JUDE User Guide

使用手冊 version 0.3

Table of Contents

1. 使用案例圖

1.1. 簡介

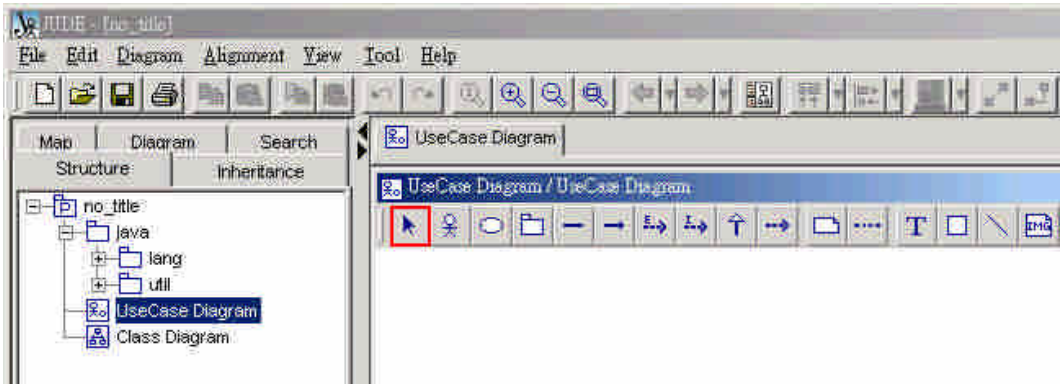
使用案例圖表達系統所提供的功能，是從使用者的觀點來描述。

1.2. 繪製使用案例圖

執行 JUDE。單擊左側區域中的 “Use Case Diagram”，右邊會出現空白的繪圖區域以及繪製使用案例圖的功具表與工作區。

[JUDE]

工具表：

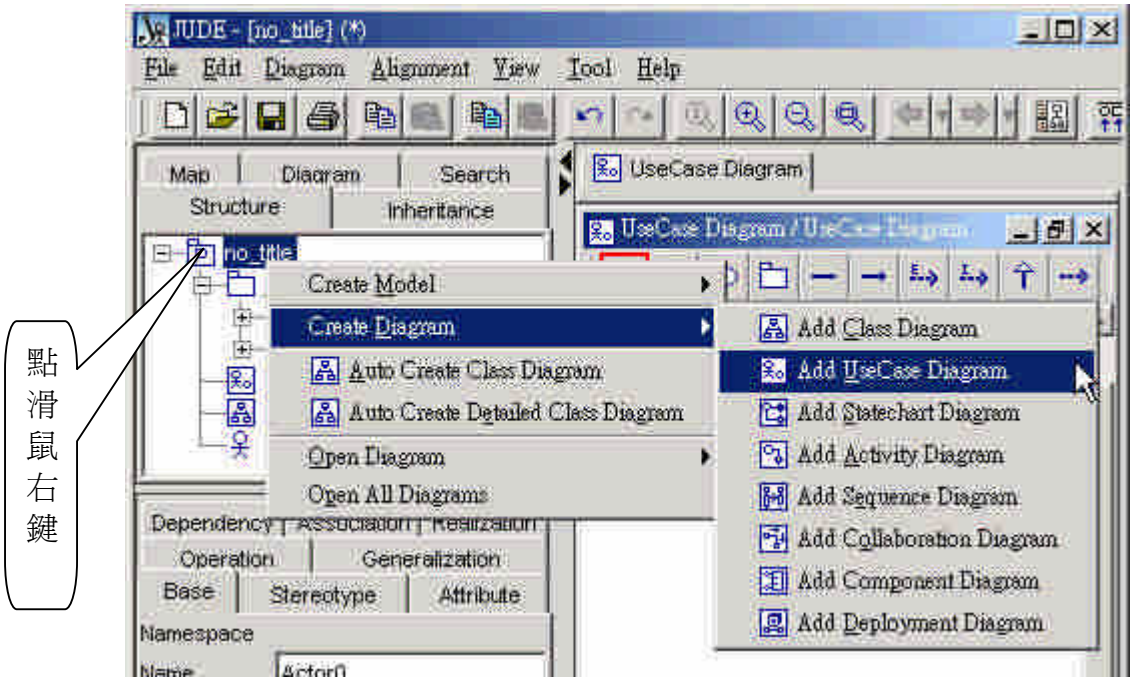


建立各式的 UML Diagram

在 JUDE 中，你可以建立許多張使用案例圖 。基本上，我們可以建立許多張各種不同的 UML 圖，在此，我們以使用案例圖爲例子。

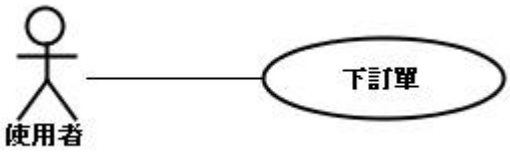
建立使用案例圖做法如下：滑鼠右鍵點選 “no_title”，一個 popup 螢幕會出現。如果你移動指標到第二個選項 “Create Diagram”。另一個子目錄會出現。任何的 UML 圖形均是以相同的方式來創建。點選 “Add Use Case Diagram” 來增加一張新的使用案例圖。

[JUDE]



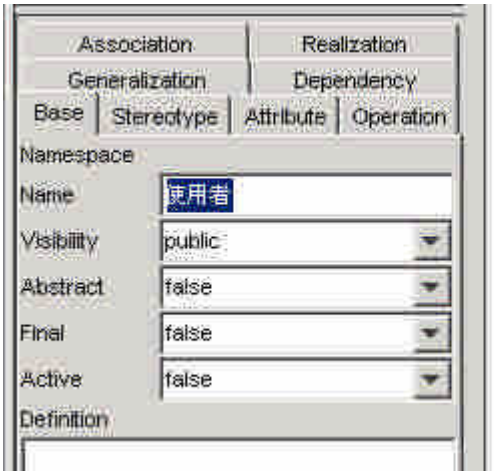
從 UseCase Diagram 的工具表上，可以看到各種在使用案例圖中出現的圖形。讓我們假設說我們想要繪製"使用者下訂單"這個使用案例。首先，我們先畫"使用者"這個角色。
作法：**點選 Actor (Actor 會出現紅色框框) > 點擊工作區。**
這時候，一個名爲 actor0 的人形圖像就會出現了。你可以在這時改變角色的名稱爲"使用者"。不然，也可以在之後改變。接著繪製使用案例，
作法：**選 UseCase 圖 (第三個橢圓形) >點擊工作區。將名稱改爲" 下訂單"**
將 actor 與 use case 連結以表達哪一個角色使用哪個系統功能。畫法如下：
作法：**選 association 圖 (第五個直線形) > 移動鼠標到 actor 圖 (會出現藍色方框) > 滑鼠左鍵點 actor 但是不要放開，然後拖曳到 use case。**

結果如下：



Actor 的屬性更新

各種圖形的屬性可以在繪圖之後再加以修正。每個圖形有其相關的屬性編輯器，比如說 actor。當你點選 actor 圖形時，左側會出現 actor 的屬性編輯框。如果你當初沒有在建立之初給定名稱"使用者"，你可以在名稱欄中輸入。輸入後請按 enter 鍵。



屬性編輯器是相當有用的工具，尤其是當圖變得相當複雜而難以直接在圖中修改時，善用屬性編輯器是一個有用的方法。你可以試著點選連結線或是使用案例圖，觀察及熟悉不同屬性編輯器的變化。

移除圖形的屬性

在 JUDE 中當你移除圖形時，基本上，只是圖形從工作區消失了，你在之後還是可以從左上方的樹狀圖中將圖型拖曳到工作區。讓我們試著刪除上圖中的連結線。
作法：**點選連結線 > 按 Delete 鍵**
線雖然不見了，可以使用者與"下訂單"案例的關聯姓還是存在。這點可以從"使用者"的屬性編輯器之 "Association" tab 看出。

另外，你也可以用另一種方法驗證：將"下訂單"使用案例刪除。然後再將"下訂單"使用案例從左上拖曳到工作區。關聯線還是又出現了。

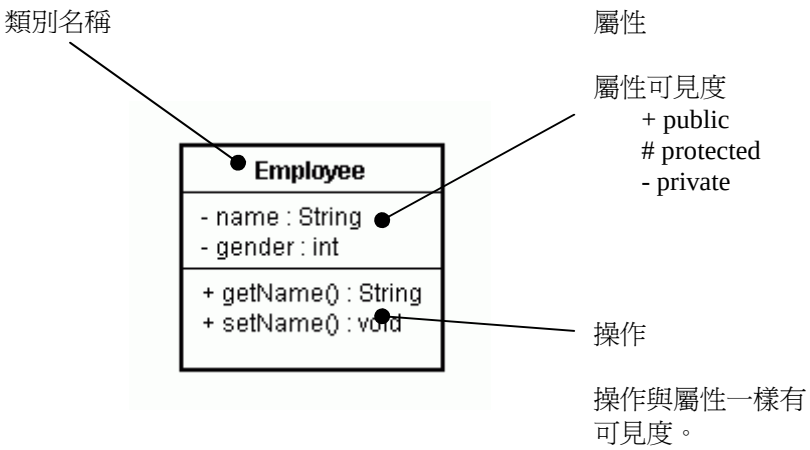
因此，當你真正需要刪除圖形時，要從屬性編輯器著手。
作法：**在 Association Tab 中，點選要刪除的關連 "下訂單" > 按 "Delete"**
這時候，你如果再從左上方拖曳"下訂單"使用案例到工作區，使用者與下訂單就不會再出現關聯了。

以上的作法適用於各類的圖型本身，或是關係上。

2. 類別圖

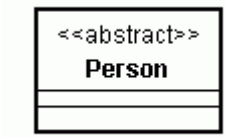
2.1. 簡介

一個類別圖包含有類別名稱，屬性，操作三部份。下面爲一個 Employee 雇員的類別。



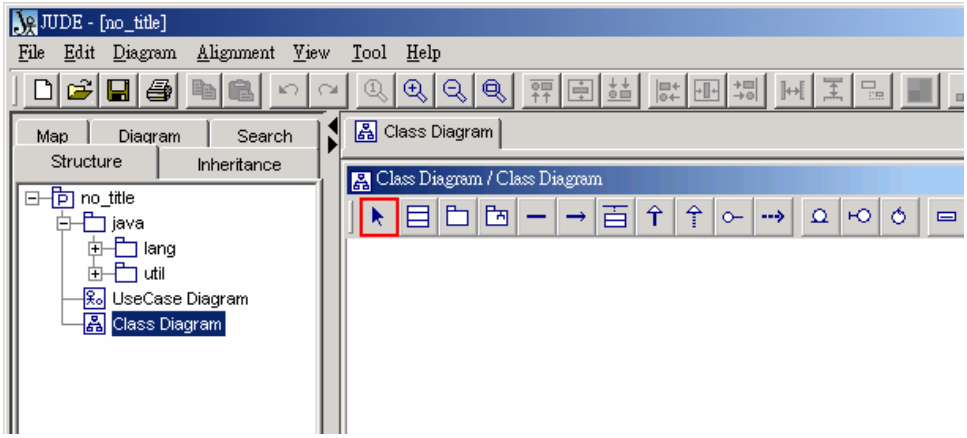
Stereotype

利用 << 以及 >> 將字框起來。一般來說，它與 CASE 工具在產生程式碼時無關。它是給人類看的。用來表示出此類別的特殊意義。比如說，如果你將所有的類別分成 A 以及 B 兩類，那麼你可以用<<A>>來標示出一個類別是屬於 A 這個種類。下面是一個人的類別圖。Stereotype 用以告訴我們，它是一個抽象的(abstract)類別。所以我們知道我們不能夠用 new 來建構一個 person 物件。



2.2. 繪製類別圖

執行 JUDE。單擊左側區域中的 “Class Diagram”，右邊會出現空白的繪圖區域以及繪製類別圖的功具表與工作區。



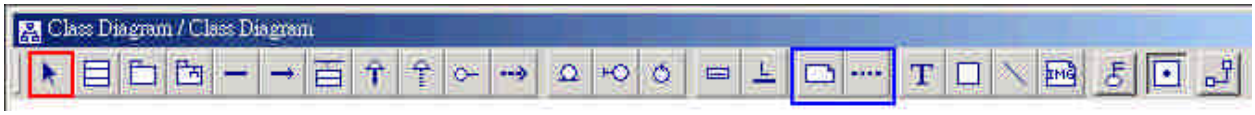
2.3. 類別圖工具列

主要工具圖形元件(從左至右)

選擇工具(Select)
類別(Class)
套件(Package)
子系統 (subsystem)
關聯關係(association)
單方向關聯關係(unidirectional association)
關聯類別(association class)
一般化(Generalization)
實現化(Realization)
介面(Interface)
相依(Dependency)

分析物件圖形(從左至右)

實體物件(Entity)
邊界物件(Boundary)
控制物件(Control)



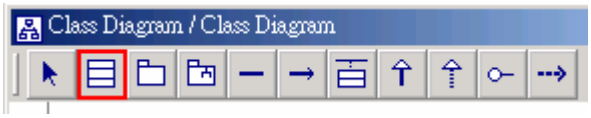
文件說明物件

註解圖形(Note)
註解連結線(Note Anchor)

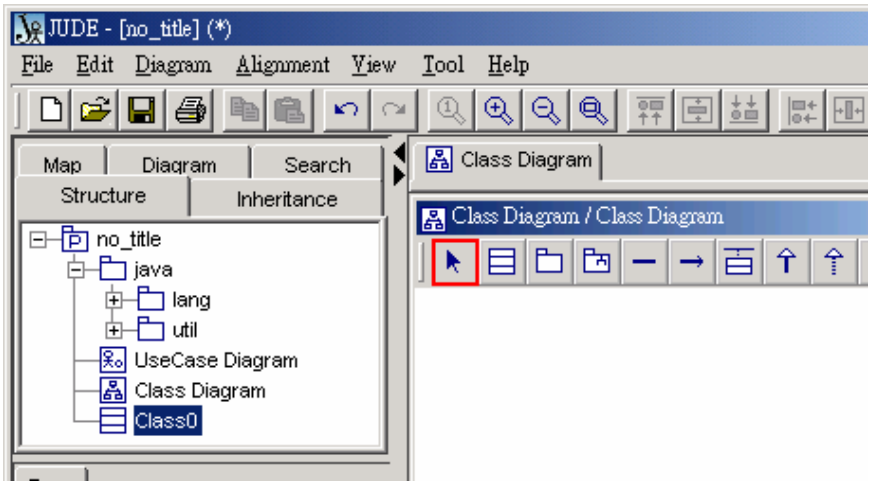
2.4. 建立新的類別

顯示了類別圖的繪圖區域後，有關類別圖的各種圖形以及符號皆可由工具圖表來完成。

假設我們要繪製一個帳戶(Account)的類別。首先點選工具圖中的第二個圖，如下：

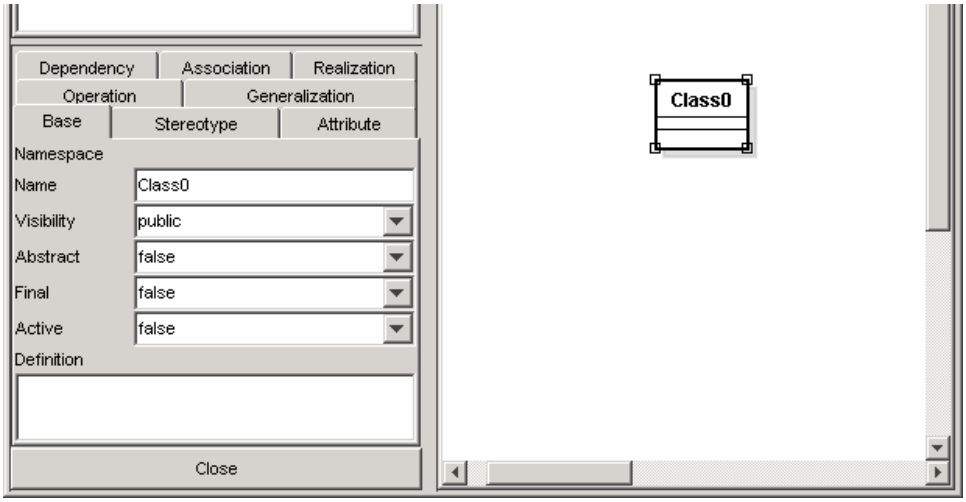


然後，點一下繪圖區。



在一開始，類別圖都有預設的名稱。例如：Class0, Class1 等等。

接著，點選該類別。(點選時，請點選圖形的邊框) 。這時，左方會出現有關該類別的編輯工具表單，如下。

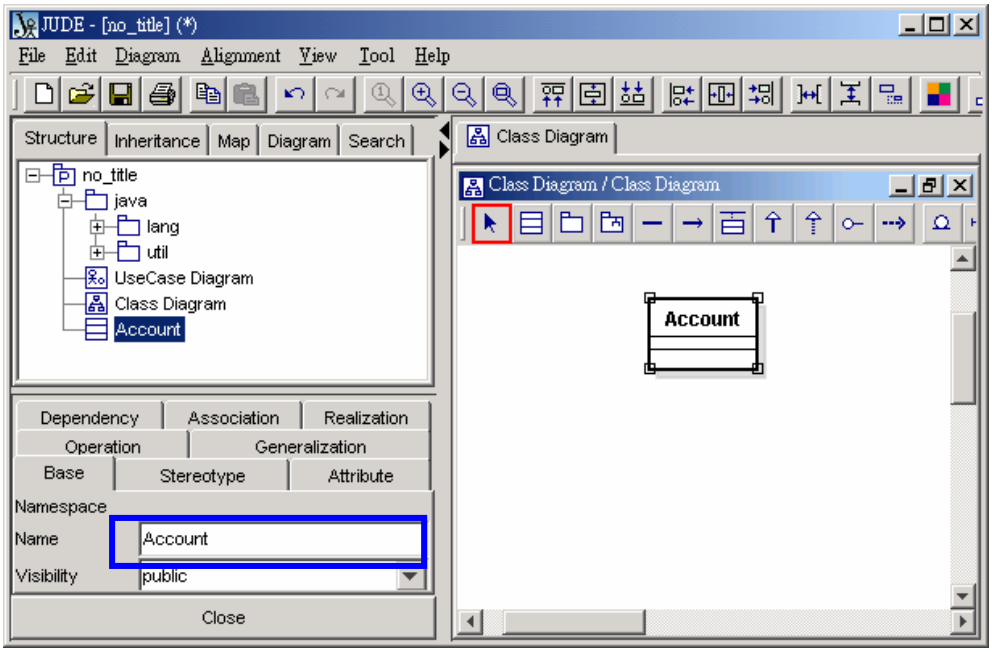


Base： 類別的基本的資訊
Stereotype： 設定 stereotype
Attribute： 屬性
Operation： 操作
Generalization：一般化
Dependency： 相依性

Association： 關聯性
Realization： 實現化

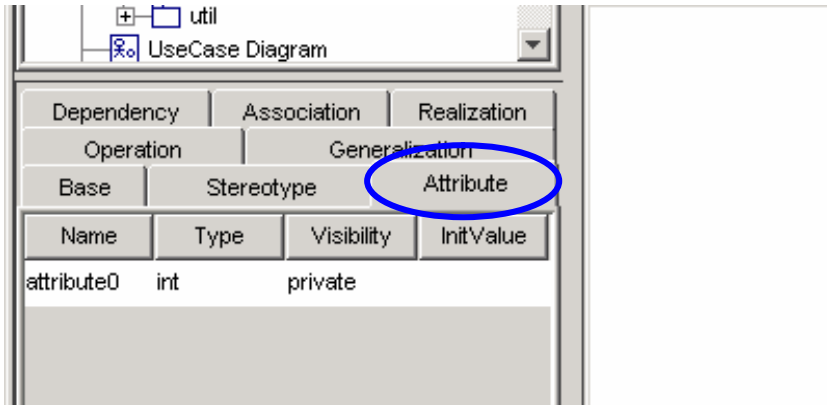
2.5. 類別的基本資料(Base 表單)

讓我們修改一下類別的名稱爲 Account。在 name 中打入 **Account**，然後按 Enter。(記得！一定要按 Enter)。除了名稱之外，我們也可以在此表單中設定類別的能見度(Visibility)，該類別是否爲抽象類別(Abstract) 以及該類別是否爲 Final 類別。

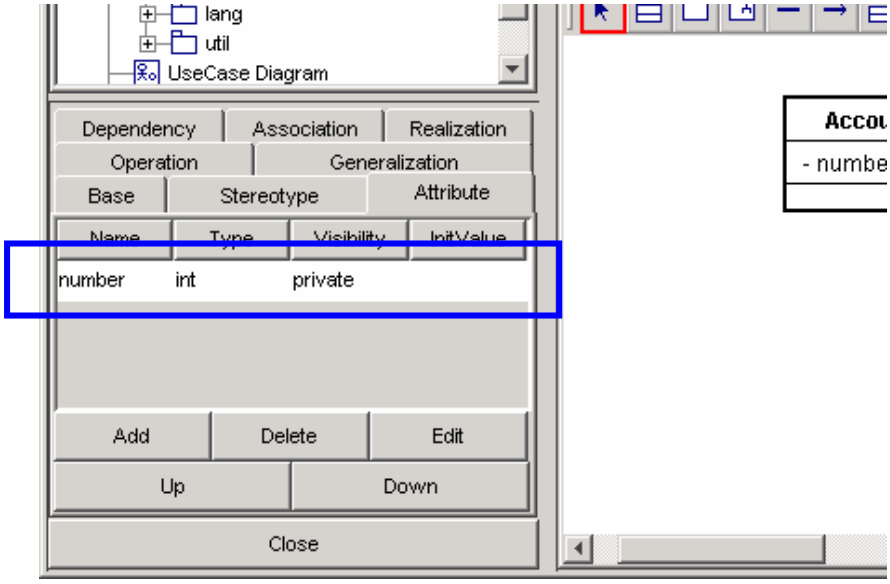


2.6. 屬性 (Attribute 表單)

基本常識告訴我們，一個帳戶有帳號(number)以及餘額(balance)。因此，我們可以爲 Account 增加這兩個屬性。點擊 Attribute 表單。在 Attribute 表單下方，點"Add"。



同樣地，JUDE 提供了預設值。雙擊 attribute0 這個字，就可以進入編輯的狀態。打入 **number**，然後按 Enter。你應該注意到了右邊的類別圖中，屬性的欄位也會同步改變資料。除了名稱之外，如果有需要，我們也可在此表單中一併修改屬性的型態(Type)，屬性的能見度(Visibility)，以及該屬性的初始值(InitValue)。

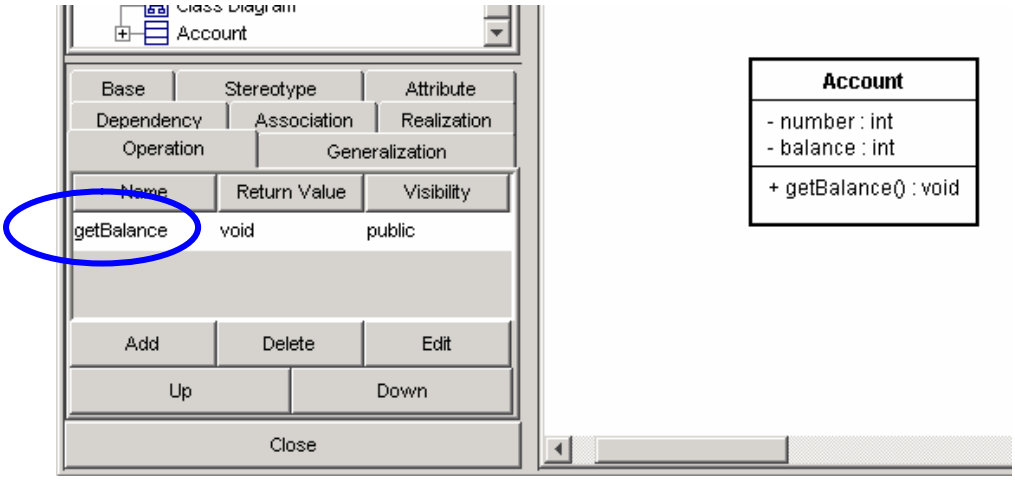


利用如上的方式，增加一個餘額(balance)屬性。

2.7. 操作 (Operation 表單)

對於一個帳號物件，它應該自己知道還有多少餘額(因爲餘額是它的屬性)。爲了讓外界可以詢問一個帳號務見有關它的餘額，我們設計讓 Account 提供一個取得餘額(get balance)的操作。我們就將它命爲 getBalance()。讓我們爲 Account 類別增加這個操作。

點選 Operation 表單，然後 “Add”。在此，操作過程如同屬性表單。所以雙擊一下操作的預設值然後輸入 **getBalance** (不需要打左括號，右括號) ，然後按 Enter。畫面結果如下。



3. 類別之間的關係

在類別圖中最不容易了解的應該是用來表示類別之間的"關係"。

- 從類別的角度來看。

  在類別中，有一些關係是用來表達結構上類別與類別之間的關係。例如:繼承(inheritance)的關係，介面(interface)實現化的關係。這一類的關係你可以把它看作是固定的，不會因時間的改變而有所改變的關係。用一個現實生活中的比喻，你跟你父親之間的血緣關係。我們稱之爲父子關係。這個關係不會因爲你成家立業搬出去住了而有所改變。

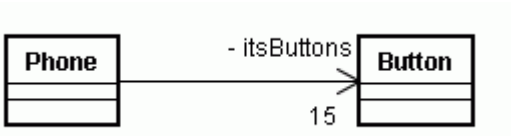
- 從物件的角度來看。也就是說物件們是利用什麼關係來一起合作的。

  在本質上，這些物件之間的關係並沒有如上所述那種無法改變的觀念。想一想你的週遭環境事物。除了上述之家庭的關係之外，你可能有在學校修課。你跟學校之間建立了學生的關係。一但你畢了業，這個關係就消失了；還有，你可能是某個組織，社團的一員，那麼，你跟此組織或是社團就有會員的關係。這些關係與你跟你的家人的那種固定不變的關係是不一樣。在物件導向的世界中，我們稱這種關係爲關聯關係(association)。

3.1. 關聯性 (Association)

關聯關係代表物件與物件在結構上的連結。關聯關係是用一條直線來連結相關的物件。然後在關聯的線上寫上兩者之間的關係名稱。

從程式語言的觀點，類別之間的關聯關係通常代表著一個類別中擁有另一個類別作爲變數的參照(reference)。比如說，一具電話有 15 個按鈕，那麼，你可以將這個關係表示成：



上圖，連結線是帶有箭頭。箭頭的方向表示出哪個類別握有其他類別的參照。所以，在程式中，它所表達的程式碼如下。

```
public class Phone{
      private Button[15] itsButtons;
}
```

也就是說，是 `Phone` 類別擁有 `Button` 類別的參照。

角色名稱(role)

靠近箭頭的文字代表著變數的名稱。一般我們稱它爲角色名稱。

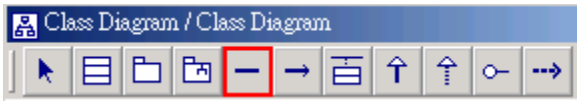
多重性(multiplicity)

靠近箭頭的數字代表著 phone 握有多少個 button 的參照。

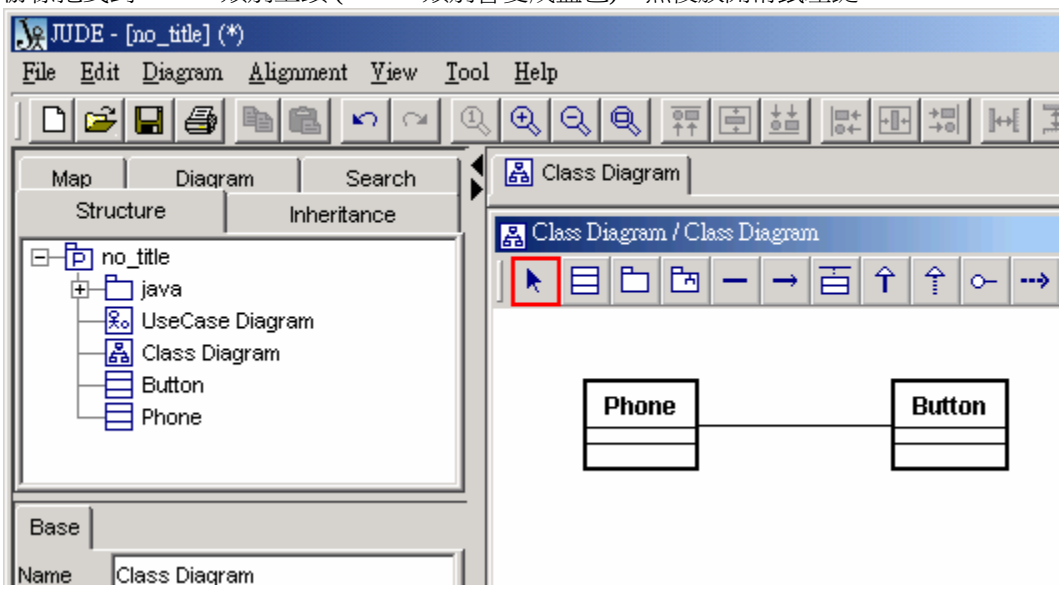
因此，上面的例子說明了 phone 有 15 個 button 參照。這個數字在 UML 中稱爲多重性。

[JUDE]

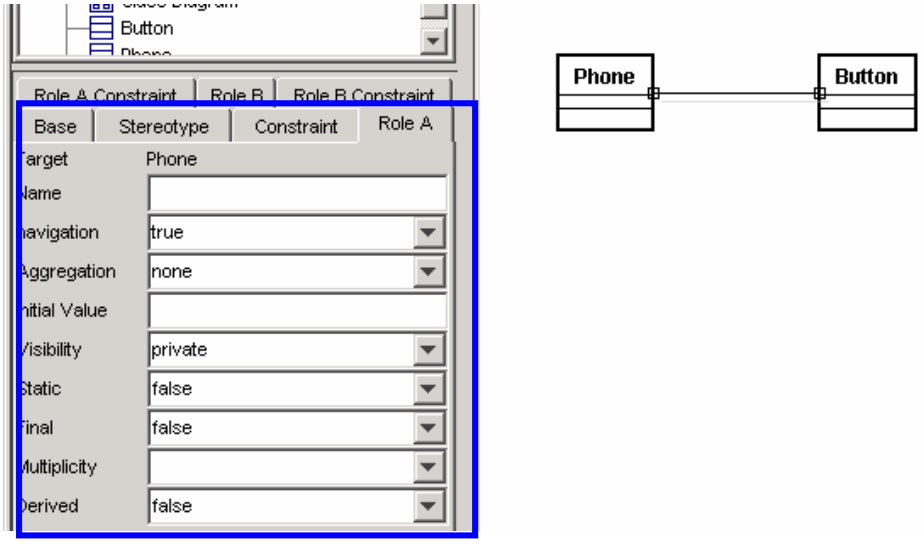
首先繪製出兩個類別，一個是 Phone，另一個是 Button。然後點選工具圖上的第五個按鈕。



將滑鼠游標移動到 Phone 類別上 (類別會變成藍色)，然後按下(不要放開)滑鼠左鍵；這時，將滑鼠游標拖曳到 Button 類別上頭 (Button 類別會變成藍色)，然後放開滑鼠左鍵。

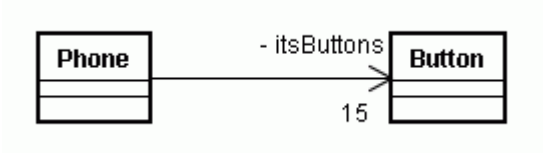


在此例中，連結線並沒有出現箭頭。爲了讓箭頭出現，請首先點選剛剛繪製的關聯線。這時候，左下方會跑出該關聯的一個設定表。

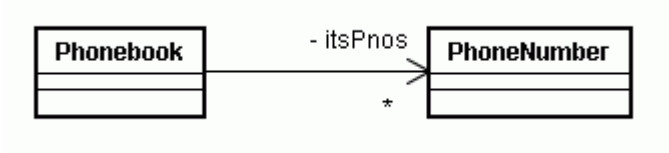


這裡有兩個 Tab 分別代表著關係中的兩方，稱爲 Role A 以及 Role B。從表單中，我們可以看到 Role A 指的是 Phone 類別。Role B 指的是 Button 類別。許多與 association 的設定均可在此修改。因爲箭頭是從 Phone 到 Button，所以我們知道對 Phone 來說，它的 navigation 要設定成 false，而 Button 的 navigation 要設定成 true。(JUDE 對於 association 的預設值是假設都爲真。)

因爲要改變 Button 的 navigation，所以，點選 Role B 這個表單，然後把 navigation 選爲 false。另外，點選 Role A 表單，讓我們爲 Button 類別加上角色名稱 (在 name 中打入 "**itsButtons**") 以及多重性 (multiplicity 中打入 **15)**。完成了這些步驟，你的類別圖應該跟上面的一樣。



練習題：利用類別圖表示一個 Phonebook 類別包含有許多的 (在 UML 中用*表示) PhoneNumber。

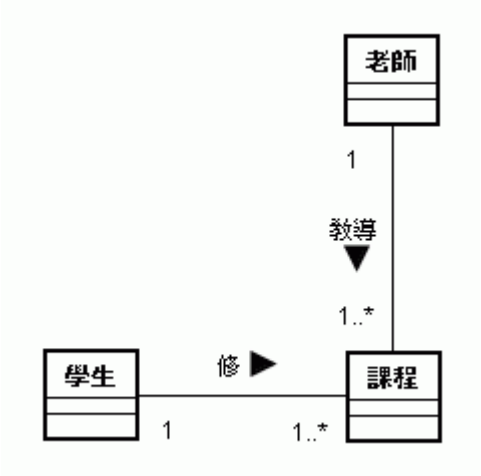



```
public class Phonebook{
      private Vector itsPnos;
}
```

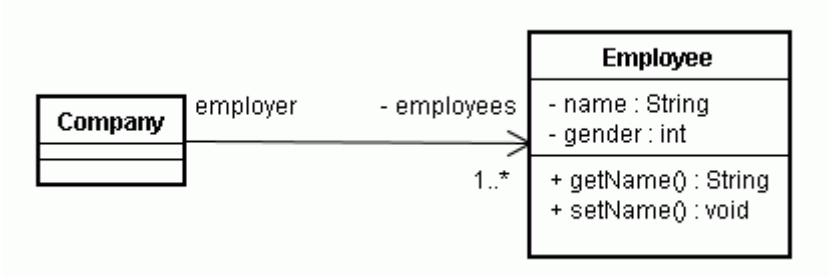
多重性表法：

名稱	表法	例子
恰好一個	1	一個系有一個系主任
零個或是更多	0…*	教師有零個或是多個行政工作
一個或是更多	1…*	學生主修一個或是多個學位
零或是一個	0…1	教師有一個或是零個計畫補助
指定範圍	2…4	職員一年可以享有兩個到四個假期

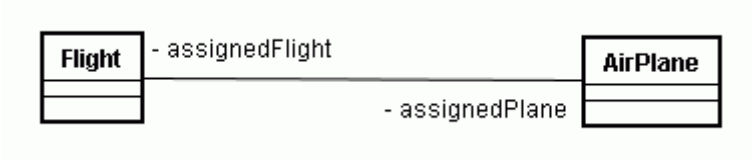
練習題：一個學生可以修 1 到 5 門課程。一門課程可能沒有人休(0)或是有很多人修(*)。老師可以教導一門以上的課程。一門課程只有 1 位老師。



練習題：一家公司有一個以上的員工。員工的屬性有名字，性別等等。你可以用如下的方式來表達：



練習題：一架飛機會有它所屬的指定航班，而一個航班也會指派某一架飛機來飛航。




```
public class Flight{
private AirPlane assignedPlane;
}

public class AirPlane{
private Flight assignedFlight;
}
```

3.2. 聚合(aggregation)關係

聚合是一種特殊的關聯關係。它是用以表達"整體和部份"的關係。 聚合關係可以看作是包含(include)的關係。
因此，聚合的關係可以用英文中的”is-part-of”， “has-a” 或是” has-parts” 語意上的關係來表示。聚合的關係不只是指出物件相互了解的關係， 它們被組合起來以行成一個新的更複雜的物件。

例子：一個球隊(Team)有球員(Player)， 球隊如果沒了， 球員還可以到別的球隊打球(聚合)。

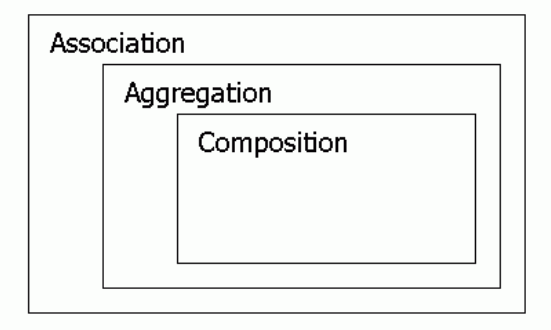

```
public class Team{
      private Player[] players;
}

public class Player{
      // 屬性
}
```

3.3. 組合(Composition)關係

組合關係是一種比聚合關係更強的包含關係。在一個聚合的關係中，如果整體的消失會造成部分(parts)的消失，那麼這個聚合是一種組合關係。例如說，一本書包含有很多章節。如果書沒了， 章節也就沒了(組合)。

上面所討論的關係是從物件的角度來看。也就是說物件們是利用什麼關係來一起合作的。關聯關係，聚合關係以及組合關係可以用如下的圖示來表達它們之間的強弱性。關聯關係最弱，組成關係爲最強。



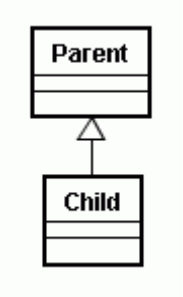
3.4. 一般化(generalization)關係

一般化的關係是一種分類的關係(taxonomic relationship)。此關係將比較一般的元素(稱爲父類別)跟比較特別的元素(稱爲子類別)分類。子類別除了跟父類別完全一致之外，它還另外具有其他的父元素所沒有的成分。除了類別圖可以有一般化的關係之外，一般化的關係也可以應用在使用案例等等相關的 UML 圖中。

在類別圖中，一般化的關係是將一組有共通性質的物件予以組織化的程序。一般化講的其實就是物件導向程式語言中的**繼承**關念。一般化可以用英文中的”is-a” 或是”is a-kind-of” 關係來表示。一個判斷一般化的簡單原則就是：在中文裡頭你如果可以用"物件 A 是物件 B 的一種"這一句話來表達時，A 跟 B 之間就可能存在著一般化的關係。例如：秘書跟工程師都是雇員的一種。雇員跟客戶都是人的一種。

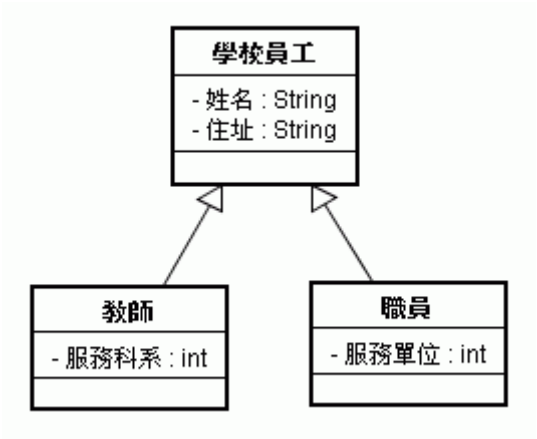
在類別圖中，透過一般化的關係，父類別的所有特徵、性質以及操作都會自動被子類別所繼承。也就是說子類別不需要再去定義它們，就可以自動擁有。這也是一般化關係所帶來的好處。適當地使用一般化的關係，也可以降低模型的複雜度，讓模型變得更好擴展以及延伸。

在 UML 中，一般化關係的表法是一條有著空心三角型的直線。畫法則是從子類別連接到父類別，如下所示：




```
public class Parent{
      //
}
public class Child extends Parent{
      //
}
```

例子：對於一個校務資訊系統，它的使用者可能爲學校授課的教授(professor) 或者是各不同層級的職員(staff)等等。不管是哪種使用者，相同的是他們都會有姓名， 住址等屬性。不同的地方，則是不同的角色所服務的場所。比如說，老師屬於其服務的科系，而職員們則是分屬於不同的行政部門。對於相同的屬性，我們可以將它們歸爲一類，利用父類別來塑模。不同的屬性，則分別屬於各子類別。

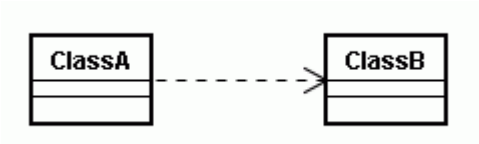


3.5. 相依(Dependency)關係

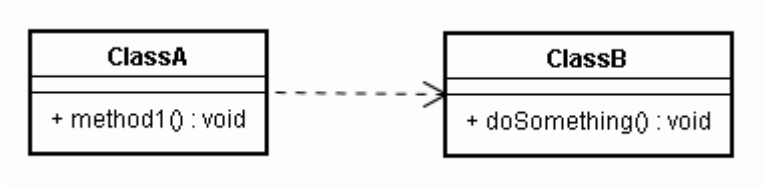
相依關係是指兩個類別之間語意上的相依關係。講的白話一點，就是說當一個類別"使用(use)" 到其他類別所提供的服務時，我們稱這兩個類別有相依關係。那麼，所謂的使用(use)到底又是在說什麼呢？當一個類別 A 會傳送訊息(passing message)給另一個類別 B 時，我們說類別 A 使用到類別 B。因此，類別 A 相依於類別 B。

類別 A 傳送訊息給類別 B，也就是說類別 A 呼叫了類別 B 的操作(方法)。

相依關係的符號表法是用一條有箭頭的虛線來表示。相依關係的箭頭是由使用類別指向被使用類別。下圖顯示出類別 A(ClassA)相依於類別 B(ClassB)。



由定義上，我們可以得出相依的關係可能存在於以下幾種情況。我們假設有一個類別 Class1 與 Class2 有相依的關係。並且給出這兩個類別的一些操作。



- ✓ 一個其他類別的區域變數(local variable)。

```
public class  Class1{
      public void method1()   {
            Class2 p = new Class2();
            // 其他敘述
            p.doSomthing();
      }
}
```

- ✓ 參照(has a reference)到其他的物件。這是類似於上圖的情形。
- ✓ 在操作的參數中使用到其他類別。

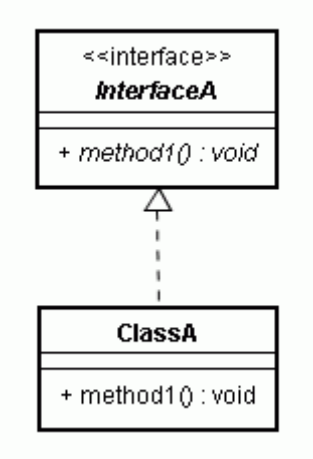
```
public class  Class1{
      public void method1(Class2 p) {
            //
            p.doSomthing();
      }
}
```

✓ 使用到一個類別的 static method。

```
public class  Class1 {
      public void method1()   {
            //
            Class2.staticMethod();
      }
}
```

3.6. 實現化(Realization)關係

實現化關係是用來表達一類別之行爲是由另一類別來描述定義的。 在 Java 中， 它是用 implements 來表示。 並且， 被實現的類別型態一定是介面(interface) 型態， 而不是類別(class)型態。




```
public class  ClassA implements interfaceA {
      // 屬性
      public void method1(){
            //..
      }
}
```

4. 循序圖