用者介面裡完成。



我們今天的目標

以下是一個小型的BMI計算機，那麼我們要怎麼做出來呢？



屬性，這是魔法，卻也不是

每一種元件都有屬於自己的運作方式

遠在第二章的時候我們就講過屬性的相關資訊，在這裡我們會講得更加詳細。

一個Button擁有text、icon等等屬性，我們可以直接修改，也可以透過程式變動。

這就像是一個元件的技能，每個元件都不同，有些甚至是獨特的。
而並不是只有元件本身能夠存取自己的屬性，這是公開的。

例如，一個Button的事件可以取用TextField的文字等等...有甚麼想法了嗎？

按一下即可新增標題

我們的確能夠這麼做！

假設說，我們有一個TextField、一個Button和一個Label。
讓那個Button按下時擷取TextField裡面的文字，並把他擺出來到Label裡面。

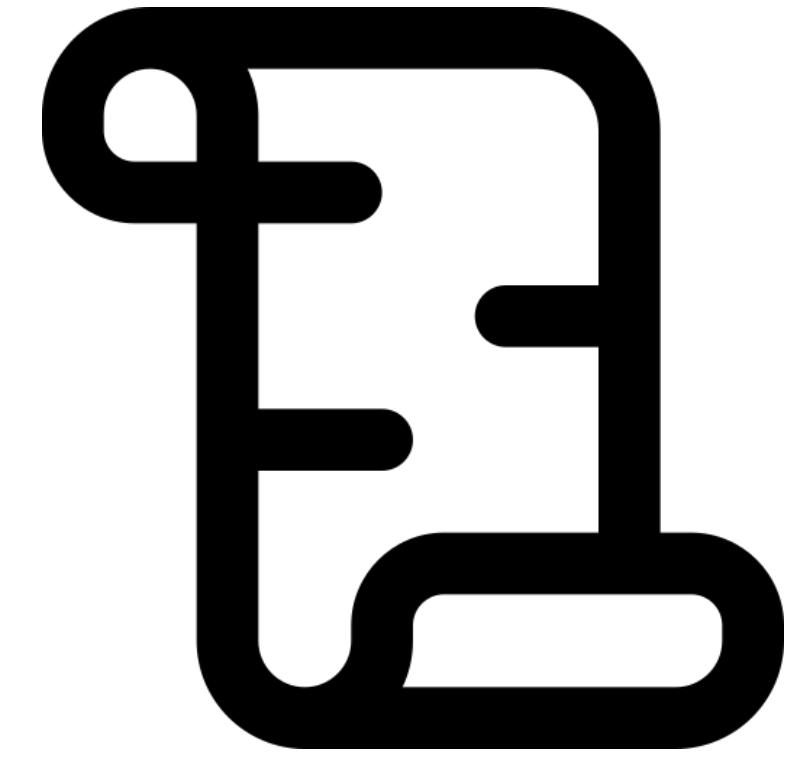
這不就能達到基礎的按一下新增標題了嗎？

沒有錯，魔法是想像的世界，程式是魔法因此他也是想像出來的。
我們當然可以這麼做。



身家調查

了解如何傳遞資訊



經過了簡單的觀察，我們會發現TextField和Label都有一個共同的屬性: text
如果要去取用它們的資訊，可以使用萬能的get/set函數

步驟是這樣: 從TextField中get到使用者輸入的文字，並將其set到Label裡面

所有的動作會在我們按下按鈕之後進行，因此可以這樣做總結...

看上去複雜多了

但實際上，真的那麼難嗎？

```
 JButton confirm = new JButton("確認");
confirm.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        if(!titleBox.getText().equals("")) {
            String title = titleBox.getText();
            titleDisplay.setText(title);
            titleBox.setText("");
            System.out.println("Success");
        }else {
            System.out.println("No success :(");
        }
    }
});

confirm.setBounds(272, 181, 87, 23);
contentPane.add(confirm);
```

就如前頁所說，在這裡我們有一些關鍵的部分：

- 整個程式位在confirm按鈕的滑鼠按下事件中
- 建立了一個字串變數來記錄titleBox裡的文字
- 將titleDisplay的文字替換為該變數
- 清除titleBox裡面的所有文字

在其中，titleBox是一個TextField，titleDisplay是一個Label。

至於外部的if判斷式我們就先不細講了。
(見附錄 3-1)

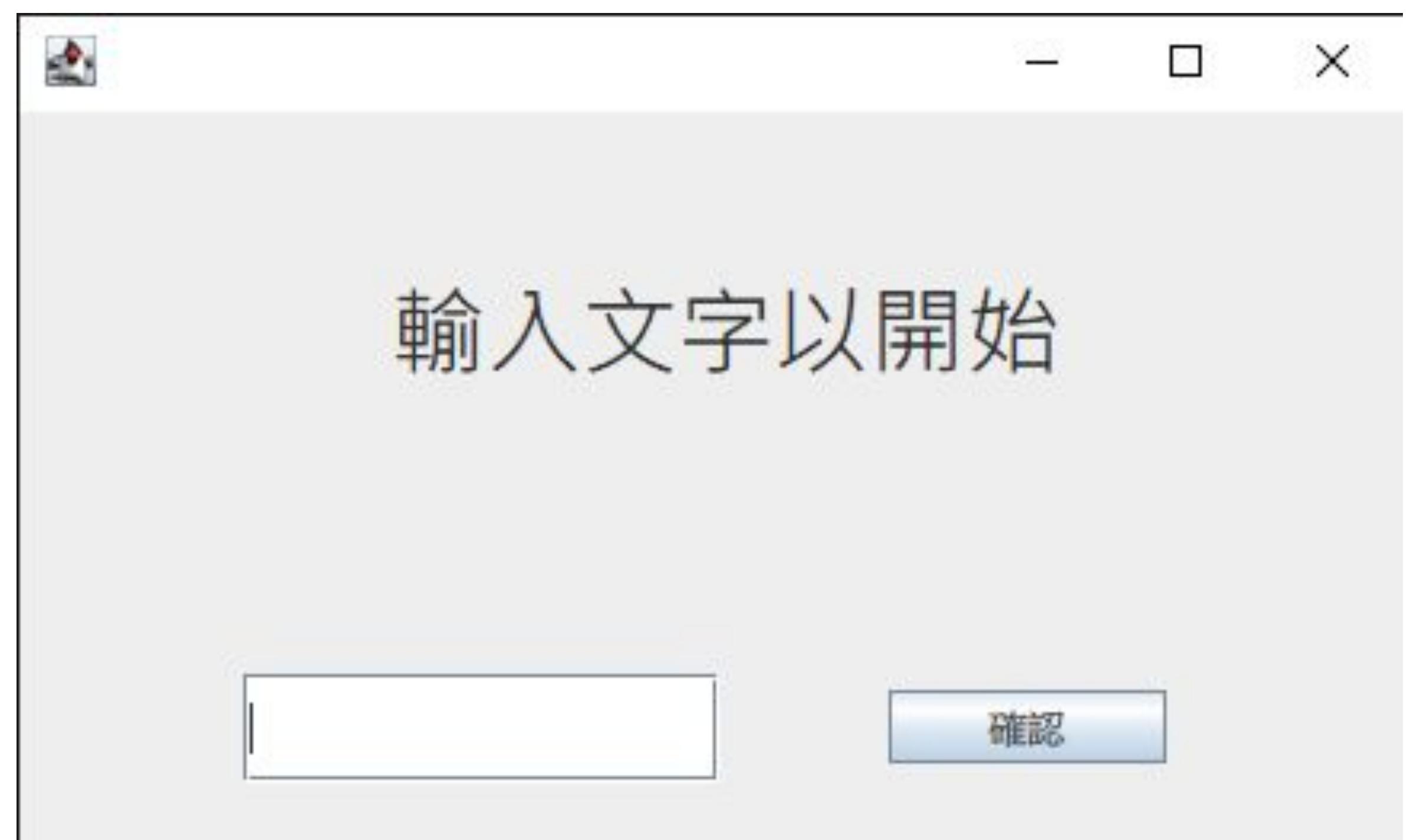
成果展示？

只寫了四行的程式能夠做出些什麼？

就這樣，陽春，卻有用！

可以測試看看是不是輸入文字、按下按鈕後會直接更新到上面的Label。

如此一來，一個簡易的應用程式就完成了！



回到正題

有了開發經驗後，我們來思考一下要如何達到原先的目標吧！

一個BMI計算機必定要有計算的功能，所以我們來分析一下：

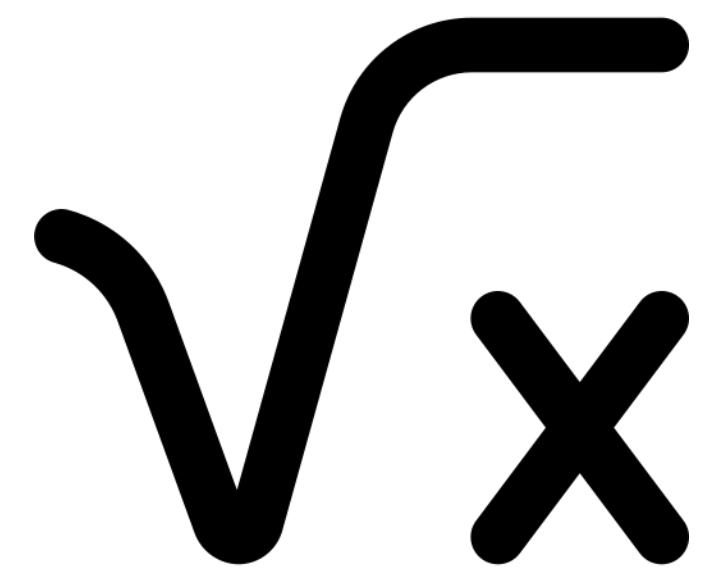
首先，要分開獲得輸入的身高還有體重

接著，把兩個數值給套進公式裡，得出計算過的BMI

最後，把得出來的BMI結果給展出出來

欸？我們是不是大部分都已經會了？

數學，對 處理數字的要點



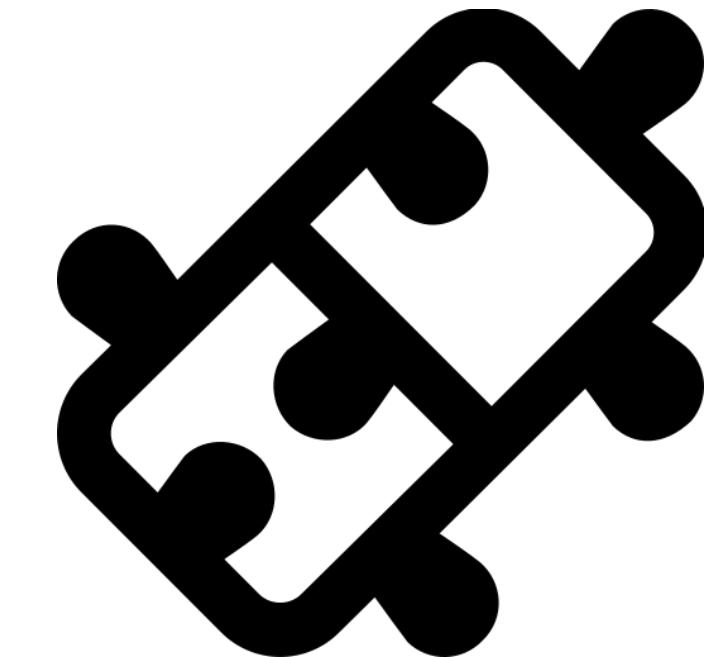
其實稍微敏銳一點的人可能已經發現了，從TextField的屬性中抓出來的text是字串，在程式當中我們是沒有辦法用字串直接去做數學運算的，何況計算BMI。

在Java中，要把字串轉成數字的方式大致有兩種: `parse[Type]` 和 `valueOf`

譬如說我們要轉成整數的話就是`Integer.parseInt(字串)`和`Integer.valueOf(字串)`我們通常會用前者，因為後者是轉成物件型態。

我們已經獲得了所有的拼圖!

想不到吧？



首先，我們處理身高與體重的數字，記得身高要除以一百(求公尺)。
接著我們直接套公式就好，`Math.round`是用來取整數的。
然後就是把他給展示出來，我們這裡用的是`TextPane`，可以想成有框的`Label`。

```
calculate.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        Double tall = Double.parseDouble(tallTextField.getText()) / 100; // 得出輸入的身高
        Double weight = Double.parseDouble(weightTextField.getText()); // 得出輸入的體重
        Long BMI = Math.round(weight / Math.pow(tall, 2));
        BMItextPane.setText(BMI.toString());
    }
});
```

從零開始的視窗製作!

恭喜大家來到這一步！



經過了兩節課的洗禮之後，我相信大家或多或少都對視窗設計有一些基礎的了解了，這是一個大坑，無論是Java也好，WindowBuilder也好。

希望這幾個禮拜的課程能夠帶給你們不少的收穫！

接下來呢...我們會再多補充一點東西





Developer Student Clubs
National Kaohsiung Normal University

5

跳脫思想的關鍵

Think outside the box, but which?

```
filterByOrg = filterByOrg ? Study.tead_organization === filterByOrg : true
filterStatus = filterByStatus ? study.status === filterByStatus : true
const matchStatus = filterStatus ? study.status === filterStatus : true

function filterStudies({ studies, filterByOrg = true, filterByStatus = true }) {
  const filteredStudies = studies.filter(study => {
    if (!filterByOrg) return true
    if (!filterStatus) return true
    if (matchStatus) return true
    if (study.tead_organization === filterByOrg && study.status === filterByStatus) return true
    return false
  })
  return filteredStudies
}
```

擴大搜尋範圍!



到目前為止，我們用的都是Java Swing提供的預設材質，但看久了真的很膩。

今天假設我有一個準備好的圖片，有甚麼辦法可以把它給放到我們的視窗裡嗎...？



哈利，你是個巫師！

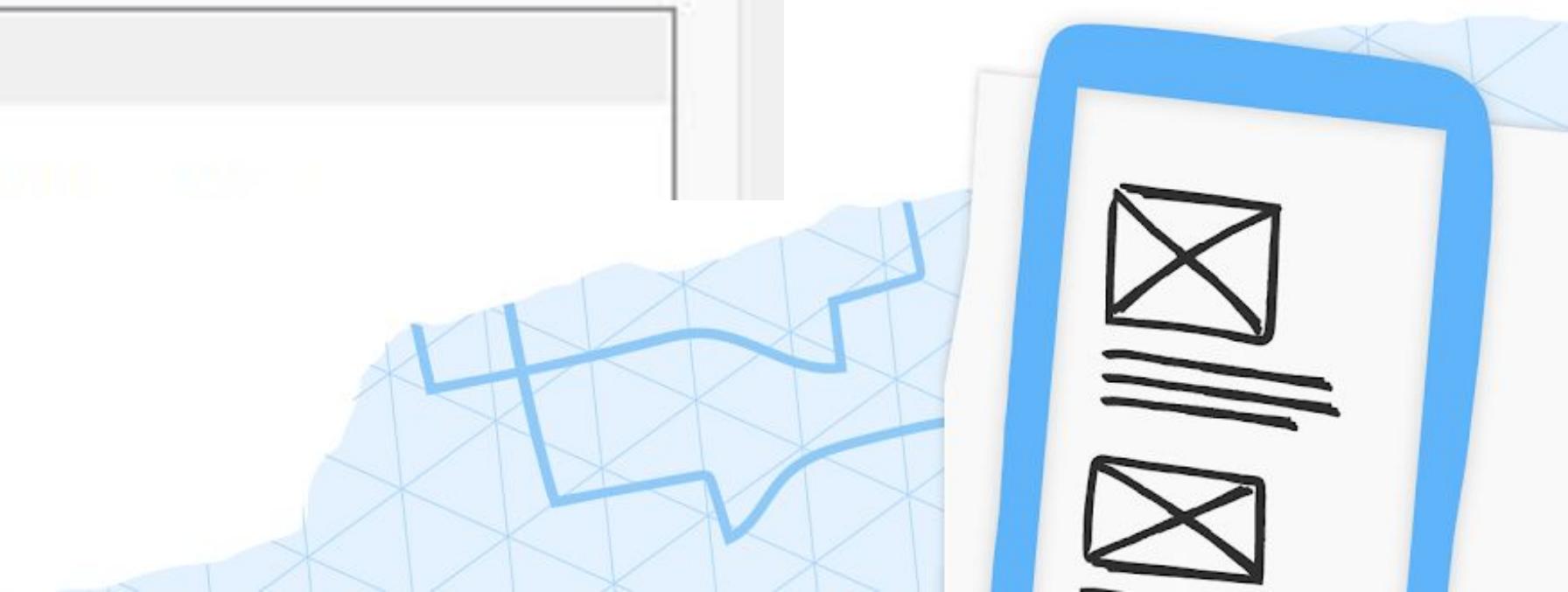
就算你不是哈利，你還是可以當巫師！



在實際將檔案物件給導入到程式裡之前，我們要先來理解一下這是怎麼運作的。

大家可能或多或少都有注意過所謂的檔案路徑吧？

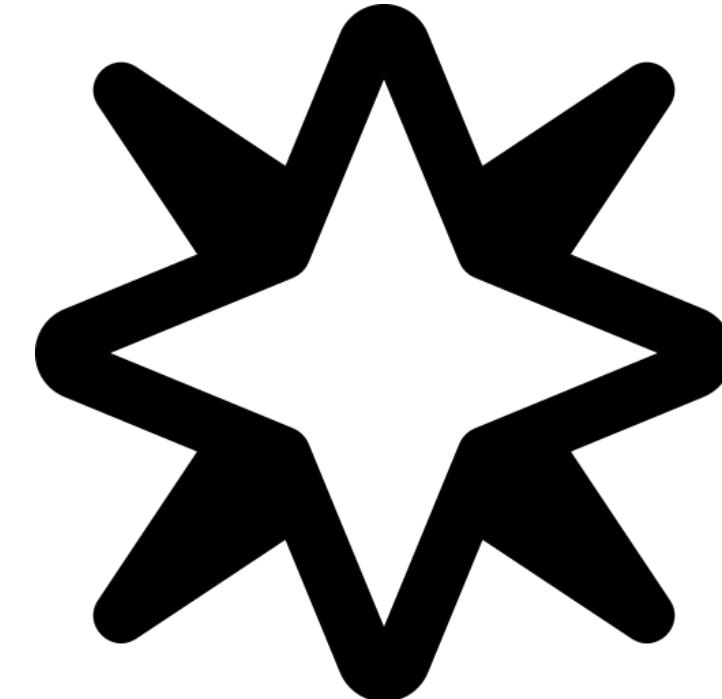
看阿，在這裡



亘古不變 v.s 星移斗轉

絕對與相對的重大差異

路徑在我們電腦科學中被分為了兩種：絕對與相對路徑



絕對路徑，顧名思義就是獨特且不會變動的地址，例如：

"D:\Videos\Videos\Ephemeral Tranquility.mp4"

這種直接註明硬碟以及完整位置的就是絕對的路徑(也可稱為OS Path)

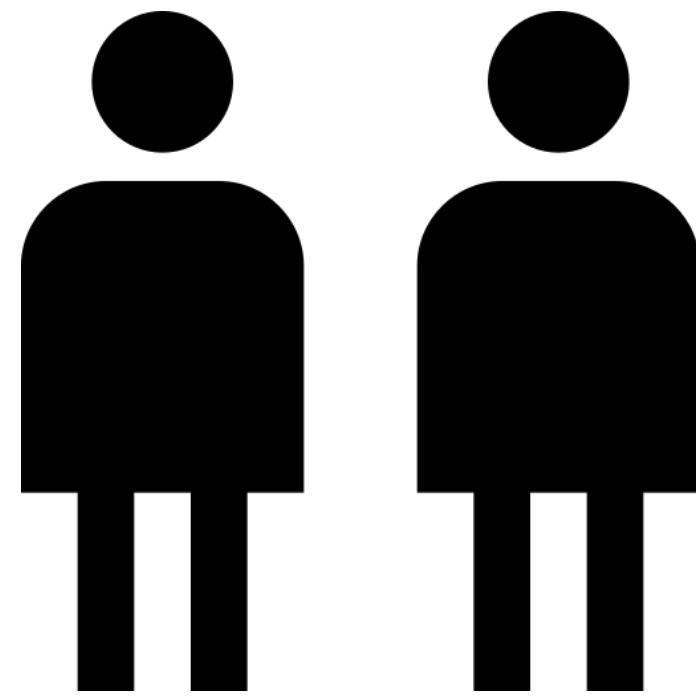
相對路徑是檔案和當前操作環境的相對距離，就像這樣：

".\Videos\Ephemeral Tranquility.mp4"

這個寫法會找到這裡為基準，父資料夾當中的Videos資料夾裡的檔案

尋找一個合適的替身

圖片也要有立足之地才行



理應上來說，我們可以導入任何的檔案，但在現在的情況下PDF、PPT甚麼的要放進視窗裡面太難了，倒不如先來嘗試圖片吧？

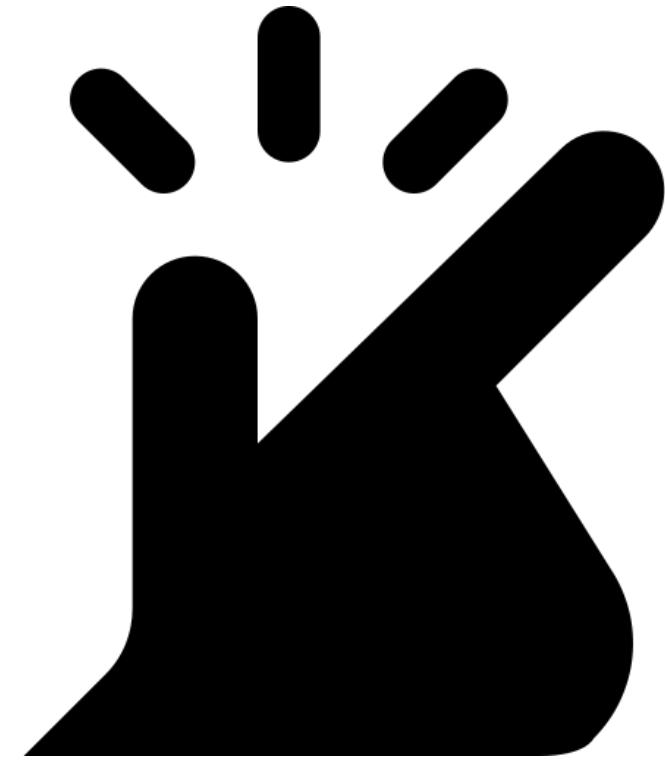
要展示圖片的話，一般來說我們會使用標籤(JLabel)來作為基礎

你會發現他有一個獨特的方法: `setIcon(ImageIcon icon1)`
這表示了他可以接受一個Icon物件作為他的參數來使用

所以我們只要想辦法把圖片給變成一個ImageIcon，然後再丟進來就好了

簡單的方法

我們直接創造一個相對路徑！



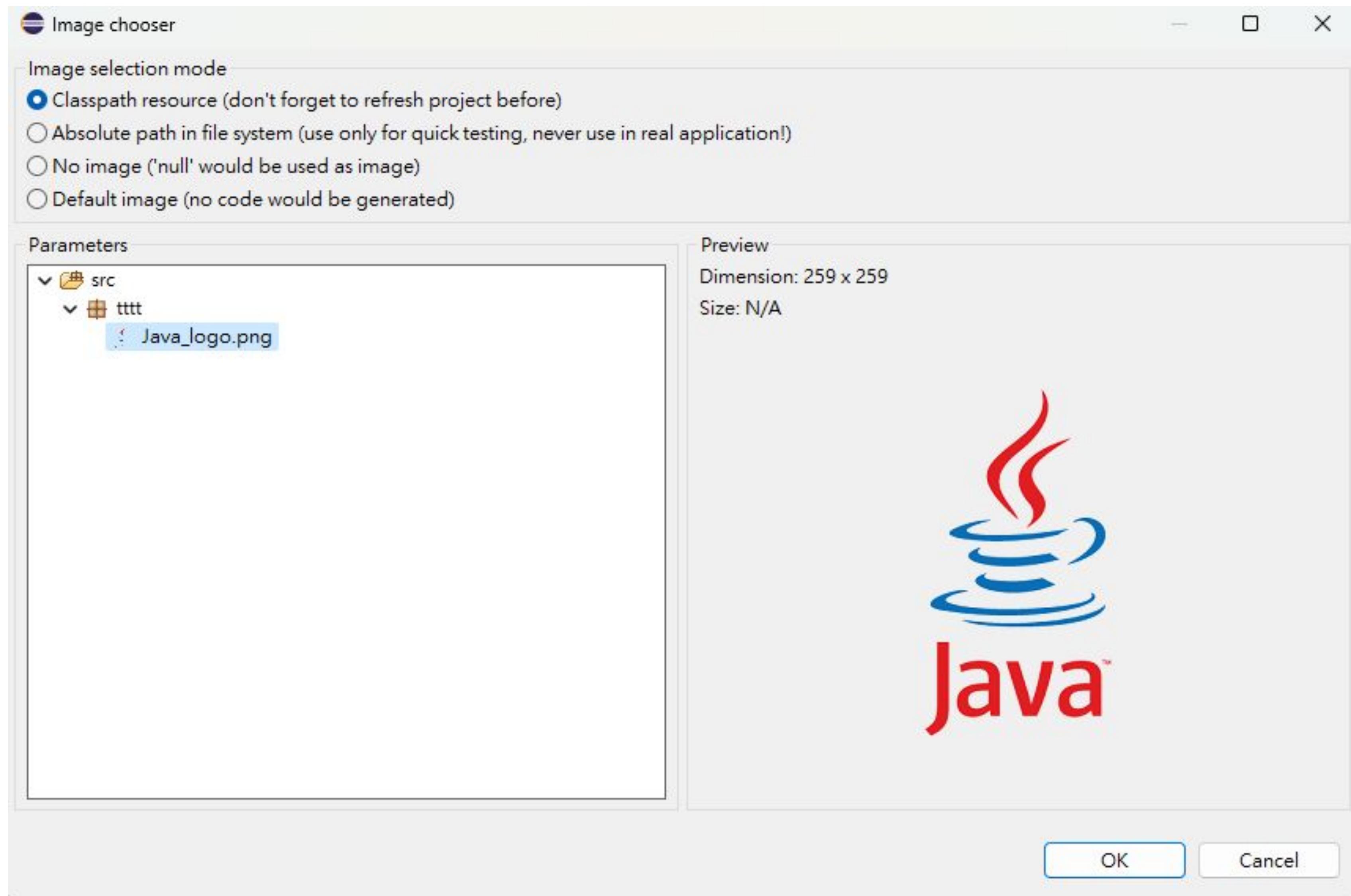
簡單的方法就是簡單，我們直接把裝著圖片的資料夾 -> 丟到專案的Package裡面

現在這個資料夾是我們的一部分資源了
接著就是打開JLabel的屬性，找到Icon的部分，點個幾下

他應該會跳出一個視窗，讓你可以選擇你要的圖片(記得選擇第一個選項)

然後耶！我們成功導入外部的檔案了！

長這樣 ->
OwO



困難的方法...

感覺好像就沒那麼簡單了(棒讀



這個部份我先賣個關子好了，既然我們知道setIcon需要的是ImageIcon，理應上來說我們可以全部都用程式去解決，不需要去碰到Designer頁面。

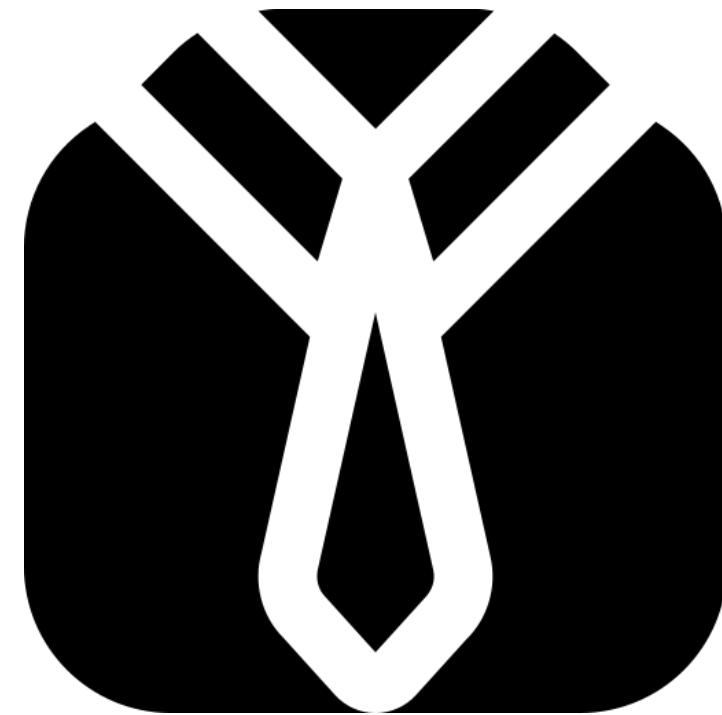
基於篇幅問題，我們會把程式作法放在附錄裡面，如果有多的時間我們再解釋。

(見附錄 3-2)

(可以查一下 ImageIcon 類別的建構式)

兄弟，衣服不合身

我的JLabel沒有圖片，在線等，急！



不要驚慌，這很正常

如果遇到了導入圖片後沒辦法正常顯示的情況，你可能小小的忽視了一個點：

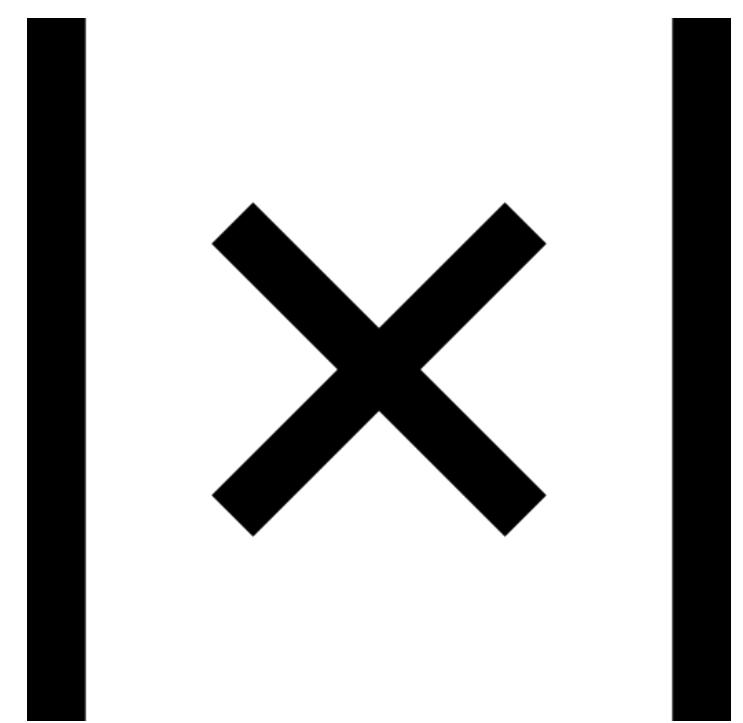
圖片的尺寸是否大於了你的標籤尺寸？

沒錯，這畢竟還是個視窗，基本的尺寸問題還是會出現的

出現了Out Of Bound的情況時，試著把標籤放大，或者把圖片縮小吧！

絕對不是必然

為什麼要避免絕對路徑的使用？



我們教的方法用的是相對路徑，但是為甚麼不用絕對路徑呢？

表面上來看，絕對路徑可以代表一個檔案的絕對位置，可以百分之百確定能夠找到想要的檔案，不過缺點也在其中

正因為絕對路徑使用的是本機的位址，只要這個程式離開了你的電腦就甚麼都不能動了，你設計好的圖片路徑也會全然失效

這一點在之後的打包上會是非常重要的，所以請銘記在心！





Developer Student Clubs
National Kaohsiung Normal University

6

更大的包容性

君君臣臣，父父子子

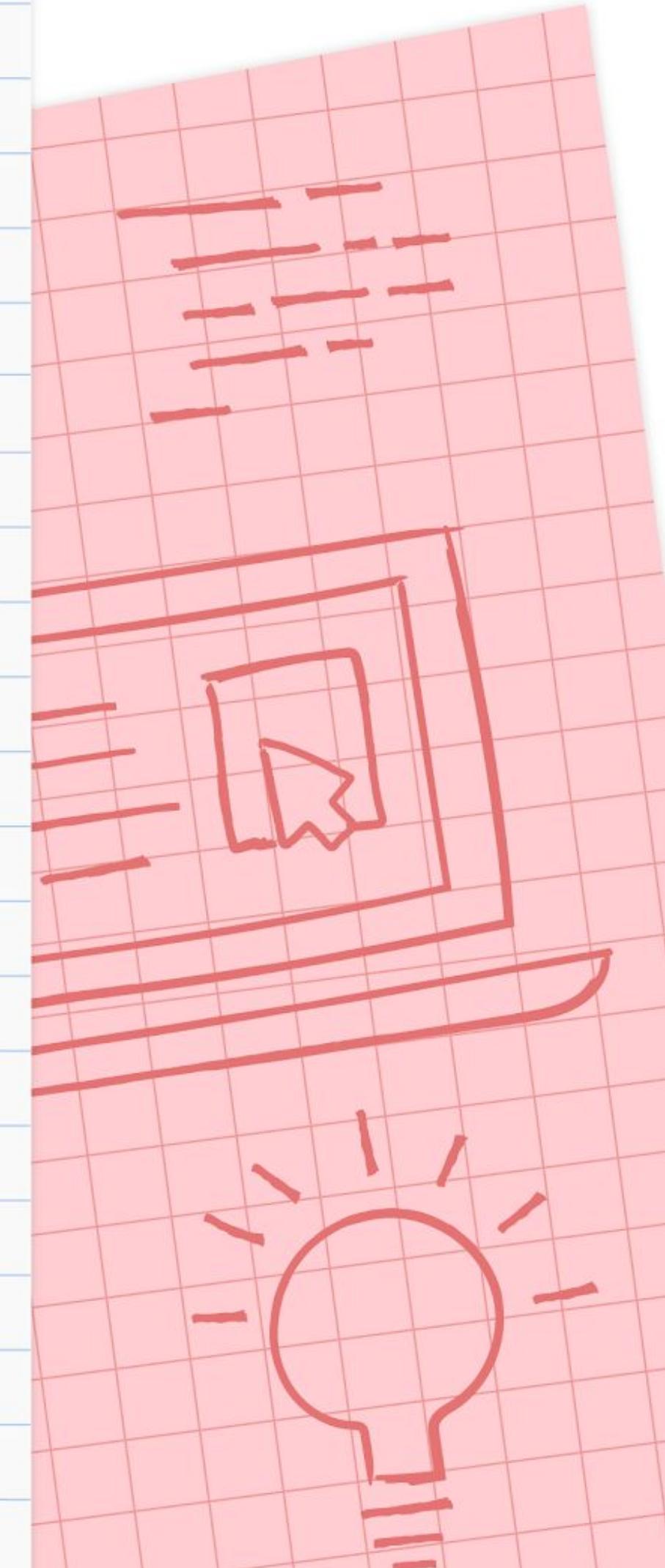
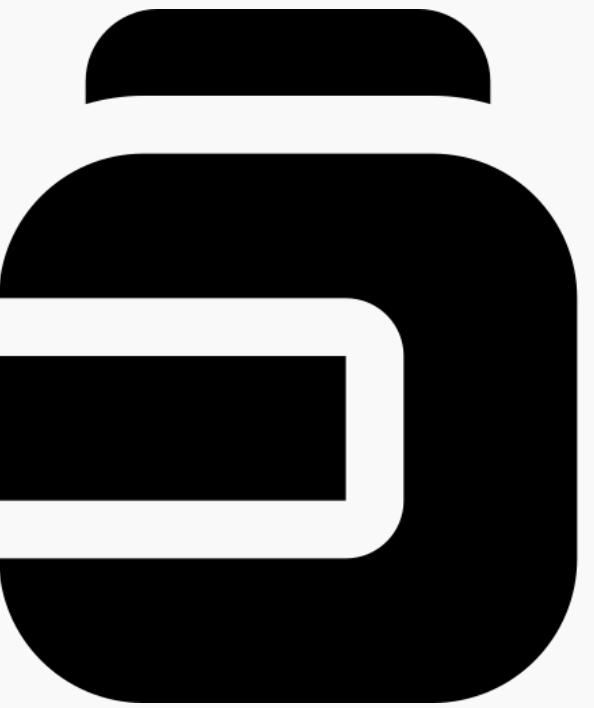
```
const filterByOrg = study => study.lead_organization === filterByOrg;
const filterStatus = filterByStatus ? study.status === filterByStatus : true;
const filterMatchStatus = filterMatchStatus ? study.match_status === filterMatchStatus : true;

function filterStudies({ studies, filterByOrg, filterByStatus, filterMatchStatus }) {
  return studies.filter(study => filterByOrg(study) && filterStatus(study) && filterMatchStatus(study));
}
```

回歸初心吧!

做了這麼久，我們一直都是在JFrame這個大平台上放置那些元件，追根究柢這個所謂的JFrame到底是甚麼樣的存在呢？

而在那之上，我們能還夠怎麼樣的去改善元件們的放置方式呢？



籃子跟更大的籃子

一種能夠包含其他元件的元件，我們稱呼其為容器



在我們的認知中，JFrame一直是畫布一般的存在，因為他的確就是視窗界面本身

只不過你知道嗎，他其實也是一種元件喔，由於他有能夠包含其他元件的特色，因此我們給了他一個更適合的稱呼：容器(Container)

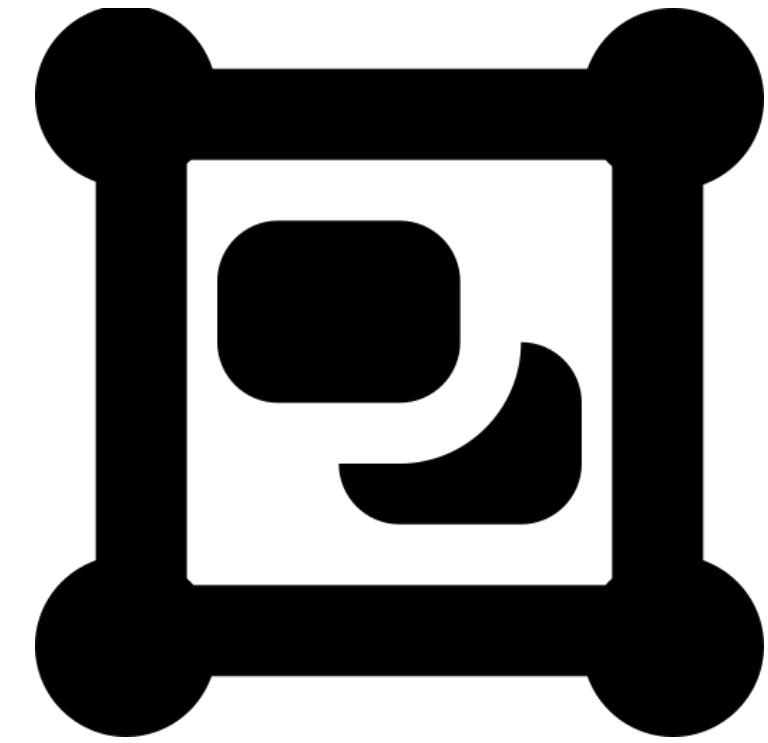
那Java Swing不可能只提供一種容器對吧
對，所以這一章接下來的部分我們會介紹一個東西，它叫做 JPanel

如果你是一個瘋狂的人，聽到JFrame的真面目也許會有大膽的想法
我們歡迎各種大膽的想法，但在這裡不多做說明



分組的必要性

容器存在的原因



JFrame的必要性是很好理解的，因為沒有視窗本身就不存在應用程式

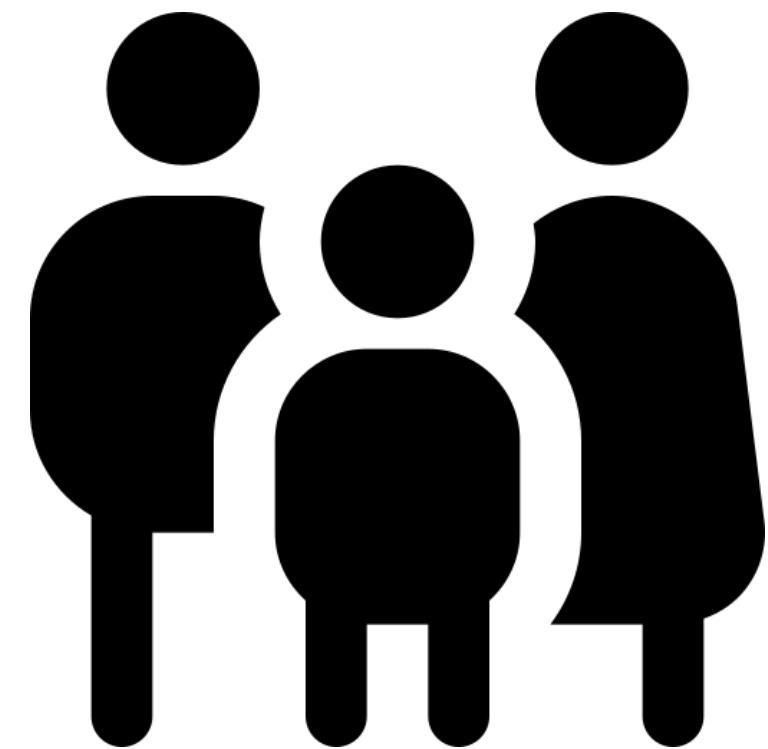
像JPanel這種中間容器卻又是為了什麼而被需要呢？

當然是為了把不同類型的元件做分組，讓他們能夠更好的被管理

- 想像JFrame是一個房子，它有牆壁、窗戶和門，可以容納家具
- JPanel則像是房子裡的房間，你可以在房間裡放置更多的家具，把不同的房間用來做不同的事情

拜見祖宗八代

我想知道元件的族譜



請找一下Designer介面當中的Component子頁面

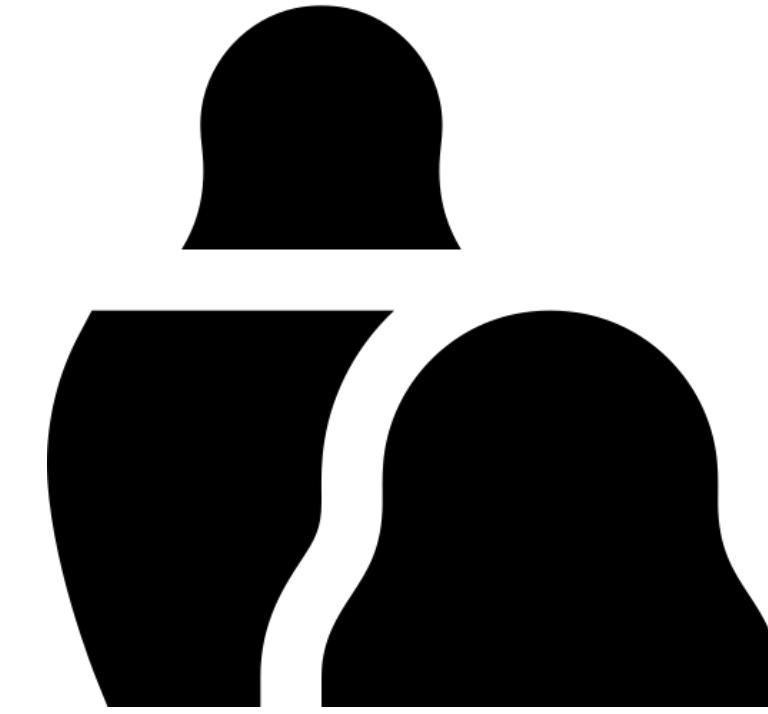
你會發現在目前視窗當中的所有元件都在一個叫做(javax.swing.JFrame)的層次底下，這代表所有的原件都共同擁有一個父容器: JFrame(也就是你當前的視窗)

稍等我們使用到JPanel時，他也將成為任何其中包含元件的父容器，沒錯

使用元件的getParent()方法可以讓你知道這個元件的父容器是什麼
(他會回傳一個Container)

俄羅斯套娃

所有東西都應該被放在它們應在的位置上



有件事你可能已經知道了，卻還是會覺得有些驚訝
 JPanel跟JFrame一樣都能去自訂容器內的佈局模式，還記得這東西嗎？

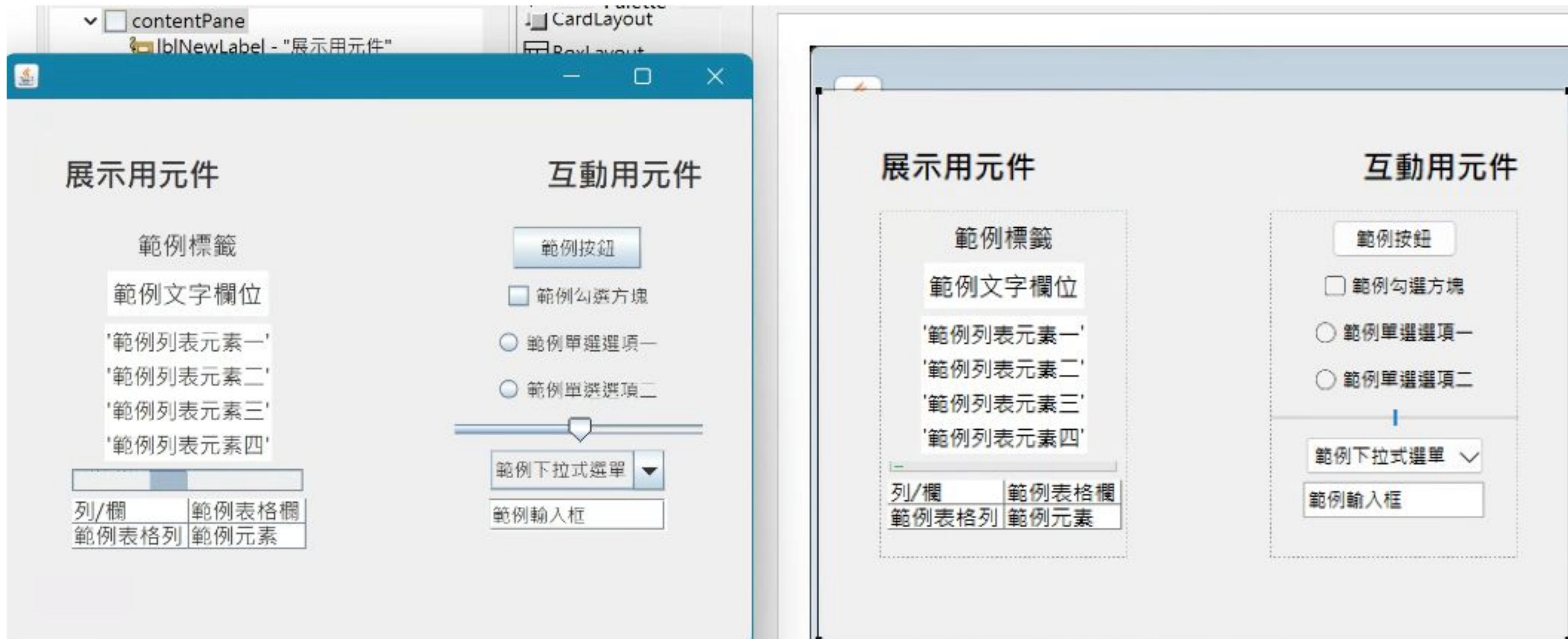
而由於父容器的概念可以抽象擴展，你可以在JPanel裡面放更多的JPanel，就像是你在房間裡面再開一個隔間的樣子

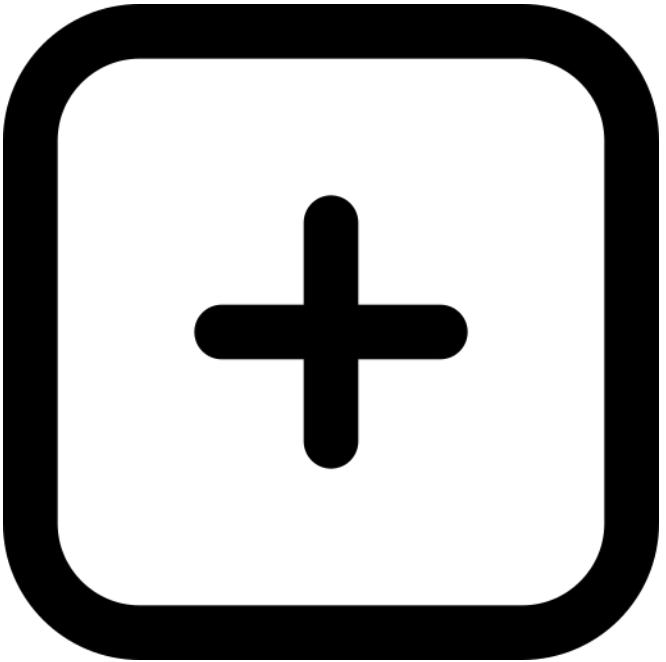
通常在設計上，我們會把展示用元件(Display)跟互動用元件(Interactive)給放在不同的隔間裡，這樣不管是看上去還是管理上都能優化許多

模擬實驗

我們來看看成效如何

很清楚地分出了兩個區塊 





點ㄟ滴滴

把元件塞進容器的方法

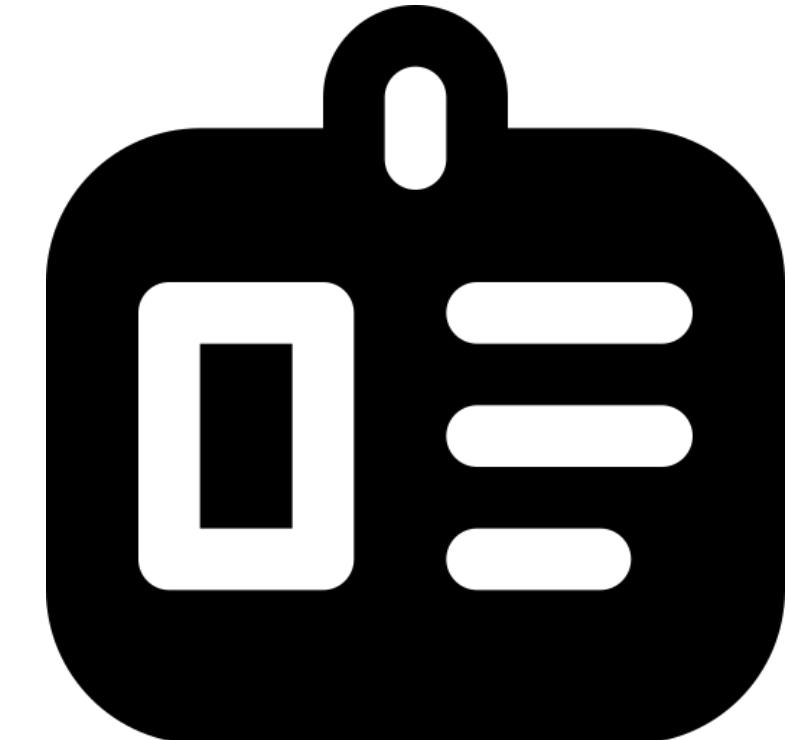
在Source頁面中，你可以發現在，我們將元件放入JPanel的動作以及將JPanel放入JFrame的動作就只是單純的把他給add進去，當然這會依照佈局模式而有所改變就是了。

```
JPanel display = new JPanel();
display.setBounds(39, 77, 159, 222);
contentPane.add(display);
```

```
JLabel exampleLabel = new JLabel("範例標籤");
exampleLabel.setFont(new Font("微軟正黑體", Font.PLAIN, 16));
display.add(exampleLabel);
```

別忘了取名字

命名很重要，真的



真沒想到我們直到這個時候才開始強調命名的重要性

當視窗中的元件開始多了起來，一般來說這個時候也出現了各種各樣的容器
在妥善分組的同時，別忘了花個一兩秒在左下角的屬性子頁面取個名字

容器也好，普通元件也好，你不會想要跟一大堆的Button1、Label2天人交戰的

為了自己也是為了其他看程式的人好，請為自己的元件命名
(然後不要取怪怪的名字!)



Developer Student Clubs
National Kaohsiung Normal University

7

就像一棵樹一樣

成為核心吧

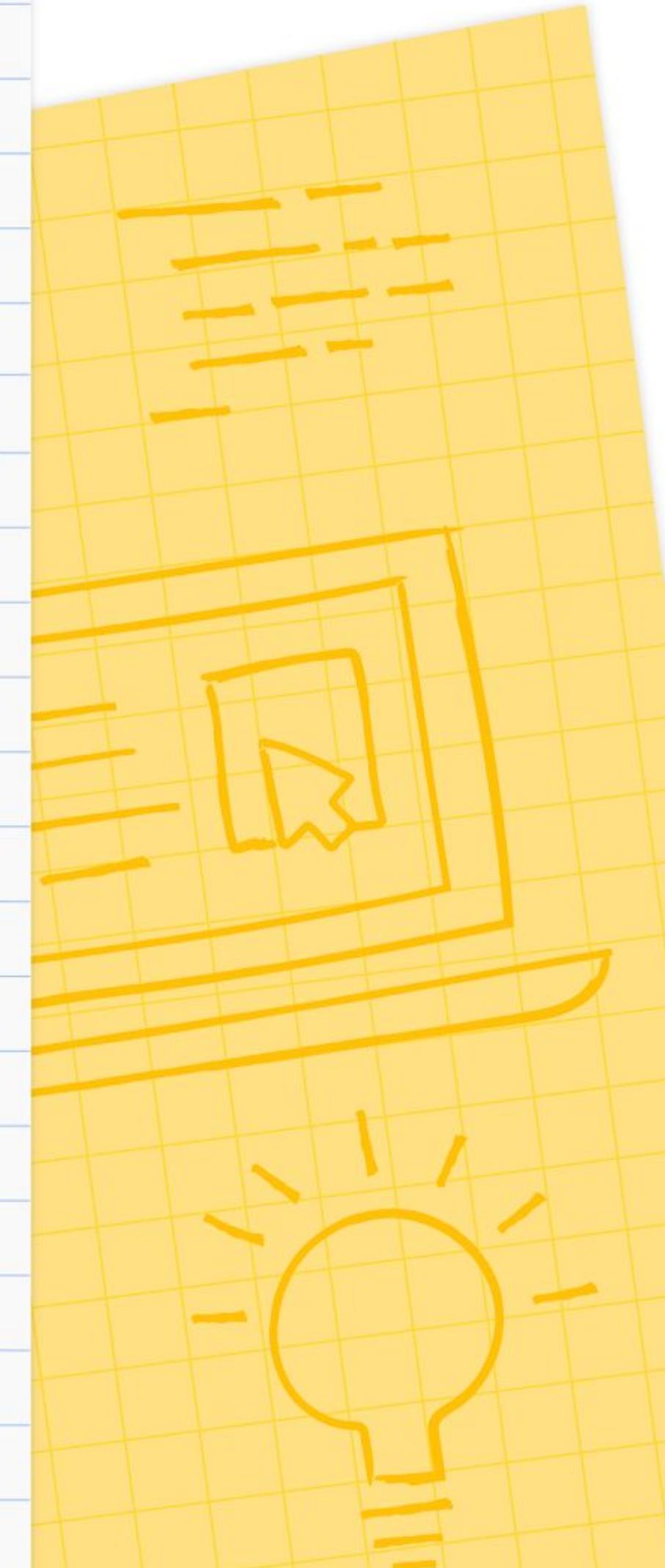
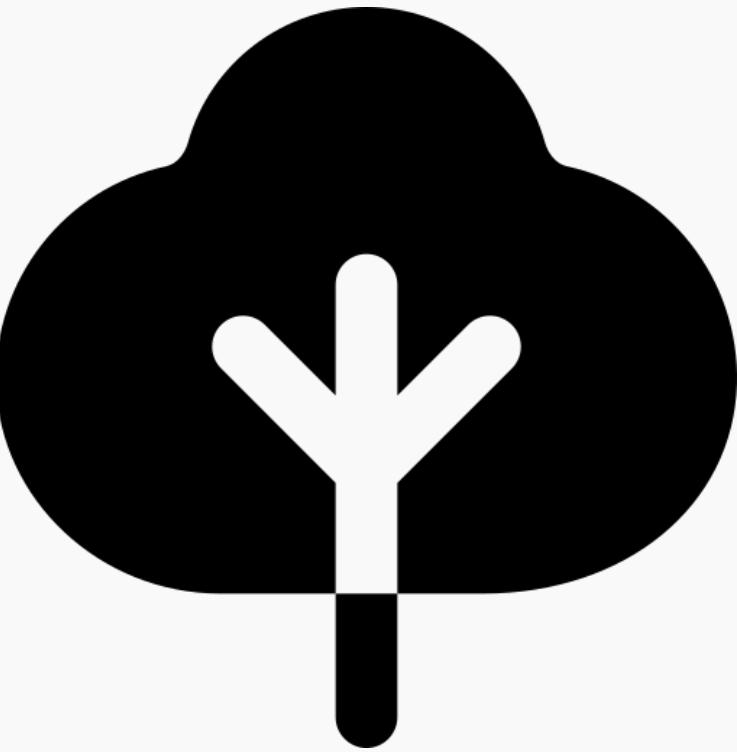
```
filterByOrg = filterByOrg ? study.lead_organization === filterByOrg : true
filterStatus = filterByStatus ? study.status === filterByStatus : true
const matchStatus = filterStatus ? study.status === filterStatus : true
```



種下種子吧

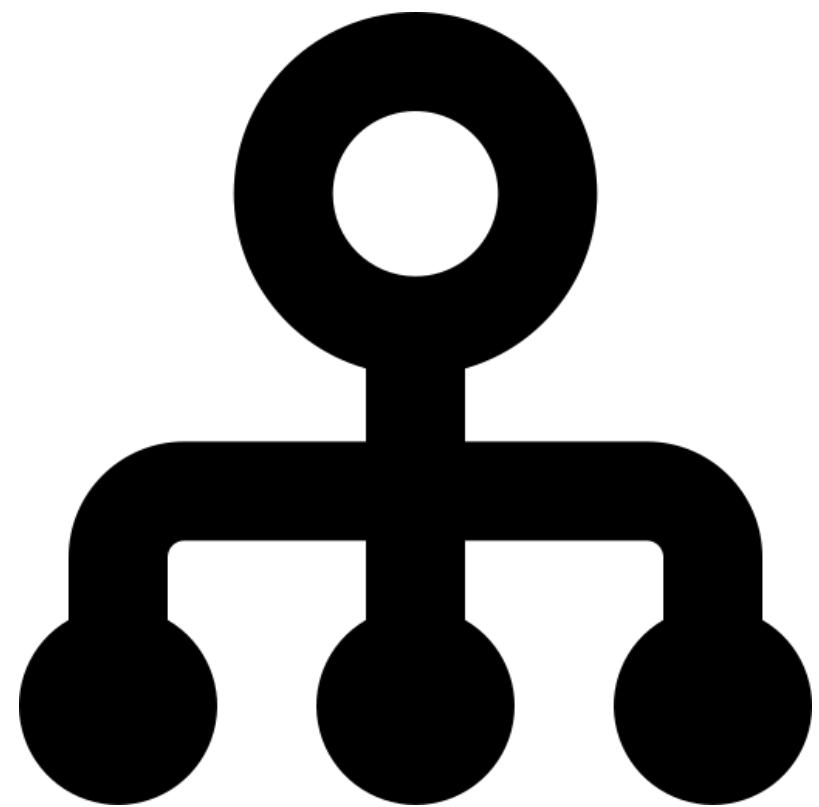
JPanel終究還是必須在一個JFrame中使用，是被當作房間一樣的容器，這並不夠

有沒有思考過彈出式視窗是怎麼運行的？他也是一種JFrame嗎？感覺真相已經很接近了...



一棵被稱為關係的樹

從主JFrame開始的無限擴張



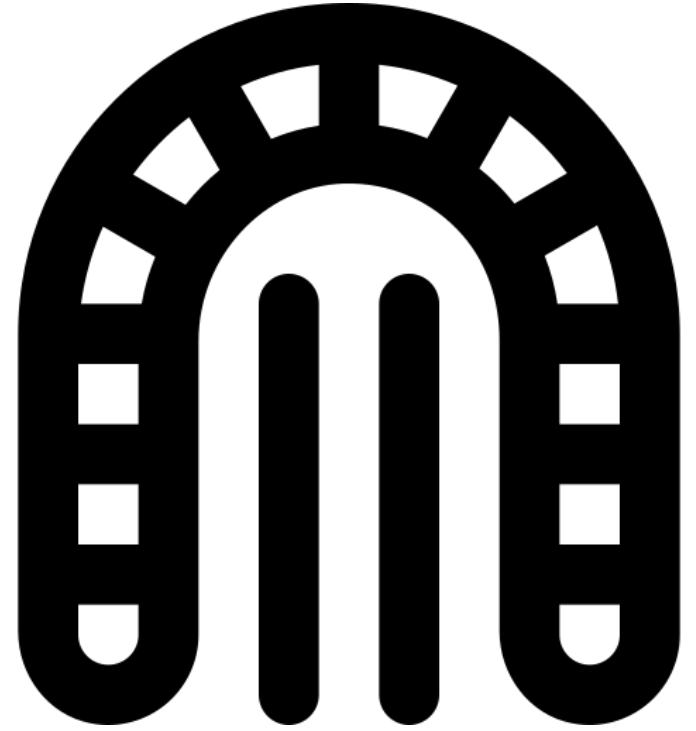
大家已經對容器的概念有些熟悉了，那我們再繼續把它擴展下去吧

主要JFrame在我們的程式中是一個根節點(root)，是我們界面層次中的頂層
隨之，繼續加入其他容器後，更多的節點被加入了，也出現了更多的層次

到最後，如果把整個相依關係給畫出來，你會發現它長得就像是一棵樹

沒錯，關係樹，這也就是整個視窗程式設計的關鍵
(父容器的對應是元件喔)





不被拘束的房間

車庫、花園、狗窩，這些房子之外的插件

進入到這一節的重點，到目前為止我們的畫布仍然侷限在一個JFrame裡面，儘管加了再多的 JPanel 也還是一樣

事實上是，容器種類有很多種，其中一個被稱作 JDialo g的小東西能夠顛覆你的想像，它與 JPanel 一樣是類似房間的概念，只不過是蓋在房子之外的房間

直接用視窗的方式來講，這東西允許使用者開啟主視窗之外的小視窗也就是我們所謂的彈出式視窗



線性與平行

平行線上沒有香蕉

JDialog有一個極為獨特的屬性，也就是所謂的「模態」(Modal)

隨著設定上模態的不同，子視窗的顯示也會有不一樣的運作方式：

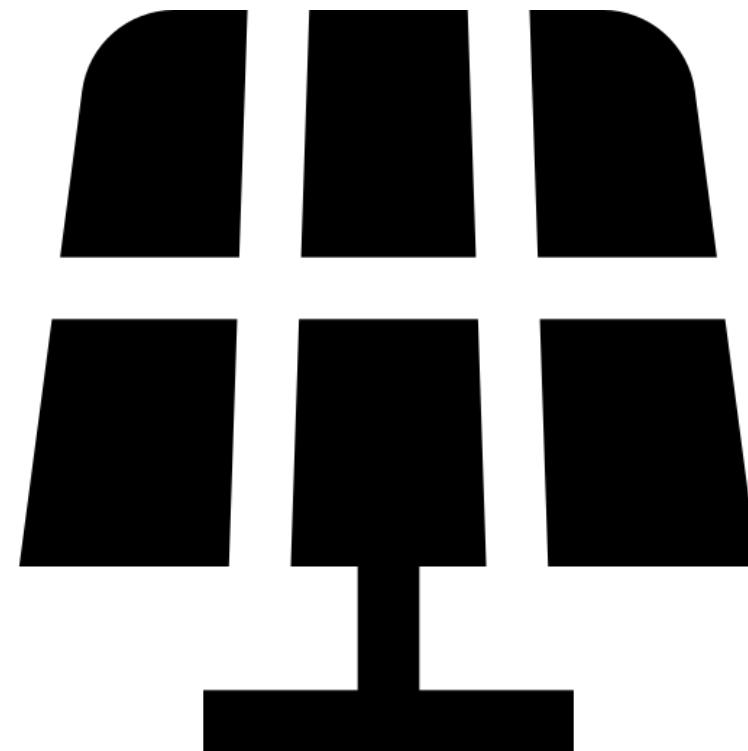
- 模態：主視窗必須等子視窗回應或關閉才能夠繼續運作
- 非模態：讓子視窗與主視窗可以平行處理

兩者在不同的情況下有獨特的用途



革命性的必要性

什麼時候需要子視窗？



曾幾何時有想過類似登入畫面的製作？

或者說像是顯示進度條、輸入對話框等等這些可能塞不下主視窗的部分

由於JDialog基本上就是一個小型的JFrame，你可以把它當作是你主視窗的擴展

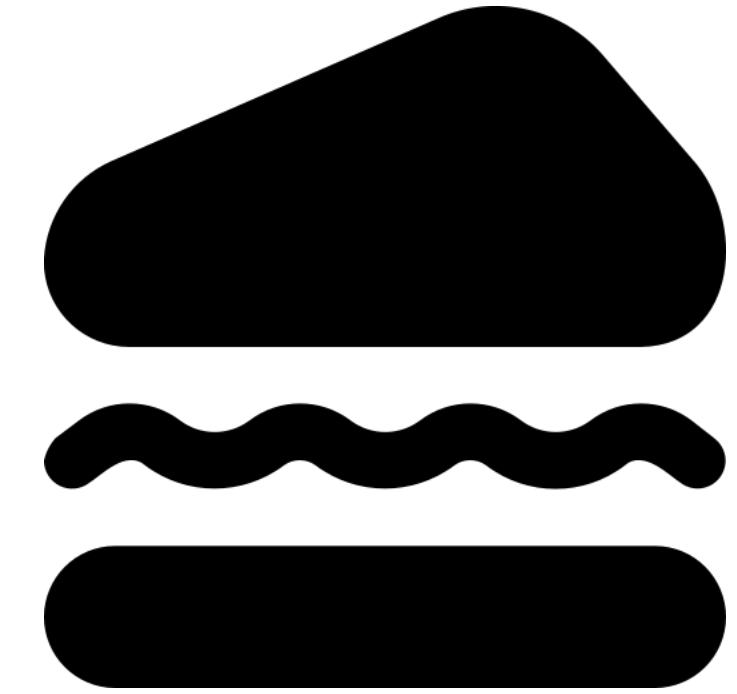
所以理所當然的，你也可以在裡面放置一些 JPanel 來擴展關係樹

但！你能夠在 JPanel 裡面放置 JDialog 嗎？！



總匯三明治

現在真的像一棵樹了



即使是容器也是有著階級之分的，房子終究還是最大的，你不能在房間裡塞下一整棟房子（那會變非歐幾里得？）

在Java Swing當中，容器可以像JFrame、JDialog一樣是屬於頂層，也有像是 JPanel、 JScrollPane（見附錄 3-3）這種中間容器

而JFrame始終獨一，屬於最高層次

到目前為止，我們已經把整體視窗設計可能會需要用到的功能都講解完成了



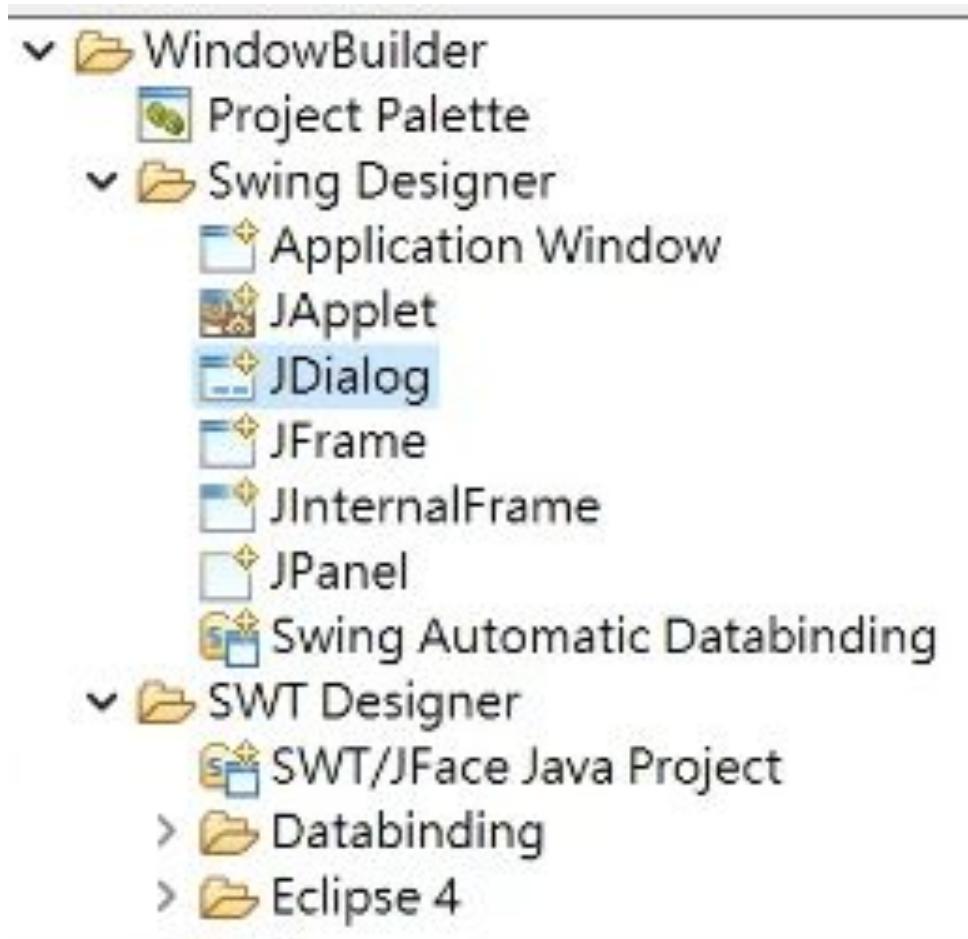
另一個程式檔案?!

這次，我們要跳脫JFrame的框架

你可能翻遍了整個工具箱都找不到一個叫做JDialog的元件？正是因為他是頂層容器，而不是能放在JFrame裡的小元件

我們往外面跳出去吧，到一開始創建JFrame的地方
在那裏你會找到JDialog的藏身之處~

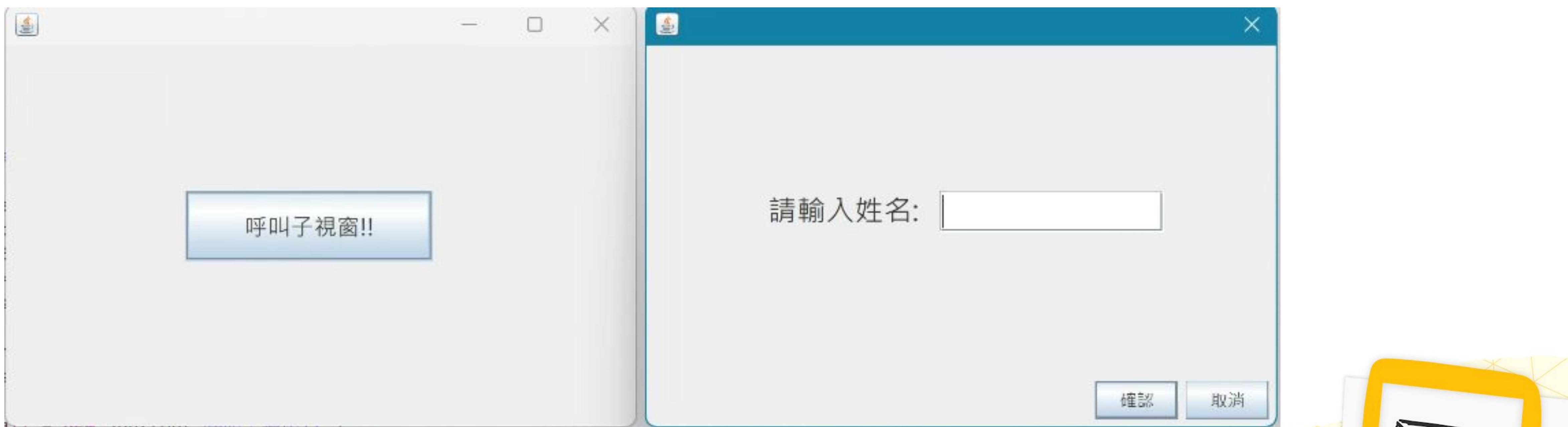
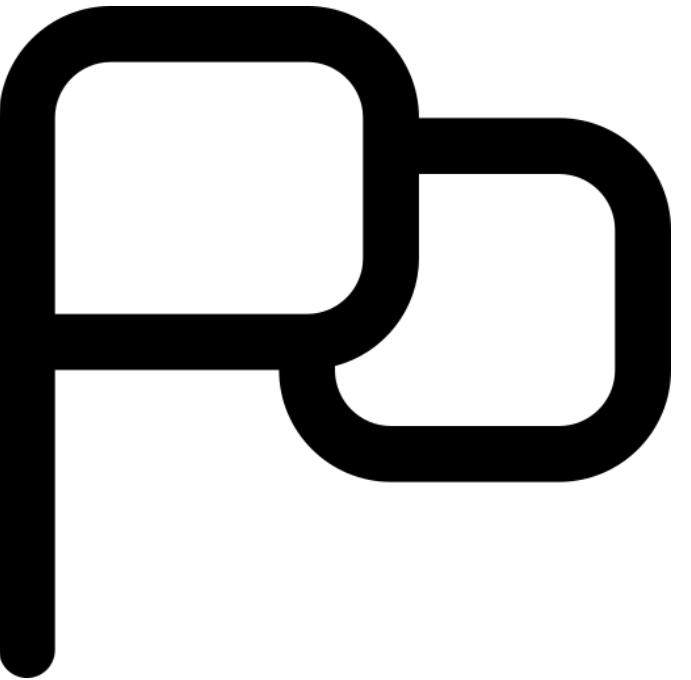
在這裡喔➤



看看我們走了多遠!

距離終點愈來愈近了

來去實作出大致的樣子吧!



是時候證明你的物件導向了

在這裡，沒有Designer的協助，我們需要你自己達成

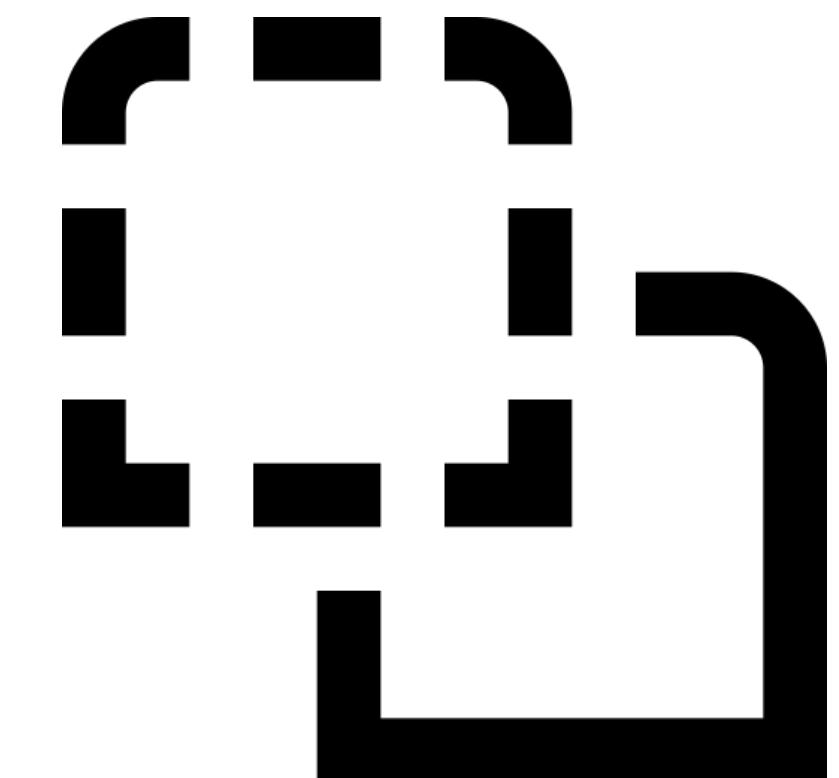
在我們新的檔案當中，你會注意到這一行

```
public class Modal extends JDialog
```

我們會需要在主視窗中的actionListener裡將其實作出來

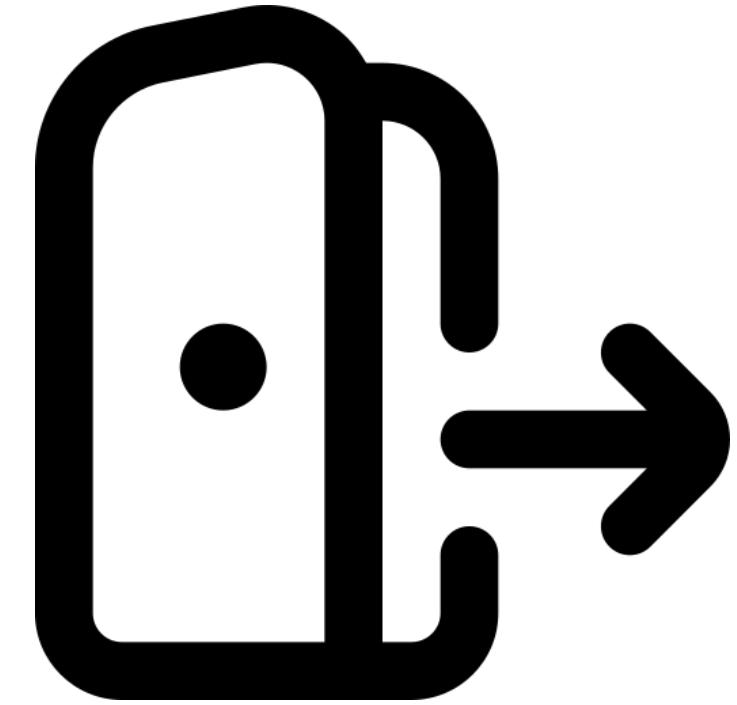
```
JButton btnCall = new JButton("呼叫子視窗!!");  
btnCall.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        Modal m = new Modal();  
        m.setVisible(true);  
    }  
});
```

← 然後要讓他能被看得見



最難的一步

學會怎麼dispose(), 然後你就正式畢業了



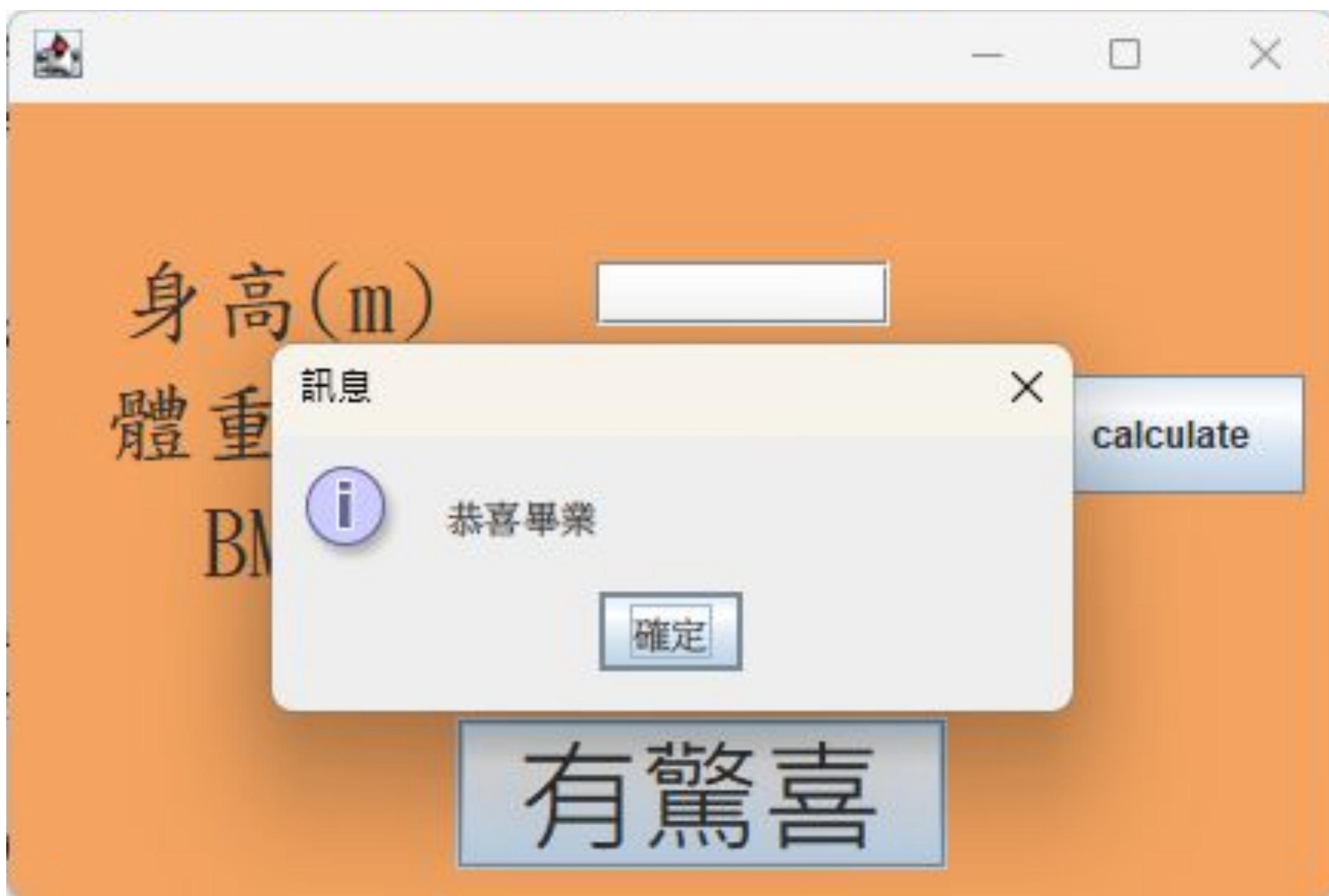
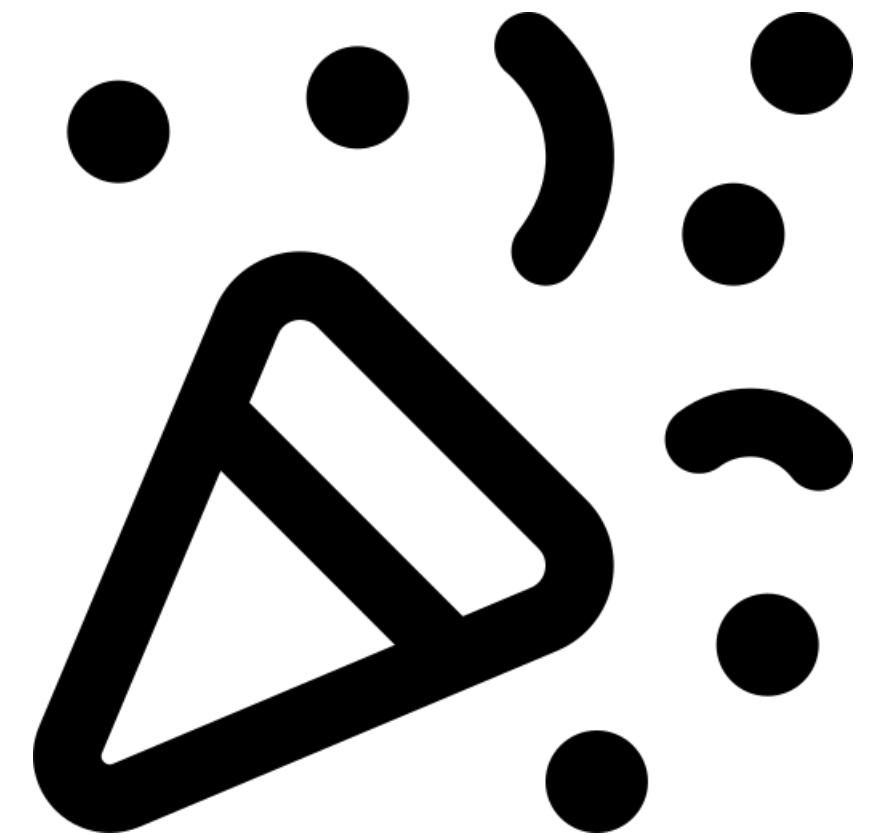
這次是在JDialog裡，我們要能夠把他給關起來(除了右上角的X以外)

```
JButton okButton = new JButton("確認");
okButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String input = txtInput.getText();
        JOptionPane.showMessageDialog(Modal.this, "輸入的姓名是: " + input, "確認", JOptionPane.INFORMATION_MESSAGE);
        dispose(); // 關閉對話框
    }
});
```

於是在這裡的按鈕裡，我們最後給他一個dispose()方法，讓他會自我毀滅
(JOptionPane的部分可以直接用System.out.print()取代，因為只是拿來演示)

故天將降大任於斯人也

你已經完成了視窗設計的課程，恭喜！





Developer Student Clubs
National Kaohsiung Normal University

8

美麗的視窗程式設計

是時候打包帶走了

```
const filterByOrg = study => study.lead_organization === filterByOrg;
const filterByStatus = filterByStatus ? study.status === filterByStatus : true;
const filterStudies = studies.filter(study => filterByOrg(study) && filterByStatus);
```





所有人都期待著這一刻

欸，我現在是畢業了嗎？

但是我的視窗還在這個叫做Eclipse的東西
裡欸？

好啊，那我們把它給打包匯出吧。

等等，就這麼簡單？



檔案，啟動

到底有幾種啊???



大家常常聽到的應用程式執行檔不外乎就是.exe (Executable Files)或者.msi (Microsoft Installer Files)等等

這種檔案不需要依賴特殊環境，只要是在Window作業系統當中都可以運行

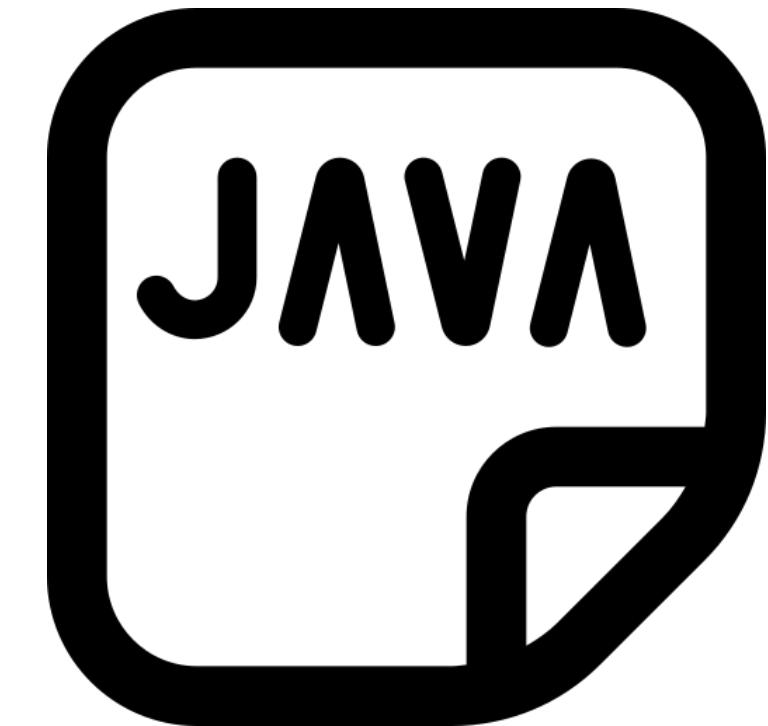
與其相對的.dmg檔(MacOS上的)、.appimage檔(Linux上的)等等也是屬於其作業系統上的獨特執行檔

我們接下來要講的是並非依賴於作業系統上，而是JRE上的執行/壓縮檔
--Jar檔案



J的家族

有時候是JDK, 有時候是JRE



在進到打包的步驟之前，我們先來介紹一下Java的獨特之處：

基於它是一個跨平台、物件導向的泛型設計程式系統，能夠在業界屹立不搖數十年也是很好理解的

我們主要使用的服務大致為兩種：JDK (Java Development Kits) 和 JRE (Java Runtime Environment)

前者是用來開發的套件，而後者則是Java檔案的執行環境

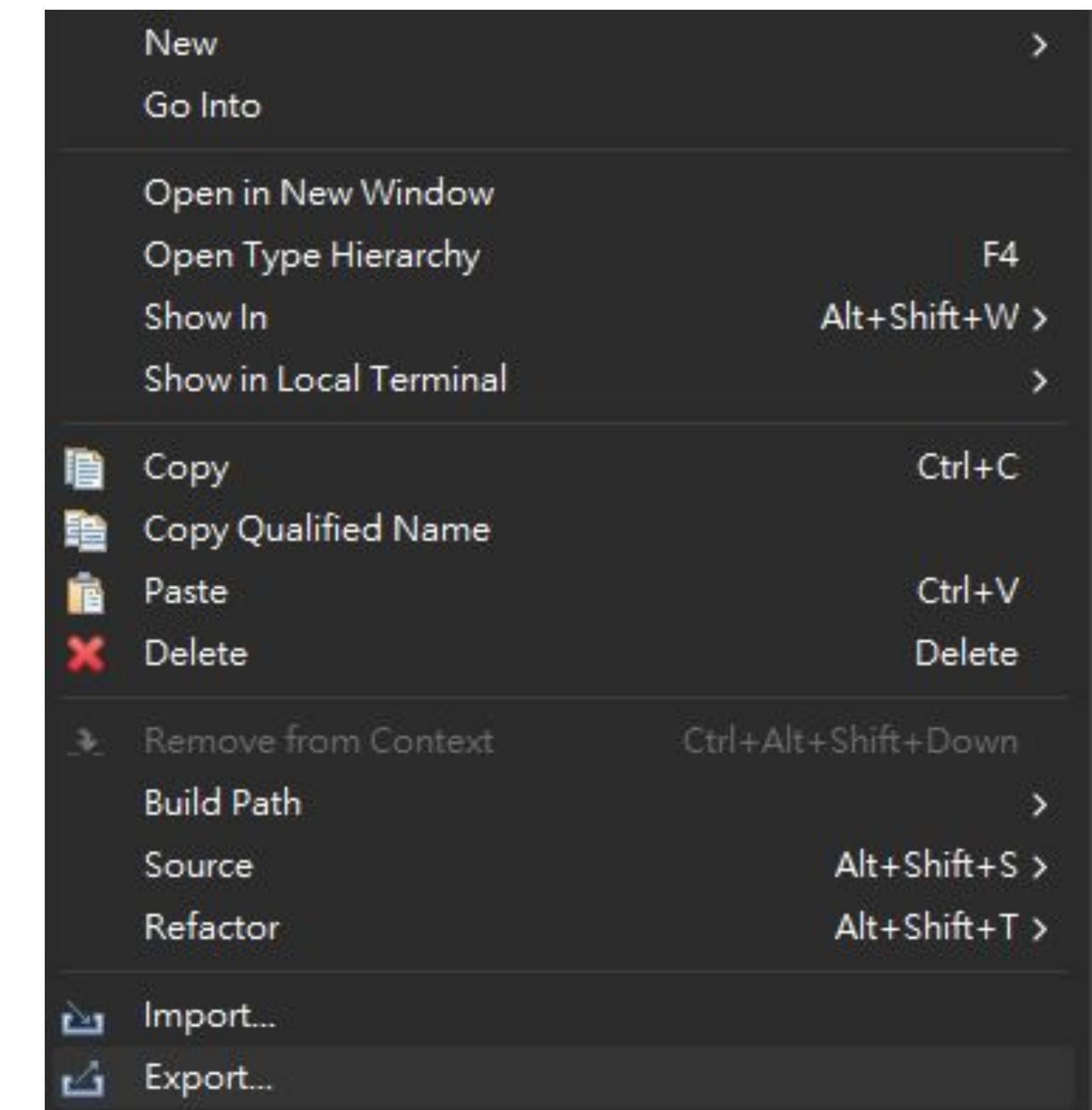
裝起來，帶走

就很直接的樣子

所幸，在我們使用的Eclipse編譯器中，導出執行文件的功能已經被很好的嵌入在其中了

與其使用原始的檔案打包程序(見附錄 3-4)，我們可以單純的動動手指就自動完成了

第一步，先從專案找到這個小小的匯出按鈕



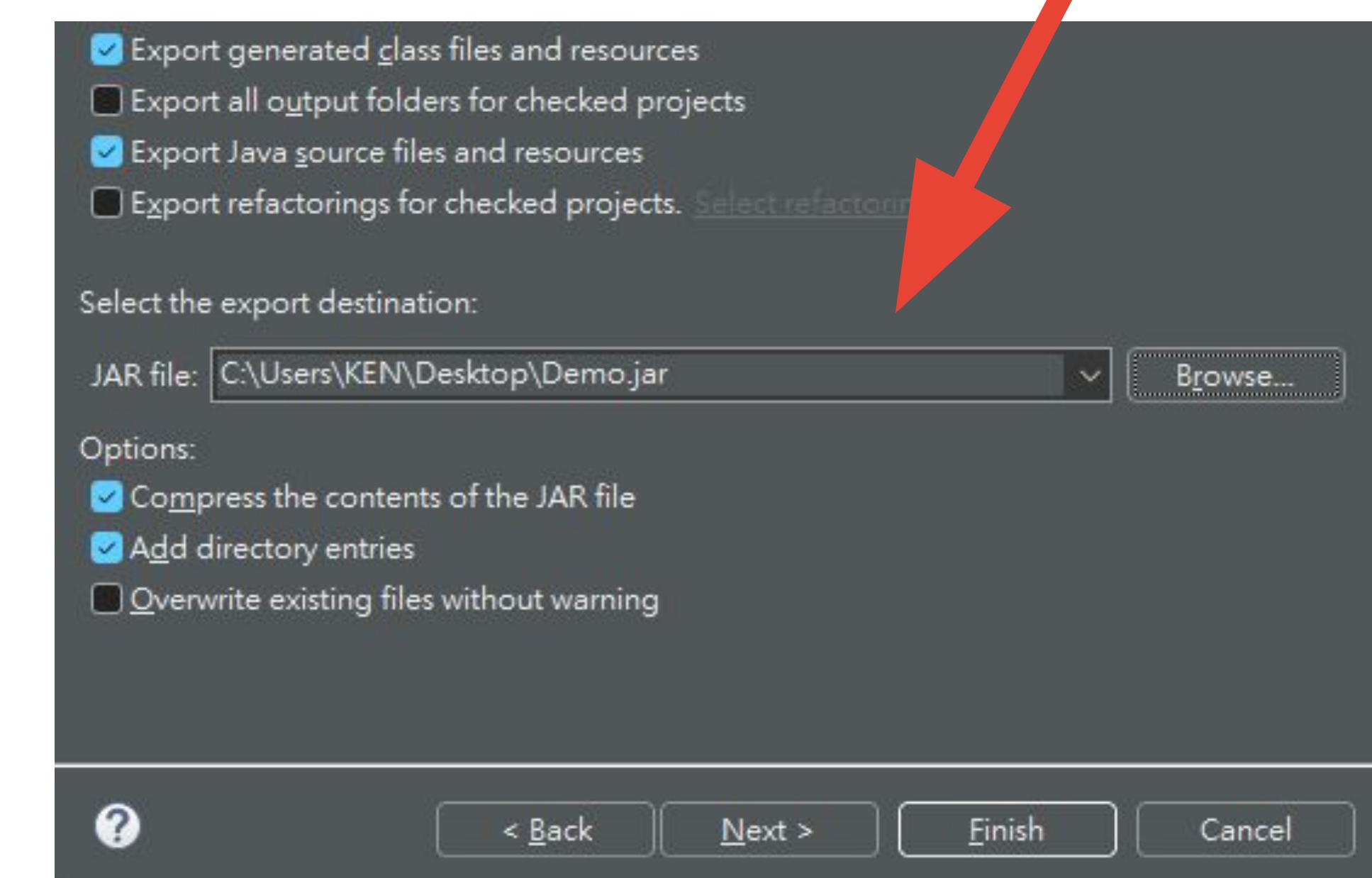
給他一個適合的家

告訴程式要去哪裡

就像儲存所有其他文件一樣，我們必須告知它的匯出路徑

除此之外，還要順便把要帶的行李也都準備好

基本上有很多它的預設選項都是沒有必要的，我們只保留我們的核心檔案，然後就是下一步



具現化(聽起來好帥)

應用程式的排查清單

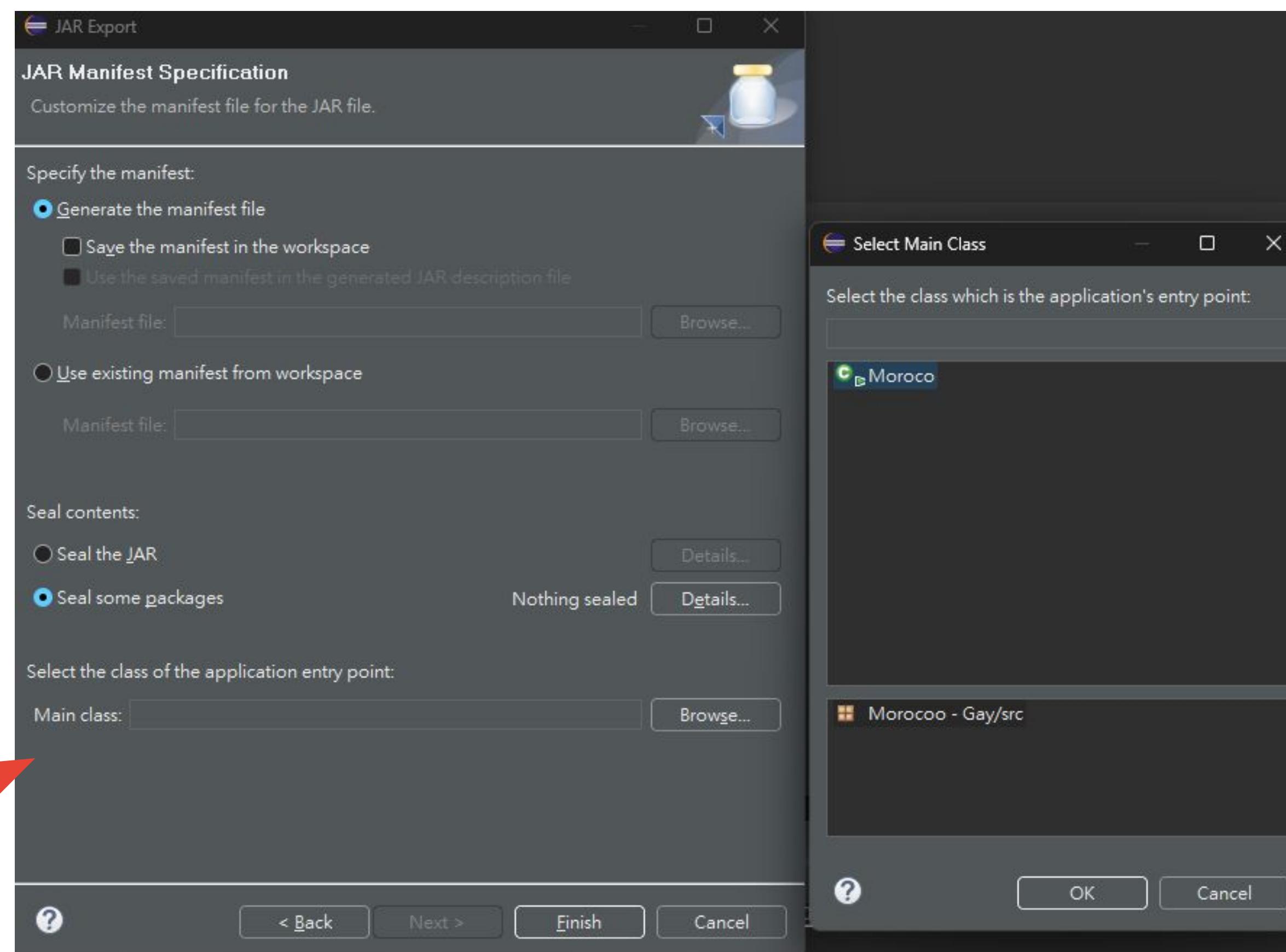
JAR (Java Archive) 檔就像是一個完整的公司，在向外展示的時候需要確保所有的人員跟程序都是正常運作的

正因如此，所謂的Manifest檔案就被發明了

這種神奇的檔案能夠確保程式的入口點，讓執行的時候不至於出錯

(不過你還是要自己去指定)

在Eclipse裡，你可以選擇要讓它自動創建還是引用外部



是風的追求，還是樹的不挽留？



我們的視窗就此成為了執行檔

在短短三頁的簡報裡我們就成功把做了三節課的視窗應用程式給匯出來了

現在，只要你的電腦已經裝好對應的JRE環境，那麼就與其它執行檔一樣點個兩下就可以開啟了

如果出現點擊打不開或者報錯的情況，也許是環境與Eclipse裡的版本不同，也有可能是在最後一步時沒有指定好MainClass

就此，我們的程式終於可以獨立於編譯器，成為一個真正的應用程式了！

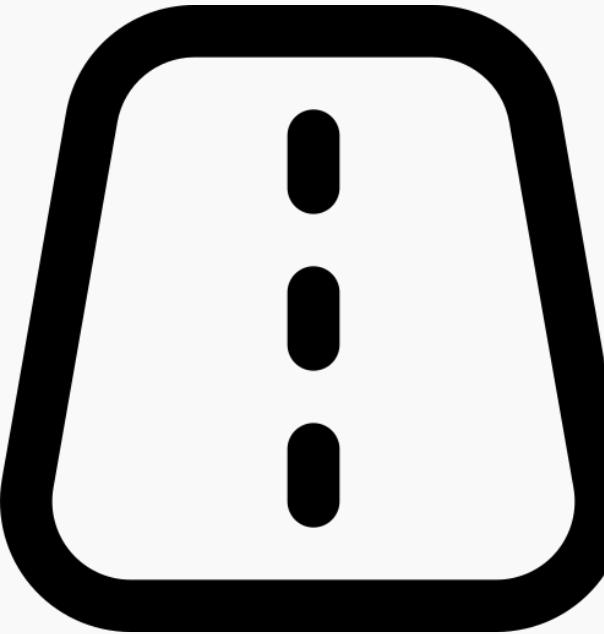
感覺走了不少的路

前面仍然無限延伸著

回顧一下到目前為止我們都學到了甚麼吧!

在第一堂課中，我們學到了視窗設計的基本概念
隨即，我們學到了怎麼樣用程式賦予元件們靈魂
接著在第三節課，我們又延伸出了很多概念

直至現在，一個完整的系列即將迎來他的尾聲！





Developer Student Clubs
National Kaohsiung Normal University

9

所以，然後呢？

未來的道路要怎麼走？

```
filterByOrg = filterByOrg ? study.team_organization === filterByOrg : true
filterStatus = filterByStatus ? study.status === filterByStatus : true
const matchStatus = filterStatus ? study.status === filterStatus : true
```

```
function filterStudies({ studies, filterByOrg = false, filterByStatus = false }) {
  return studies.filter(study => filterByOrg || !filterByOrg || study.team_organization === filterByOrg) .filter(study =>
```

履歷表+1

學會視窗設計了，接下來可以做甚麼？



實際上，我們學的所有東西都屬於前端開發的範圍，也就是所謂的UI/UX

這條路可謂無遠弗屆阿



接下來是延伸知識時間

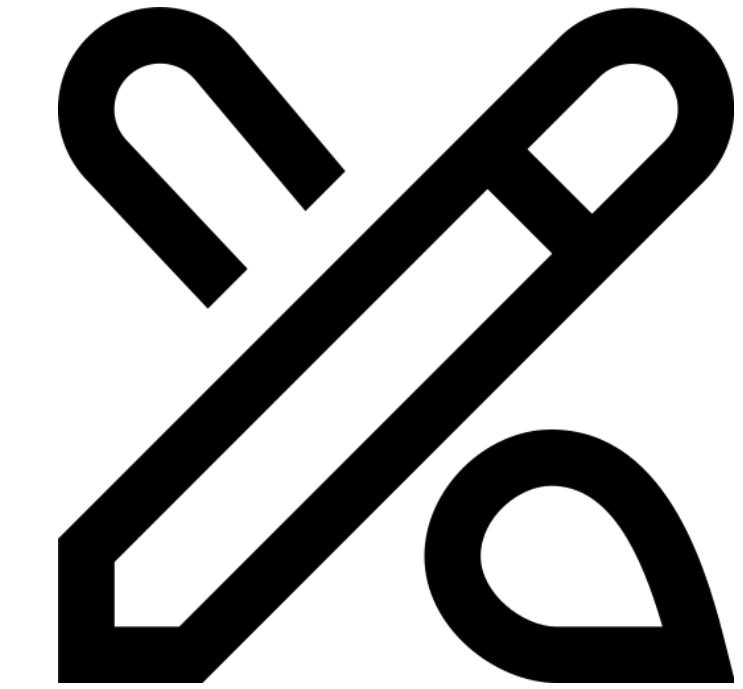
希望可以多少聽一下

哇！廢話時間！

```
function filterStudies({ studies, filterByOrg = false, filterByCategory = false }) {
  return studies.filter(study => {
    if (filterByCategory) {
      return study.categories.some(category => category.id === filterByCategory)
    }
    if (filterByOrg) {
      return study.organizations.some(organization => organization.id === filterByOrg)
    }
    return true
  })
}
```

主題一：設計原則

並不強迫，不過在目前的業界是共識



在進行視窗設計的時候應該要注意以下幾個點，以免出現太過雜亂、使用者不知道要從哪裡開始使用等等

- **一致性:** 保持界面設計的一致性，確保用戶在不同的窗口和操作之間有統一的體驗。
- **簡潔性:** 界面設計應簡單明瞭，不要讓用戶感到負擔。
- **響應性:** 確保界面對用戶的操作能夠迅速且正確地響應，避免用戶等待過久。
- **可用性:** 設計界面時考慮用戶的使用情況，確保界面易於使用和理解。

主題二：其他設計工具

當然有很多更多的作法

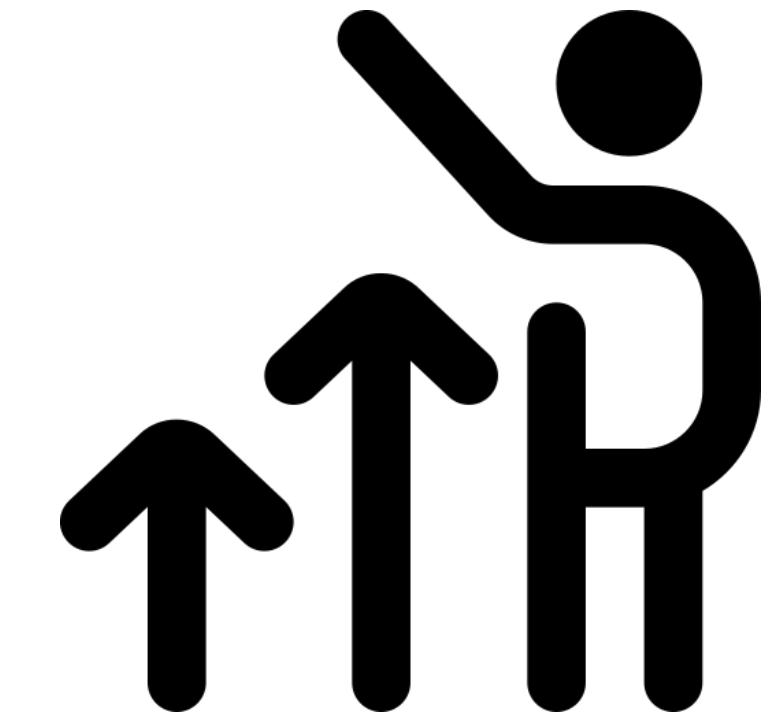


Java Swing並非業界獨佔，在外面還有很多的開發方法

- **WinForm:** 微軟自家的Visual Studio可以使用C#開發視窗，其設計界面就跟WindowsBuilder差不多，不過多出了很多功能，並且與系統相容性很高
 - 缺點大概是.NET環境只適用於Windows作業系統
- **TKinter:** 屬於Python的最原始的開發工具，沒有設計界面，但也是最多人第一次嘗試使用的工具，可以再擴張與wxPython、PyQt、PyGTK等等做結合，歸功於Python是一個很廣泛的程式語言。

主題三：其他前端開發領域

作為稱職的前端工程師，視窗開發可不是唯一技能



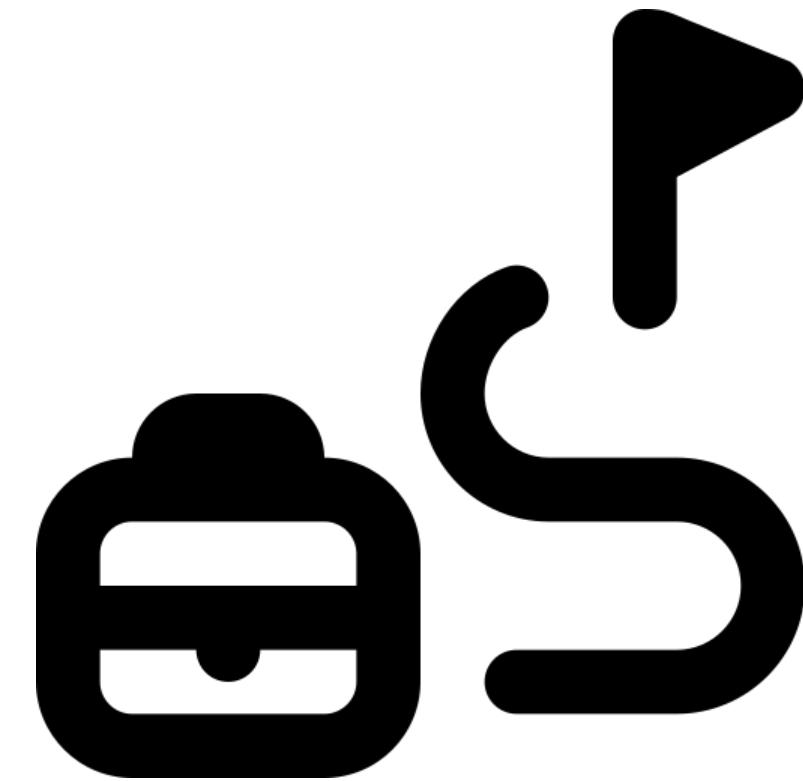
有沒有想過除了淺顯易懂的視窗開發設計以外，還有甚麼其他的領域是屬於再前端的範疇之內？其實意外的很多喔！

- **Web開發**: HTML、CSS和JavaScript等等能夠做到的動態響應式網頁開發
- **移動應用開發**: React、Flutter和Swift的iOS和Android應用程式開發
- **UX設計**: 與我們所做的UI不同，注重於原型設計、可用性測試等等的技術屬於UX設計

補充: UI (User Interface) 和 UX (User Experience) 是不同卻息息相關的兩點喔

主題四：進階技巧

在GUI設計上的進階主題

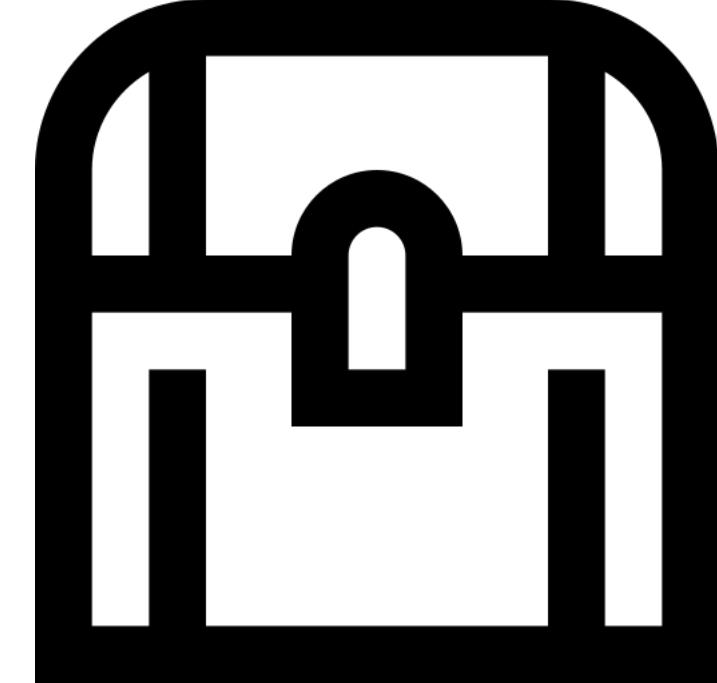


除了我們教過的元件、容器等等的概念，你們也可以去多多觀察一下目前大家常用的應用程式還具備了甚麼技術，以下是幾點很常見到的：

- **動畫效果**: 單純的出現與消失讓人提不起興趣，有動態感覺不是比較好玩嗎？
- **本地化與國際化**: 提供多語系支援，讓你的客群可以擴展到更多的地區
- **可訪問性設計**: 提供鍵盤導航、色盲模式等功能，讓所有用戶都能夠輕易使用
- **設計模式(見附錄 3-5)**: 使用MVC、MVP、MVVM等等架構去讓整體介面更加完整

真正的寶藏

你期許的成果將由自己的手創造



到這裡，我們已經沒有甚麼東西可以講了，所以這也等於這個系列課程結束了

無論是Eclipse的使用也好，程式設計的實作也好，我們試著盡可能地將課程設計的平易近人，以詼諧逗趣的方式去進行教學

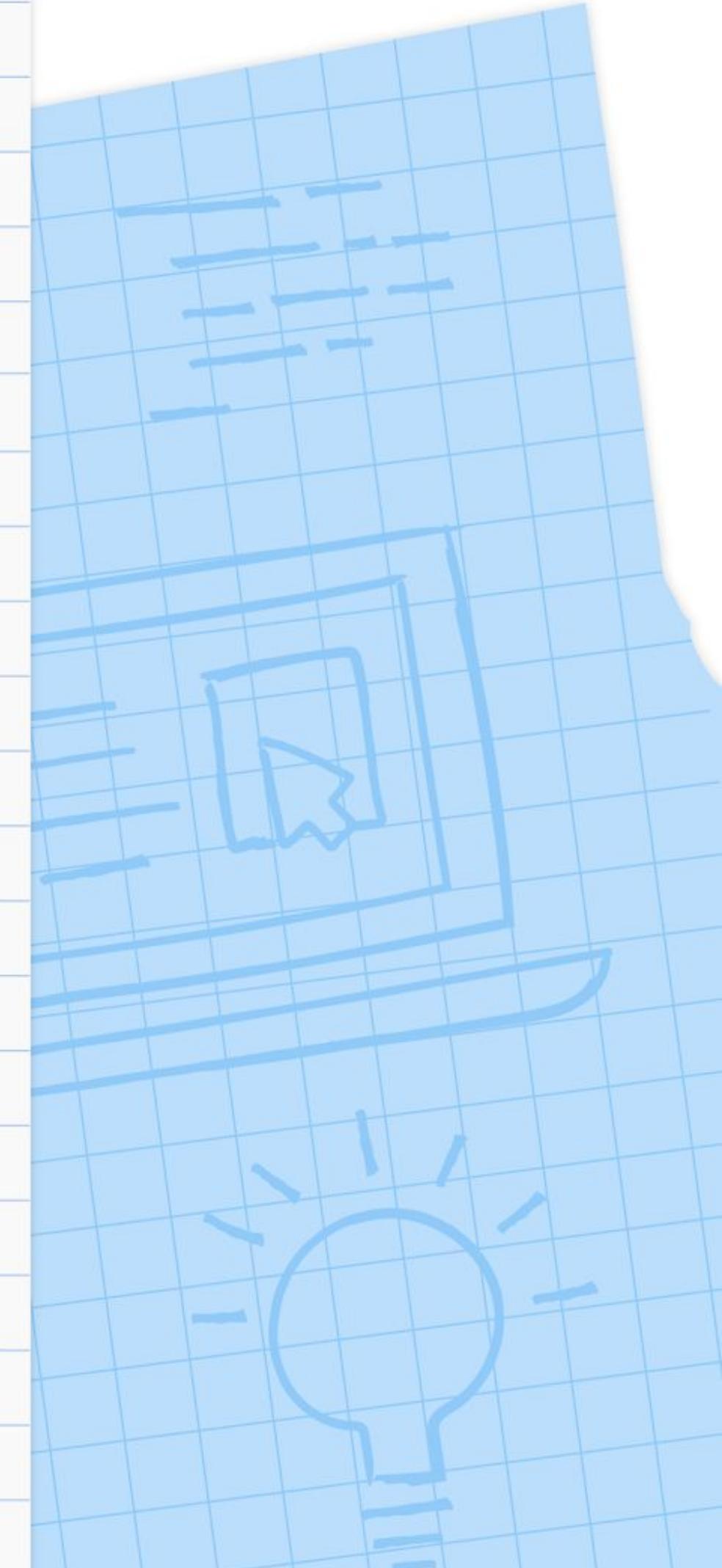
目的單純只是希望聽眾們真的有學到一些重要的知識

在前端的部分，你們已經不是外行人了
光是這一點就代表我們的課程是真正有意義的吧

凱奇會了!

有了視窗設計的基礎技能，他之後就可以做出一些實用且美觀好看的应用程式了！

現在他只需要研讀計算機網路，認識一些演算法並且學會資料庫的管理就可以成為軟體工程師了！~~(真容易)~~



就醬了~

真的真的感謝你們的參與!

有任何其他問題可以到我們的DC群詢問，也可以直接寄件給講師們！

Bernie: ptyc4076@gmail.com

吳稼澂: charlie930320@gmail.com



Developer Student Clubs
National Kaohsiung Normal University

附錄1：元件大家族

跟大師兄問聲好啊！

```
choic = filterByOrg ? study.lead_organization === filterByOrg : true;
ifStatus = filterByStatus ? study.status === filterByStatus : true;
ifMatchStatus) {
    return study;
}
}

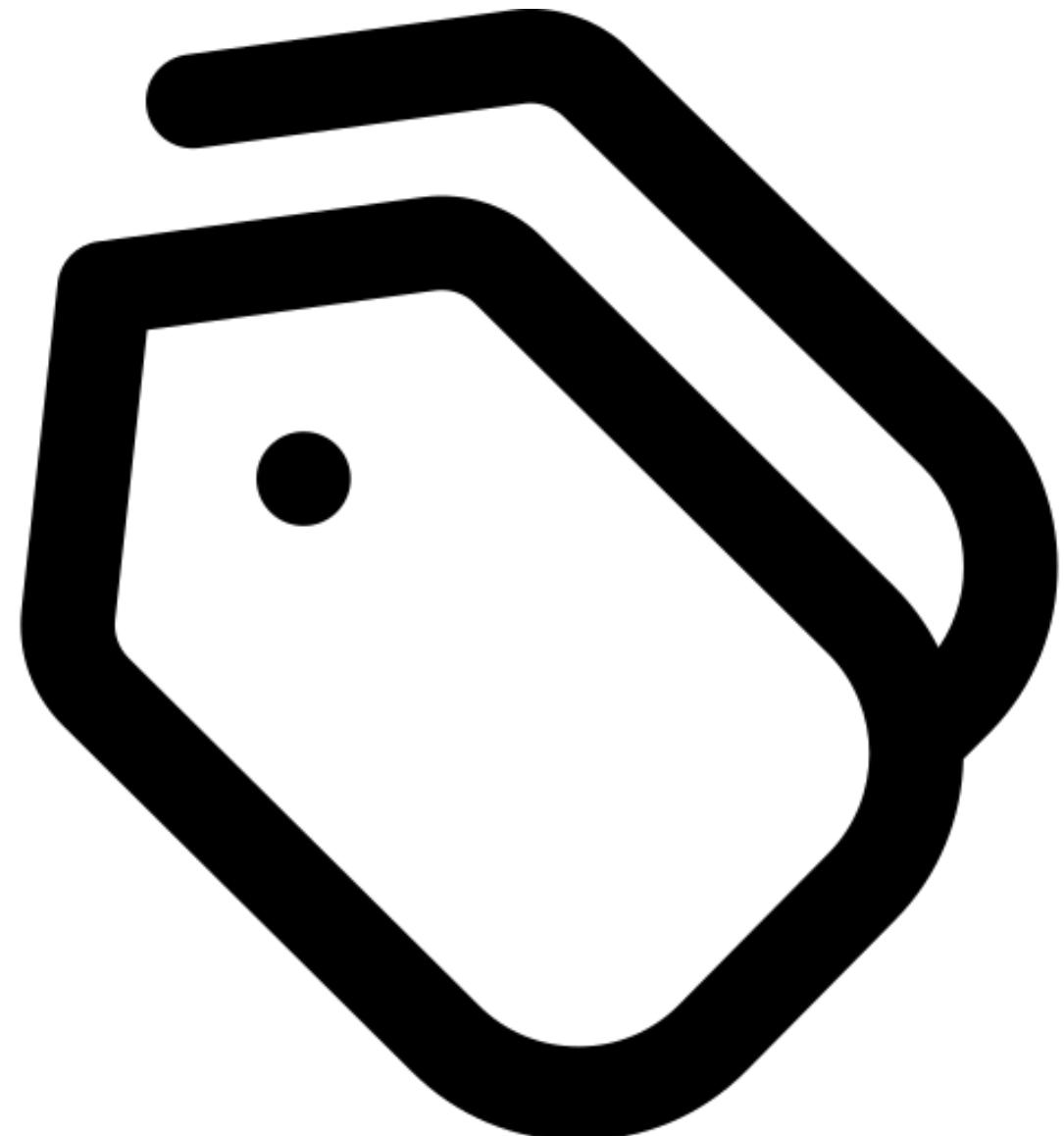
function filterStudies({ studies, filterByOrg =
filterByStatus }) {
    return studies.filter(study => {
        if (filterByOrg) {
            return study.lead_organization === filterByOrg;
        }
        if (filterByStatus) {
            return study.status === filterByStatus;
        }
        return true;
    });
}
```



Label 標籤

最直接的顯示方式

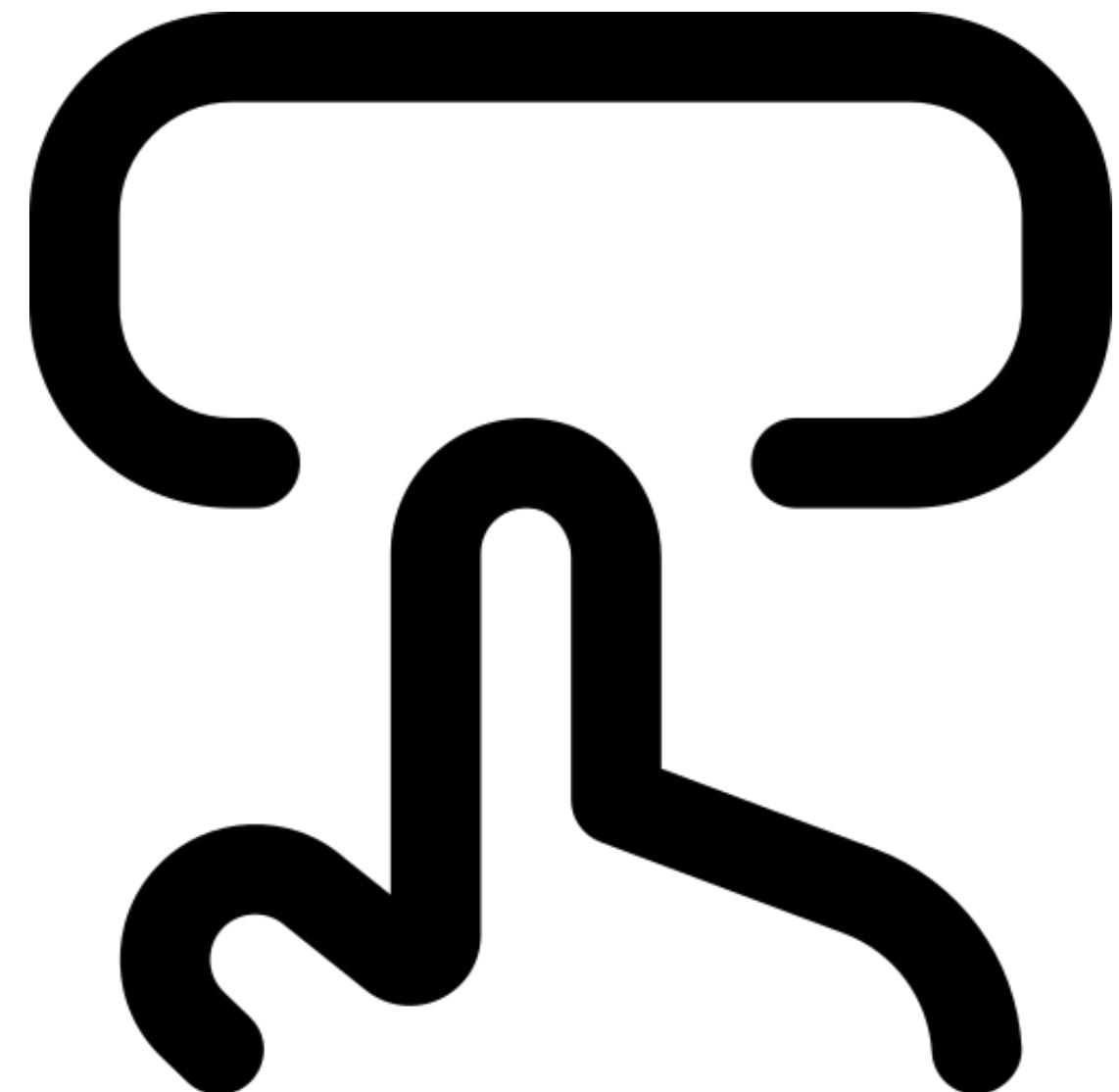
- 用於顯示文字或圖像。
- 靜態且非互動式。
- 向使用者提供資訊或上下文。
- 通常用於標題、說明文字或說明。



Button 按鈕

按下, 彈起, 好好玩

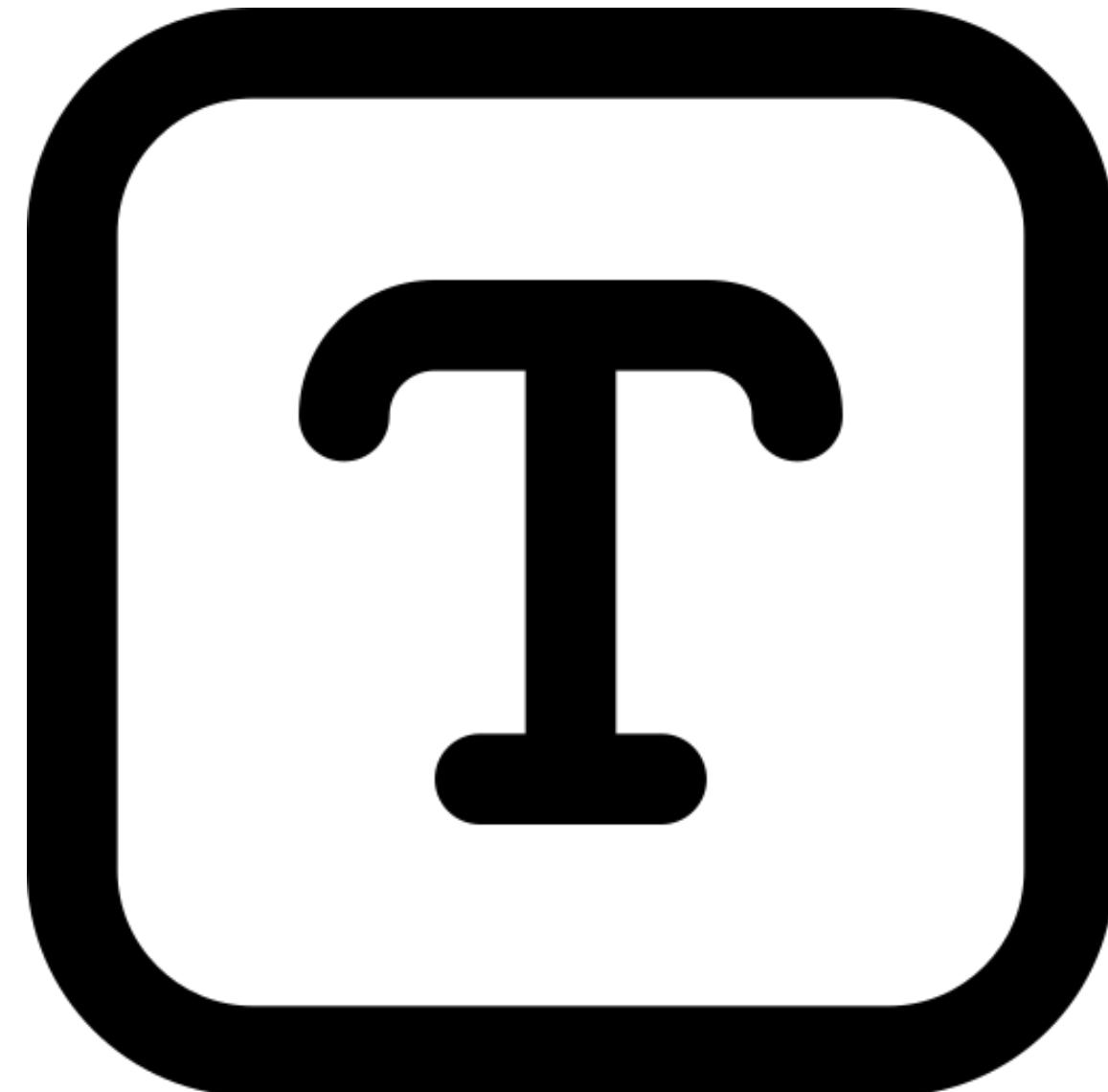
- 最直接了當的使用者操作。
- 通常標有指示操作的文字或圖示。
- 可以有不同的視覺狀態(例如: 正常、懸停、按下)。
- 介面內使用者互動和導航的基礎。



TextField 輸入框

請問您今年貴庚？

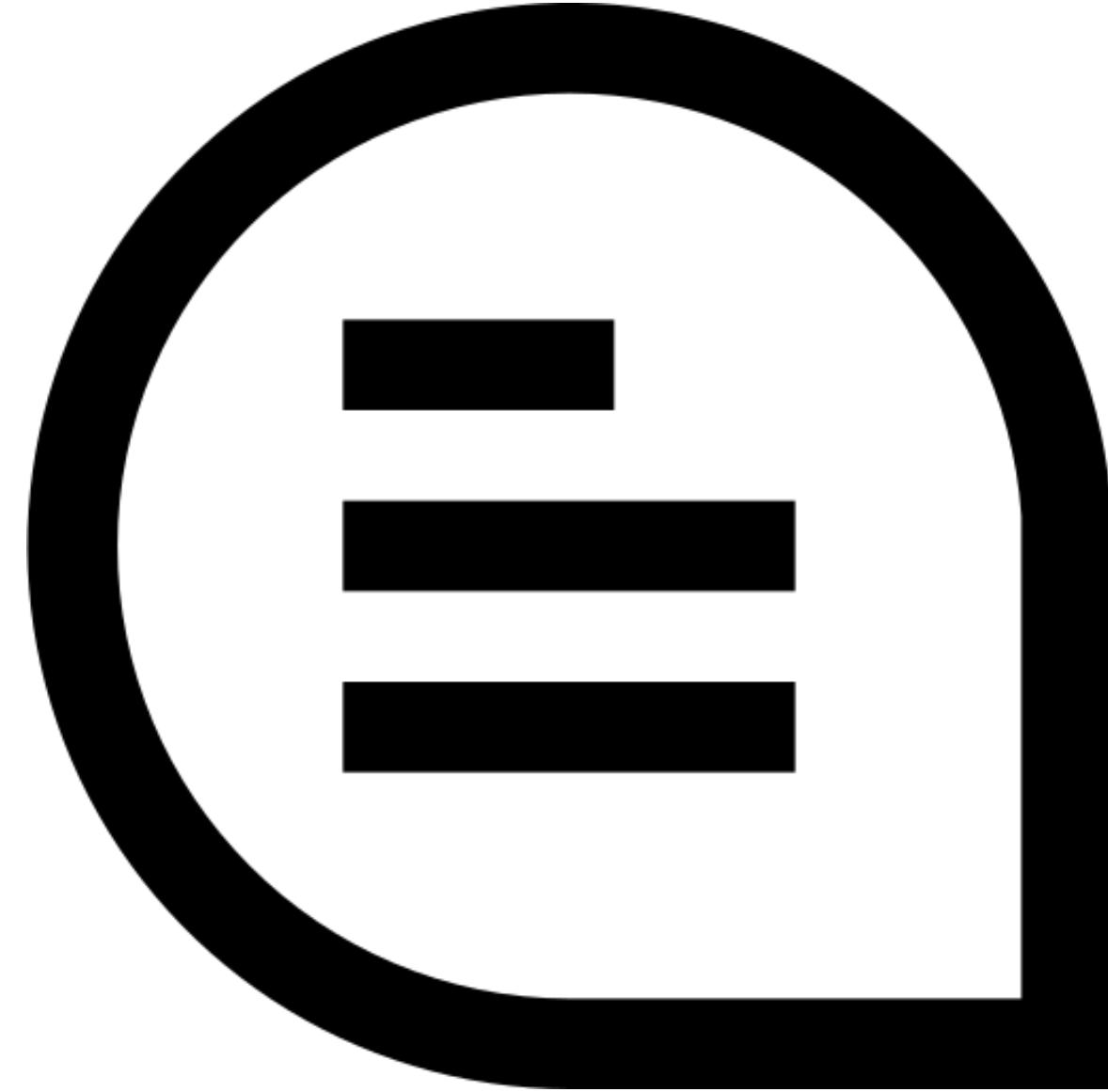
- 允許使用者輸入文字或數值。
- 可以是單行或多行輸入。
- 支援純文字、密碼或格式化文字等各種輸入類型。
- 可以加入自動完成或驗證等功能。



OptionPane 彈出視窗

喜歡系統錯誤嗎，這裡有一個

- 用於向使用者顯示訊息、警報或提示。
- 可以包含使用者輸入的選項
- 提供一種在不中斷主介面的情況下與使用者互動的方式。
- 一般也會稱作Dialogue Box



CheckBox 勾選方塊

我想要收到系統發布的定時訊息

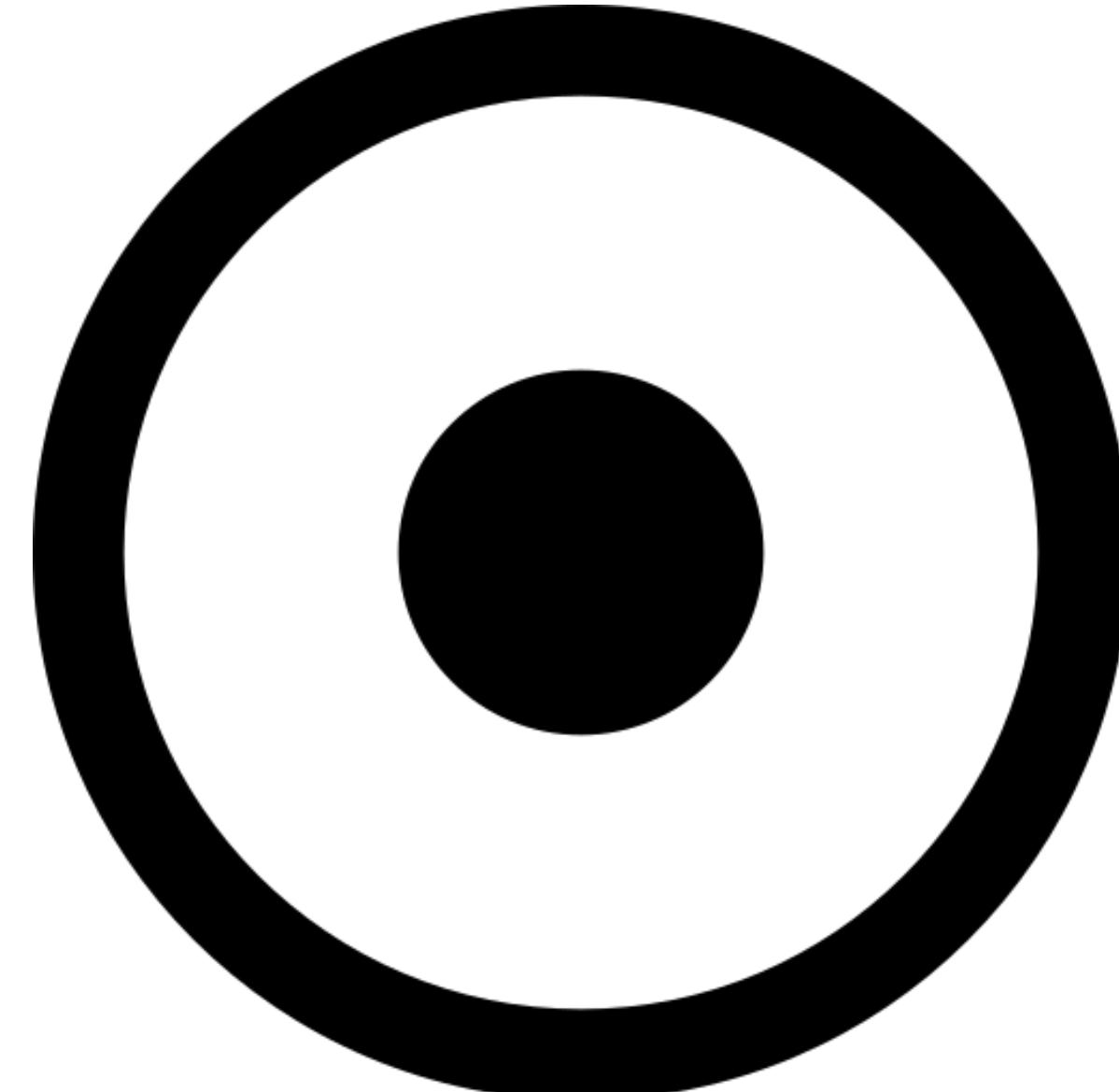
- 允許使用者從選項清單中選擇一個或多個選項。
- 通常用於二進制(選中/未選中)選擇。
- 提供使用者選擇狀態的清晰視覺指示。
- 通常在表單或設定中用於啟用或停用特定選項。



RadioButton 單選方塊

Radio Ga Ga? 不是，不要

- 向使用者呈現一組互斥的選項。
- 用戶一次只能從群組中選擇一個選項。
- 用實心圓圈或類似標記直觀地指示所選選項。
- 通常在使用者需要從清單中精確選擇一個選項時使用。



ComboBox 下拉式選單

是誰用字母排序月份的啦？

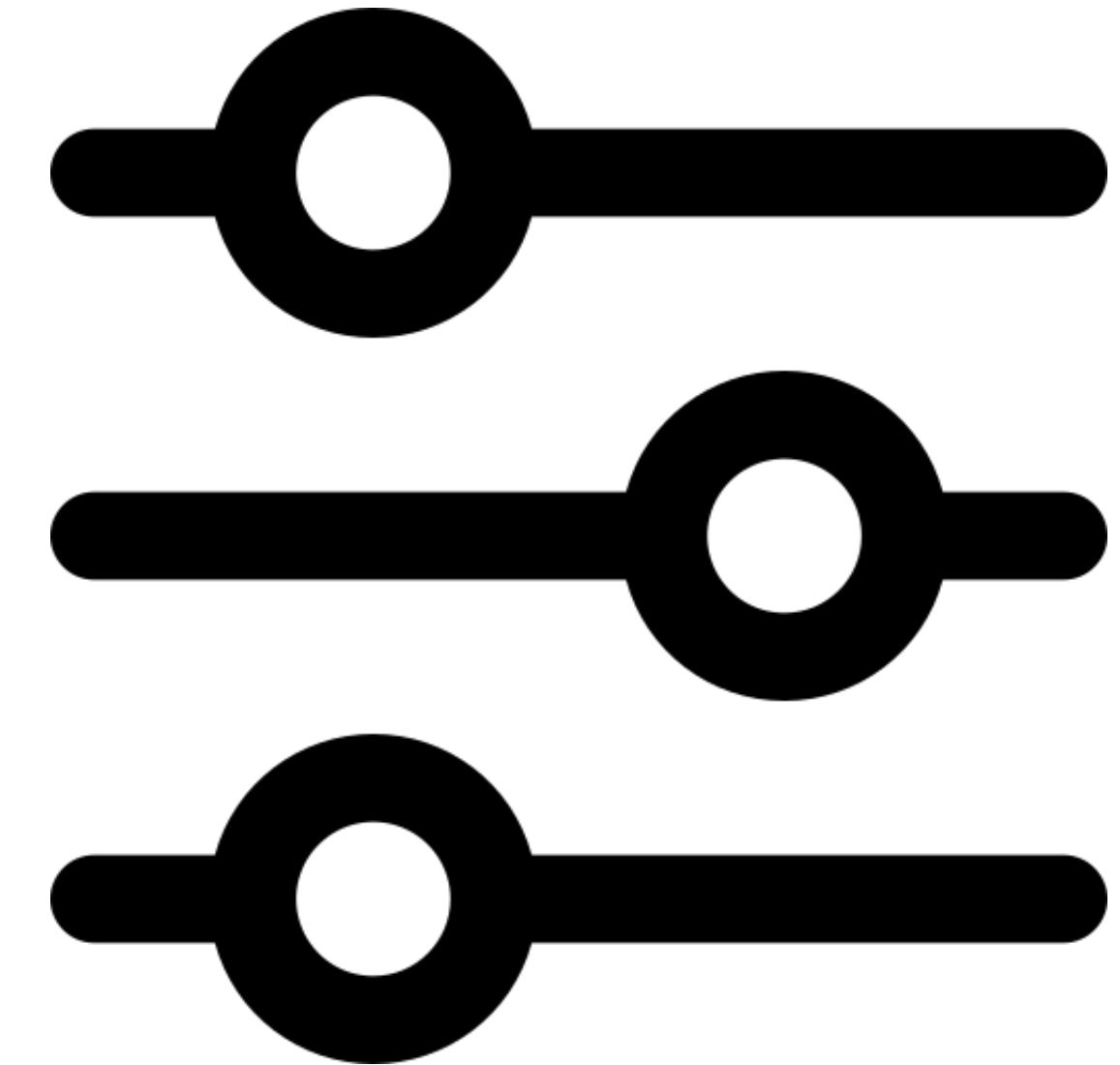
- 下拉清單與可編輯文字欄位結合。
- 為使用者提供預定義選項清單以及輸入自訂值的能力。
- 節省空間，同時提供輸入彈性。
- 對於從預定義清單中進行選擇或輸入自訂資料很有用。



Slider 滑槽

說真的我不知道要怎麼介紹這個

- 允許使用者透過沿著軌道拖曳手柄來從連續範圍中選擇一個值。
- 提供可用範圍和所選值的直觀表示。
- 對於調整具有各種可能值的設定或參數非常有用。
- 可以包含標籤或標記來指示特定值或間隔。





Developer Student Clubs
National Kaohsiung Normal University

附錄2: Java簡單語法

想不到還是會碰到程式...

```
    org = filterByOrg ? study.lead_organization == filterByOrg : true;
    if (status = filterByStatus ? study.status === filterByStatus : true) {
        matchStatus);
    }
}
function filterStudies({ studies, filterByOrg =
    filterByStatus }) {
    return studies.filter(study =>
```



變數的制定

我說你是甚麼你就是什麼

先前有碰過過程式的同學應該會對變數型別有一點印象，在靜態型別的程式語言之中，我們要指定一個變數就要清楚地告知其型別。

```
int x = 10;
```

但在物件導向程式語言之中，由於很多型別都物件化了，所以我們往往會需要做分配給記憶體的動作。

```
Integer x = new Integer(10);
```



原始型態 v.s 物件

Auto-Boxing不是自動拳擊

在Java當中，為了方便起見，所有原始型態與物件類型轉換都有一個特別的機制稱作自動裝箱、拆箱，讓你方便的在兩者之間做轉換。

物件的特性是他擁有成員，包含屬性(Attribute)和方法(Method)
而原始型態純粹佔較少記憶體，並且能夠進行直接運算。

所以像是Integer x = 10; 這種狀況是被允許的，因為自動裝箱把他轉成物件了。

如果可能，敬請迴轉

程式不會優柔寡斷

條件式(Statement)存在於幾乎所有程式語言當中，其概念就是提供一個判斷的機制。

if(判斷式){} else{}

在if的區塊當中，假如判斷式的值為真，就會執行區塊內的程式碼，否則執行else區塊內的程式碼。

else區塊不是必要的，並且可以串接多個if來達到判斷多個條件的目的。

在等待甚麼事發生嗎？

把幾百行縮成幾十行的方法

迴圈(Loop)，名副其實就是重複執行一個區塊內的所有程式碼，直到一個條件被達成。

一般常見的迴圈有while跟for兩種，前者單純就是等待一個條件，後者比較麻煩。

while(判斷式)會在判斷式不成立時跳出迴圈。

for(初始式; 判斷式; 疊代式)會在進入迴圈時執行初始式、在每一個循環前進行判斷式(不成立時跳出)並在每一次迴圈結束後進行疊代式。



我覆蓋一張卡，結束這回合

交給外包廠商做事就好

函式(Function)的概念與數學相仿，都(可能)有參數，並皆會進行封閉的運算。你可以想像成寫一篇筆記，需要的時候就拿來用。

在Java中，方法的定義是這樣的：

[作用域] [回傳類型] [函式名稱] (參數) {}

例如: public void testMethod(int argumentX) {}

我們就成功建立一個公開(public)的無回傳值(void)函式了。



物件的模板

食譜、說明書，或者該說是設計圖

類別(Class)可以包含很多東西，在其中的變數被稱為屬性(Attribute)，函式被稱為方法(Method)，並且通常會包含一個建構式(Constructor)。

用類別可以去產生物件，就像變數一樣去進行其他操作。

之所以我們能夠拉出按鈕、標籤等等物件，就是因為元件本身是一種類別。實例化的物件會擁有該類別當中的所有公開成員，包含屬性和方法。

其他詳細的介紹就屬於進階物件導向的範疇了，我們先在這裡打住。



哪裡都拿一點

今天，任何產品都會有屬於它的標準

介面(Interface)可能會是更抽象的概念，但是可以試著去這樣理解：

介面提供了一個樣板給類別，告訴他們可以加入什麼方法或者屬性，一個類別可同時實作多個介面，但是介面本身不能被指派為物件。

在介面的定義當中，只能夠有常數和抽象的方法存在，字面上就是不能夠有任何可運行的程式碼，實作的部分會留給後續的類別處理。

(靜態方法與預設方法除外，但我們並不會提到)





Developer Student Clubs
National Kaohsiung Normal University

附錄3：困難的方法

專門為強者設計的方法

```
filterByOrg = filterByOrg ? study.lead_organization === filterByOrg : true
filterStatus = filterByStatus ? study.status === filterByStatus : true
filterMatchStatus = filterMatchStatus ? study.status === filterMatchStatus : true

function filterStudies({ studies, filterByOrg, filterByStatus, filterMatchStatus }) {
  return studies.filter(study => {
    if (filterByOrg) {
      if (filterMatchStatus) {
        return study.lead_organization === filterByOrg && study.status === filterByStatus
      } else {
        return study.lead_organization === filterByOrg
      }
    } else if (filterMatchStatus) {
      return study.status === filterByStatus
    }
  })
}
```



保險措施 (簡報 P.38)

給他一個空值保護

還記得這段程式碼嗎? 我們並沒有提到外面的
!titleBox.getText().equals("") <- 字串比較式

這樣子的寫法其實是用if判斷是去察看那個
TextField裡面是否有文字, 如果沒有的話就不動作

不加這個不會讓程式爆炸, 只是這樣可以預防出
現空白的標題

```
JButton confirm = new JButton("確認");
confirm.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        if(!titleBox.getText().equals("")) {
            String title = titleBox.getText();
            titleDisplay.setText(title);
            titleBox.setText("");
            System.out.println("Success");
        }else {
            System.out.println("No success :(");
        }
    });
    confirm.setBounds(272, 181, 87, 23);
    contentPane.add(confirm);
```

直接塞路徑 (簡報 P.51)

圖片也只是一種二進位檔案

由於setIcon所需要的參數是一個ImageIcon物件，我們就直接手搓一個放進去

我們能發現這個ImageIcon類別的建構式只需要其路徑作為參數即可，因此可以這麼做

```
ImageIcon x = new ImageIcon("你的路徑");
```

然後再把x丟進去setIcon裡就可以了

ImageIcon(String filename)

Creates an ImageIcon from the specified file.

ImageIcon(String filename, String description)

Creates an ImageIcon from the specified file.

ImageIcon(URL location)

Creates an ImageIcon from the specified URL.

ImageIcon(URL location, String description)

Creates an ImageIcon from the specified URL.



卷軸面板 (簡報 P.69)

堪比魔術大空間的容器

JScrollPane的最大特色是他具有水平與垂直滑軸，當內容物大於其尺寸時會自動出現，因此像是JLabel裡放圖片、JList有多筆資料存在等等就很好用

不過要注意的是，這種容器裡只能放最多一種元件，所以可以把它當作是展示框一樣的存在

實作上這個元件基本由 JScrollPane、一個 JViewport 以及它們之間的連線組成

Continents Largest To Smallest			
S.No	Continent	Area (square km)	Percentage
1	Asia	44,579,000	29.5%
2	Africa	30,370,000	20.4%
3	North America	24,709,000	16.5%
4	South America	17,840,000	12.0%
5	Antartica	14,000,000	9.2%
6	Europe	10,180,000	6.8%
7	Australia	9,600,000	6.0%

原始打包程序 (簡報 P.79) - 1

我沒有像是Eclipse這樣的好東西

在沒有像是Eclipse提供的包裝器的情況下，我們又該怎麼把檔案給打包起來呢？
大多數情況下，我們會使用系統的CMD (命令提示字元)

記起來，簡單(?)四步驟如下：

1. 確認程式可以跑，沒有問題
2. 用javac編譯成字節碼
3. 創建Manifest文件
4. 將其打包並匯出成一個完整的 JAR檔案



原始打包程序 (簡報 P.79) - 2

命令提示字元太強大

在這裡我們實作一次流程：

- 確認程序可以運行：

首先，確保你的Java應用程序在開發環境中可以正常運行，並且所有功能都已實現和測試完畢。

- 編譯成字節碼 (.class)：

在CMD中輸入以下指令-

```
javac -d bin src/com/example/App.java
```

示例

假設有一個簡單的Java Swing應用程序，主類為`com.example.MyApp`，代碼結構如下：

```
css 複製程式碼  
src/  
└ com/  
  └ example/  
    └ MyApp.java
```

`MyApp.java` 的內容如下：

```
java 複製程式碼  
package com.example;  
  
import javax.swing.*;  
  
public class MyApp {  
    public static void main(String[] args) {  
        JFrame frame = new JFrame("My Swing Application");  
        frame.setSize(400, 300);  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        frame.setVisible(true);  
    }  
}
```

原始打包程序 (簡報 P.79) - 3

Shell指令的部分我們下回再作影片解析 (X)

在這裡我們實作一次流程：

- 創建Manifest文件：

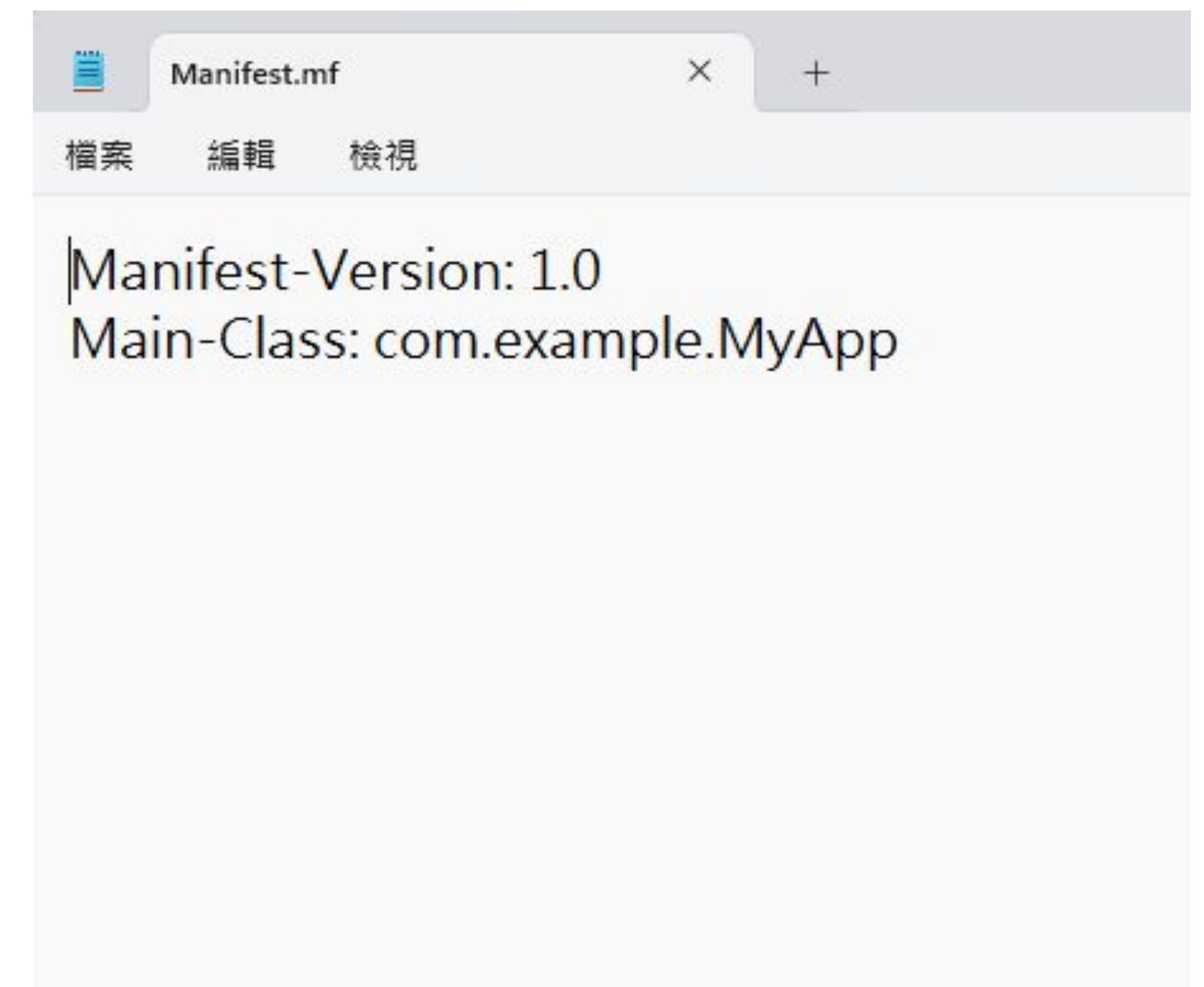
打開你的記事本，寫進右邊那些東西，注意最重要的 Main-Class一定要指向你的程式的進入點。

- 打包成JAR：

在CMD中輸入以下指令-

```
jar cfm MyApp.jar MANIFEST.MF -C bin .
```

然後就完成了，檔案會保存在你的根目錄裡



軟體的設計模式 (簡報 P.90) - 1

就像是模板，照著做準沒錯

我們業界的軟體設計並不是無頭蒼蠅，而是有著一些固定的規範，也就是所謂的設計模式的存在，不同的設計模式會帶給使用者不同的體驗，因此相當重要。

這裡介紹幾個最重要的設計模式：

1. **MVC**: 當有用戶的行爲觸發操作時，控制器(Controller)更新模型，並通知視圖(V)和模型(M)更新，這時視圖(V)就會向模型(M)請求新的數據。
2. **MVP**: 在 MVP 中，Presenter 可以理解為鬆散的控制器，其中包含了視圖的 UI 業務邏輯，所有從視圖發出的事件，都會通過代理給 Presenter 進行處理；同時，Presenter 也通過視圖暴露的接口與其進行通信。



軟體的設計模式 (簡報 P.90) - 2

我們真的有必要知道這些東西嗎

續前頁:

3. MVVM: View Model 的狀態改變可以自動傳遞給 View。典型的情況是, View Model 通過使用 obsever 模式(觀察者模式)來將 View Model 的變化通知給 model。

總結來說, MVC 只是把控件跟 M 綁定, 一遍遍刷新 UI。

而 MVP 則是把控件跟事件單向綁定, 它的假設是程序員最愛寫低級的代碼來操作控件。

MVVM 則是把控件跟 VM 雙向綁定, 它的假設是交互界面設計時最愛寫高層次一些的聲明來操作用戶業務行爲上的變化。



Developer Student Clubs
National Kaohsiung Normal University

附錄?: 簡報小彩蛋

Bernie的LinkTree



被你發現了，真可惡

她的名字是U.E.P, 是我
(Bernie)的看版娘喔~~~

```
const filterByOrg = study => organization === filterByOrg ? true : false
const filterStatus = filterByStatus ? study.status === filterByStatus : true
const filterMatchStatus = filterMatchStatus ? study.matchStatus === filterMatchStatus : true

function filterStudies({ studies, filterByOrg, filterByStatus, filterMatchStatus }) {
  return studies.filter(study => filterByOrg(study) && filterStatus(study) && filterMatchStatus(study))
}
```

