個人記帳行事曆

Object-Oriented Analysis and Design

Homework #7106598038 資工碩一 李柏霖

106598048 資工碩一 郭士銓

106598053 資工碩一 陳韻文

**目錄**

[**1. Requirement Document 1**](#_Toc517713674)

[1.1 Change History 1](#_Toc517713675)

[1.2 Problem Statement 3](#_Toc517713676)

[1.3. System Context Diagram(\*New) 4](#_Toc517713677)

[1.4. Summary of System Features(\*New) 4](#_Toc517713678)

[1.5. Use Case Diagram(\*New) 5](#_Toc517713679)

[1.6. Use Cases 6](#_Toc517713680)

[**1.6.1. CRUD 行程(\*New) 6**](#_Toc517713681)

[**1.6.2. CRUD金錢花費(\*New) 9**](#_Toc517713682)

[**1.6.3. 查看每月花費明細 11**](#_Toc517713683)

[**1.6.4. 匯出花費明細報表 12**](#_Toc517713684)

[**1.6.5. 查看該日行事曆 13**](#_Toc517713685)

[**1.6.6. 查看該日花費紀錄 14**](#_Toc517713686)

[1.7. Non-functional Requirements and Constraints 15](#_Toc517713687)

[1.8. Glossary 16](#_Toc517713688)

[1.9. The Development Language 16](#_Toc517713689)

[**2. Domain model 17**](#_Toc517713690)

[2.1. Domain class diagram showing only concepts(\*New) 17](#_Toc517713691)

[2.2. Add associations (\*New) 18](#_Toc517713692)

[2.3. Add attributes(\*New) 20](#_Toc517713693)

[**3. Design 21**](#_Toc517713694)

[3.1. Logical Architecture 21](#_Toc517713695)

[3.2. Use-Case Realizations with GRASP Patterns 22](#_Toc517713696)

[**3.2.1. System Sequence Diagram(\*New) CRUD行程 22**](#_Toc517713697)

[**3.2.2. Contract(\*New) 26**](#_Toc517713698)

[**3.2.3. Sequence Diagram(\*New) 33**](#_Toc517713699)

[3.3. Design Class Model 48](#_Toc517713700)

[**4. Implementation Class Model 49**](#_Toc517713701)

[4.1. Implementation Class Diagram 49](#_Toc517713702)

[4.2. Implementation Class Model and Design Class Model Difference 50](#_Toc517713703)

[**4.2.1. Comparison with Design and Implementation Class 50**](#_Toc517713704)

[**4.2.2. Summary of Implementation of Class/Method Changed 52**](#_Toc517713705)

[4.3. Calculate Line of Code 53](#_Toc517713706)

[**4.3.1. Line of Code of Classes 53**](#_Toc517713707)

[**5. Programming 54**](#_Toc517713708)

[5.1. Snapshot of System Execution 54](#_Toc517713709)

[5.2. Source Code Listing 62](#_Toc517713710)

[**6. Unit Testing 128**](#_Toc517713711)

[6.1. Snapshot of Testing Result 128](#_Toc517713712)

[6.2. Unit Testing Code Listing 129](#_Toc517713713)

[**7. Measurement 169**](#_Toc517713714)

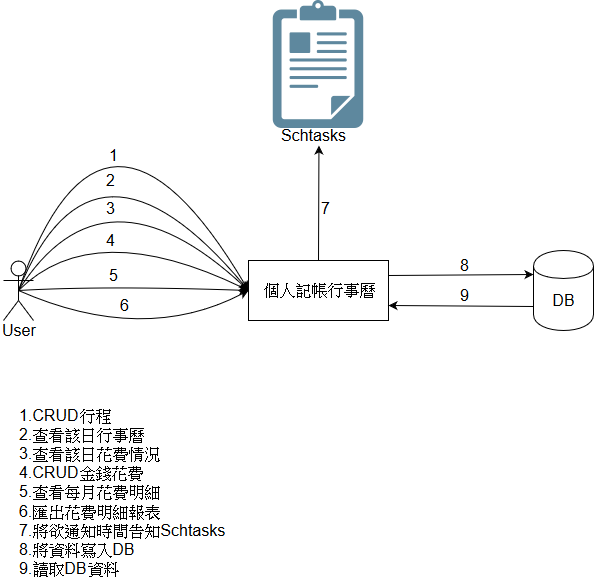
1. **Requirement Document**

## Change History

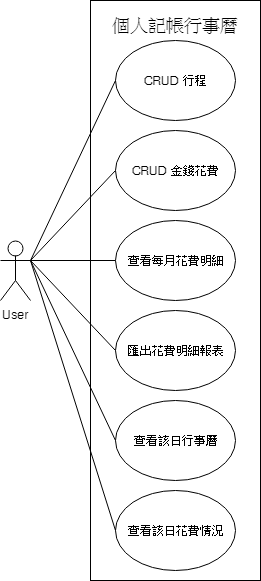
|  |  |  |
| --- | --- | --- |
| **Revision** | **Description** | **Date** |
| **Iteration I** | | |
| **1** | Problem statement. | Mar 9, 2018 |
| **2** | Problem Statement refinement. System Context Diagram.  Summary of System Features. Use Case Diagram.  Use Cases.  Non-functional Requirements and Constraints.  Glossary.  Software Environments refinement. | Mar 21, 2018 |
| **3** | Domain Model.  Add association.  Add attributes.  Glossary refinement. | Apr 2, 2018 |
| **4** | Logical Architecture.  System Sequence Diagram.  Contract.  Sequence Diagram.  Class Diagram.  System Context Diagram refinement.  Summary of System Features refinement.  Use Case refinement.  Use Case Diagram refinement.  Domain Model refinement.  Measurement refinement. | May 3, 2018 |
| **5** | Snapshots of testing result.  Unit testing code listing.  System Sequence Diagram refinement.  Sequence Diagram refinement.  Use Case refinement (extension part).  Design Class Diagram refinement.  System Context Diagram refinement.  Domain Model refinement.  Implement FEA-01、FEA-02、FEA-05、FEA-06. | May 11, 2018 |
| **Iteration II** | | |
| **6** | System Context Diagram refinement.  Use Case refinement (extension part).  Domain Model refinement. | May 28, 2018 |
| **7** | Summary of System Features refinement.  Use Case Diagram refinement.  Use Case refinement.  System Sequence Diagram refinement.  Contract refinement.  Sequence Diagram refinement.  Unit Testing Code Listing refinement.  Implement FEA-01、FEA-03、FEA-04.  Line of code of classes refinement  Source code listing refinement、  Comparison with design and implementation class refinement  Summary of Implementation and design refinement | Jun 27, 2018 |

## Problem Statement

個人記帳行事曆為一個結合記帳及行事曆功能的電腦桌面應用程式。由於現代人事務繁忙，容易遺忘代辦事項；或是想要規劃行程，卻因為事務過多而不知該如何下手；又或是想要管理金錢支出，卻總是無法清楚得知金錢流向，因此我們決定結合這幾項現代人最容易遇到的問題，提出一個良好的方案。使用者透過個人記帳行事曆，藉由行事曆的介面輸入代辦事項、提醒時間等資訊，當提醒時間到達時即會跳出視窗通知，幫助上班族、學生及擁有大量事務的使用者記憶大量的代辦事項及行程；此外也能夠透過記帳介面輸入花費事項、金額等資訊，顯示當日總花費，使需要金錢規劃的使用者更能清楚花費的種類。此程式可融入日常生活中，不僅是在工作場合，或是在學校等地方皆能方便使用，幫助使用者提升自我管理能力。

1. **System Context Diagram(\*New)**
2. **Summary of System Features(\*New)**

|  |  |
| --- | --- |
| **Feature ID** | **Description** |
| FEA-01 | CRUD行程 |
| FEA-02 | CRUD金錢花費 |
| FEA-03 | 查看每月花費明細 |
| FEA-04 | 匯出花費明細報表 |
| FEA-05 | 查看該日行事曆 |
| FEA-06 | 查看該日花費紀錄 |

1. **Use Case Diagram(\*New)**
   1. **Use Cases**
2. **CRUD 行程(\*New)**

|  |  |
| --- | --- |
| **Feature ID** | FEA-01 |
| **Scope** | 個人記帳行事曆 |
| **Level** | User-goal |
| **Primary Actor** | User |
| **Stakeholders and Interests** | - User:欲規劃個人行程，並期望記錄在電腦中方便檢視 |
| **Preconditions** | 1. 使用者必須使用電腦 2. 使用者必須事先下載本系統 |
| **Success Guarantee** | - 使用者成功編輯個人行程 |
| **Main Success Scenario** | 1. 使用者進入本系統 2. 使用者選擇欲新增行程的日期 3. 使用者編輯當日行程   3.1 使用者輸入行程相關資訊以新增當日行程  3.2 使用者編輯已存在的行程  3.3 使用者刪除已存在行程   1. 系統更新行程資訊 |
| **Extensions** | 1a.使用者無法順利開啟本系統  1.使用者需安裝Java  1a.使用者需升級Java版本  3a.系統無法顯示當日行程資訊   1. 系統回報錯誤訊息給使用者 2. 使用者重新啟動本系統   3.1a.使用者新增行程資訊時輸入多於20個字   1. 系統回報錯誤訊息給使用者 2. 使用者刪減行程資訊的字數至20字內   3.1b.使用者新增行程資訊時未輸入值   1. 系統回報錯誤訊息給使用者 2. 使用者填入行程資訊   3.1c.使用者新增行程時間時輸入非數字字元   * + - 1. 系統回報錯誤訊息給使用者       2. 使用者輸入數字   3.1d.使用者新增行程時間時輸入多於4位數字   1. 系統回報錯誤訊息給使用者 2. 使用者刪減數字至4位內   3.1e.使用者新增行程時間時未輸入值   1. 系統回報錯誤訊息給使用者 2. 使用者填入行程時間   3.1g.使用者新增行程時開啟行程提醒   1. 系統於該行程時間回報通知訊息給使用者   3.2a.使用者編輯行程資訊時輸入多於20個字   1. 系統回報錯誤訊息給使用者 2. 使用者刪減行程資訊的字數至20字內   3.2b.使用者編輯行程資訊時未輸入值   1. 系統回報錯誤訊息給使用者 2. 使用者填入行程資訊   3.2c.使用者編輯行程時間時輸入非數字字元   1. 系統回報錯誤訊息給使用者 2. 使用者輸入數字   3.2d.使用者編輯行程時間時輸入多於4位數字   1. 系統回報錯誤訊息給使用者 2. 使用者刪減數字至4位內   3.2e.使用者編輯行程時間時未輸入值   1. 系統回報錯誤訊息給使用者 2. 使用者填入行程時間   3.2g.使用者編輯行程時開啟行程提醒   1. 系統於該行程時間回報通知訊息給使用者   3.2h. 使用者編輯行程時關閉行程提醒   1. 系統將不會於該行程時間回報通知訊息給使用者   3.3a.使用者刪除行程資訊失敗   1. 使用者重新啟動本系統 2. 使用者刪除行程資訊   使用者持續執行1、2直到行程 資訊刪除成功 |
| **Special Requirements** | 1. 每筆行程資訊最多輸入20字 2. 每筆行程時間最多輸入4字 |
| **Technology and Data Variations List** | - 輸入行程資訊的文字必須為半形 |
| **Frequency of Occurrence** | 經常 |
| **Open Issues** | 無 |

* + - * 1. **CRUD金錢花費(\*New)**

|  |  |
| --- | --- |
| **Feature ID** | FEA-03 |
| **Scope** | 個人記帳行事曆 |
| **Level** | User-goal |
| **Primary Actor** | User |
| **Stakeholders and Interests** | - User: 欲記錄個人金錢花費，並期望記錄在電腦中方便檢視 |
| **Preconditions** | 1. 使用者必須使用電腦 2. 使用者必須事先下載本系統 |
| **Success Guarantee** | -使用者成功編輯金錢花費紀錄 |
| **Main Success Scenario** | 1. 使用者進入本系統 2. 使用者選擇欲記錄花費的日期 3. 使用者編輯當日花費資訊   3.1 使用者輸入當日花費相關內容以新增花費紀錄  3.2 使用者編輯已存在的花費紀錄  3.3 使用者刪除已存在的花費紀錄   1. 系統更新花費紀錄資訊 |
| **Extensions** | 1a.使用者無法順利開啟本系統  1.使用者需安裝Java  1a.使用者需升級Java版本  3a.系統無法顯示當日花費紀錄   1. 系統回報錯誤訊息給使用者 2. 使用者重新啟動本系統   3.1a. 使用者新增花費紀錄時輸入多於20字   1. 系統回報錯誤訊息給使用者 2. 使用者刪減花費紀錄的字數   3.1b. 使用者新增花費紀錄時未輸入值   1. 系統回報錯誤訊息給使用者 2. 使用者填入花費紀錄   3.1c.使用者新增花費費用時輸入非數字字元  1. 系統回報錯誤訊息給使用者  2. 使用者輸入數字  3.1d.使用者新增花費費用時輸入多於12字   1. 系統回報錯誤訊息給使用者 2. 使用者刪減花費費用的字數至12字   3.1e.使用者新增花費費用時未輸入值   1. 系統回報錯誤訊息給使用者 2. 使用者填入花費費用   3.2a. 使用者編輯花費紀錄時輸入多於20字   1. 系統回報錯誤訊息給使用者 2. 使用者刪減花費紀錄的字數   3.2b. 使用者編輯花費紀錄時未輸入值   1. 系統回報錯誤訊息給使用者 2. 使用者填入花費紀錄   3.2c.使用者編輯花費費用時輸入非數字字元  1. 系統回報錯誤訊息給使用者  2. 使用者輸入數字  3.2d.使用者編輯花費費用時輸入多於12字   1. 系統回報錯誤訊息給使用者 2. 使用者刪減花費費用的字數至12字   3.2e.使用者編輯花費費用時未輸入值   1. 系統回報錯誤訊息給使用者 2. 使用者填入花費費用   3.3a.使用者刪除花費紀錄失敗   1. 使用者重新啟動本系統 2. 使用者刪除花費紀錄   使用者持續執行1、2直到花費 紀錄刪除成功 |
| **Special Requirements** | 1. 每筆花費紀錄最多輸入20字  2. 花費費用最多輸入12位數字 |
| **Technology and Data Variations List** | 1. 輸入花費紀錄的文字必須為半形 2. 輸入花費費用的字元必須為數字 |
| **Frequency of Occurrence** | 經常 |
| **Open Issues** | 無 |

* + 1. **查看每月花費明細**

|  |  |
| --- | --- |
| **Feature ID** | FEA-04 |
| **Scope** | 個人記帳行事曆 |
| **Level** | User-goal |
| **Primary Actor** | User |
| **Stakeholders and Interests** | - User:使用者每日紀錄花費狀況，期望系統可顯示月份的總花費情況 |
| **Preconditions** | 1. 使用者必須使用電腦 2. 使用者必須事先下載本系統 3. 使用者必須存在記帳紀錄 |
| **Success Guarantee** | - 系統成功顯示當月花費明細 |
| **Main Success Scenario** | 1. 使用者進入本系統 2. 使月者選擇欲顯示的月份 3. 系統回傳給使用者此月份消費明細及總花費狀況 |
| **Extensions** | 1a.請參考FEA-01以順利進入系統  2a.使用者無法選擇月份  1. 系統回報錯誤訊息給使用者  2. 使用者重新啟動本系統  3a.系統未正確顯示明細   1. 系統回報錯誤訊息給使用者 2. 使用者重新啟動本系統   3b.系統未正確顯示曾花費品項   1. 使用者refresh明細資訊 2. 使用者重新啟動本系統   3c.系統未正確顯示總花費   1. 系統回報錯誤訊息給使用者 2. 使用者重新啟動本系統   3d.系統計算總花費錯誤  1. 使用者refresh明細資訊  2. 使用者重新啟動本系統 |
| **Special Requirements** | 無 |
| **Technology and Data Variations List** | 無 |
| **Frequency of Occurrence** | 偶爾 |
| **Open Issues** | 無 |

1. **匯出花費明細報表**

|  |  |
| --- | --- |
| **Feature ID** | FEA-05 |
| **Scope** | 個人記帳行事曆 |
| **Level** | User-goal |
| **Primary Actor** | User |
| **Stakeholders and Interests** | - User:記錄每日花費，並期望系統能輸出該月之總花費及明細報表 |
| **Preconditions** | 1. 使用者必須使用電腦  2. 使用者必須事先下載本系統  3. 使用者必須要記錄每日花費 |
| **Success Guarantee** | - 系統成功輸出該月總花費及明細報表 |
| **Main Success Scenario** | 1. 使用者進入本系統 2. 使月者選擇要輸出的月份 3. 系統輸出該月總花費及明細報表 |
| **Extensions** | 1a.請參考FEA-01以順利進入系統  2a.使用者無法選擇要輸出的月份  1.系統回報錯誤訊息給使用者  2.使用者重新啟動本系統  3a.系統未正確輸出該月總花費及明細報表  1.系統回報錯誤訊息給使用者  2.使用者重新啟動本系統  3b.系統於輸出報表中未正確顯示花費明細  1.系統回報錯誤訊息給使用者  2.使用者重新啟動本系統  3c.系統於輸出報表中未正確顯示總花費  1.系統回報錯誤訊息給使用者  2.使用者重新啟動本系統 |
| **Special Requirements** | 無 |
| **Technology and Data Variations List** | 無 |
| **Frequency of Occurrence** | 偶爾 |
| **Open Issues** | 無 |

**查看該日行事曆**

|  |  |
| --- | --- |
| **Feature ID** | FEA-06 |
| **Scope** | 個人記帳行事曆 |
| **Level** | User-goal |
| **Primary Actor** | User |
| **Stakeholders and Interests** | - User:記錄該日行程，並期望系統能顯示該日行程 |
| **Preconditions** | 1. 使用者必須使用電腦  2. 使用者必須事先下載本系統 |
| **Success Guarantee** | - 系統成功顯示該日行程 |
| **Main Success Scenario** | 1. 使用者進入本系統  2. 使月者選擇欲顯示的日期  3. 系統顯示該日行程 |
| **Extensions** | 1a.請參考FEA-01以順利進入系統  2a.使用者無法選擇日期  1.系統回報錯誤訊息給使用者  2.使用者重新啟動本系統  3a.系統未正確顯示該日行程   1. 系統回報錯誤訊息給使用者 2. 使用者重新啟動本系統 |
| **Special Requirements** | 無 |
| **Technology and Data Variations List** | 無 |
| **Frequency of Occurrence** | 頻繁 |
| **Open Issues** | 無 |

1. **查看該日花費紀錄**

|  |  |
| --- | --- |
| **Feature ID** | FEA-07 |
| **Scope** | 個人記帳行事曆 |
| **Level** | User-goal |
| **Primary Actor** | User |
| **Stakeholders and Interests** | - User:記錄該日花費紀錄，並期望系統能顯示該日花費紀錄 |
| **Preconditions** | 1. 使用者必須使用電腦  2. 使用者必須事先下載本系統 |
| **Success Guarantee** | - 系統成功顯示該日花費紀錄 |
| **Main Success Scenario** | 1. 使用者進入本系統  2. 使月者選擇欲顯示的日期  3. 系統顯示該日花費紀錄 |
| **Extensions** | 1a.請參考FEA-01以順利進入系統  2a.使用者無法選擇日期  1.系統回報錯誤訊息給使用者  2.使用者重新啟動本系統  3a.系統未正確顯示該日花費紀錄   1. 系統回報錯誤訊息給使用者 2. 使用者重新啟動本系統 |
| **Special Requirements** | 無 |
| **Technology and Data Variations List** | 無 |
| **Frequency of Occurrence** | 頻繁 |
| **Open Issues** | 無 |

* 1. **Non-functional Requirements and Constraints**

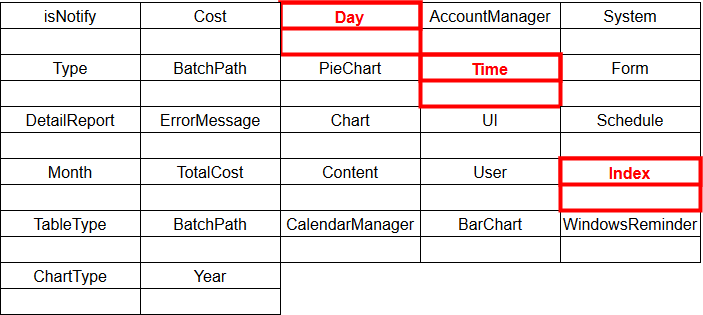
|  |  |  |
| --- | --- | --- |
| **NFR ID** | **Category** | **Description** |
| NFR-01 | Performance | 使用者每CRUD行程，系統需立即更新資料庫 |
| NFR-02 | Performance | 使用者每CRUD金錢花費紀錄，系統需立即更新資料庫 |
| NFR-03 | Performance | 系統須在5秒內回報錯誤訊息 |
| NFR-04 | Performance | 系統須在5秒內完成啟動 |
| NFR-05 | Reliability | 系統遭遇重大錯誤時需保持程式穩健 |

* 1. **Glossary**

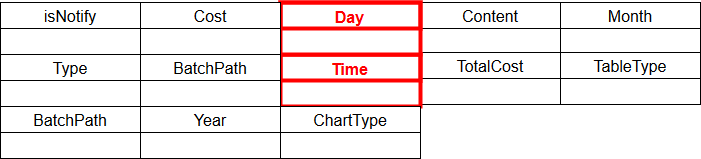
|  |  |
| --- | --- |
| **Item** | **Definition or Description** |
| Java | Java是一種廣泛使用的電腦程式設計語言，擁有跨平台、物件導向、泛型程式設計的特性，廣泛應用於企業級Web應用開發和行動應用開發。 |
| Identify noun phrase | Identify noun phrase為一產生Conceptual Model所使用的策略，主要於Use Case及Requirement Documents中獲取名詞，以作為domain model中的concepts。 |
| Error Message | Error Message為本系統使用的例外處理機制，當系統發生不可預期之錯誤時，將於畫面中顯示警告視窗。 |
| System | System在本文件中意指「個人記帳行事曆」。 |
| UI | UI是系統和用戶之間進行互動和資訊交換的媒介，它實現資訊的內部形式與人類可以接受形式之間的轉換。 |

* 1. **The Development Language**

本程式採用Java作為主要的開發語言。

1. **Domain model**
   1. **Domain class diagram showing only concepts(\*New)**

利用Identify noun phrase的方式找出 Use Case中出現的所有名詞，藉此定義Concepts：

**Attributes:**

**Role:**

Domain Model 中不適合出現使用者角色。

**Implement Construction:**

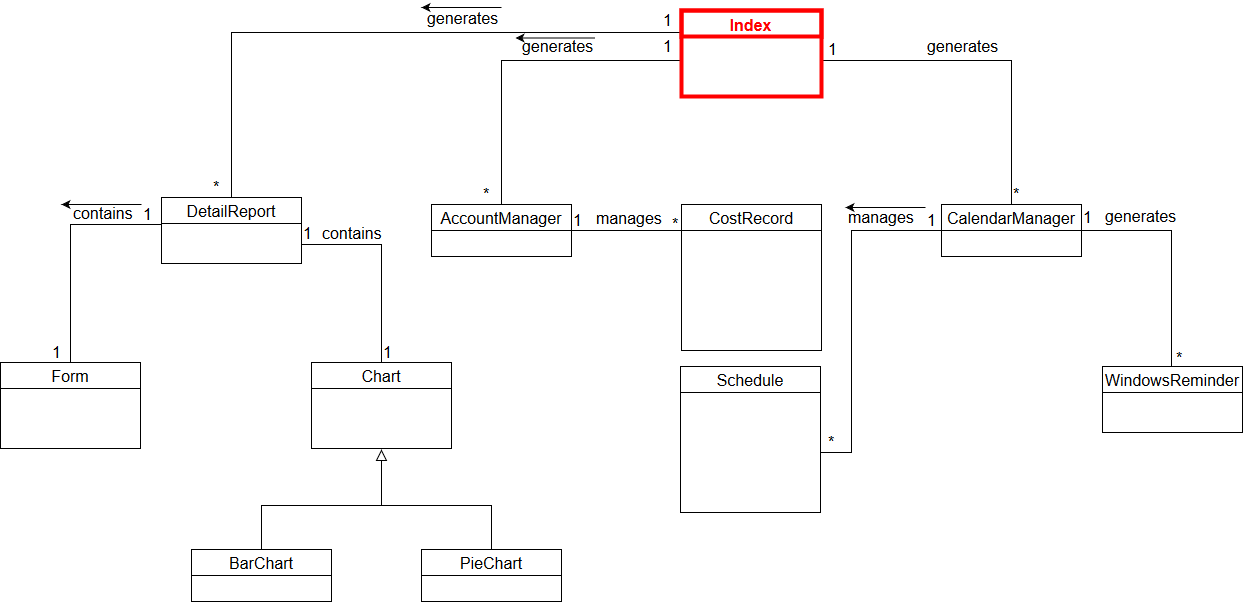
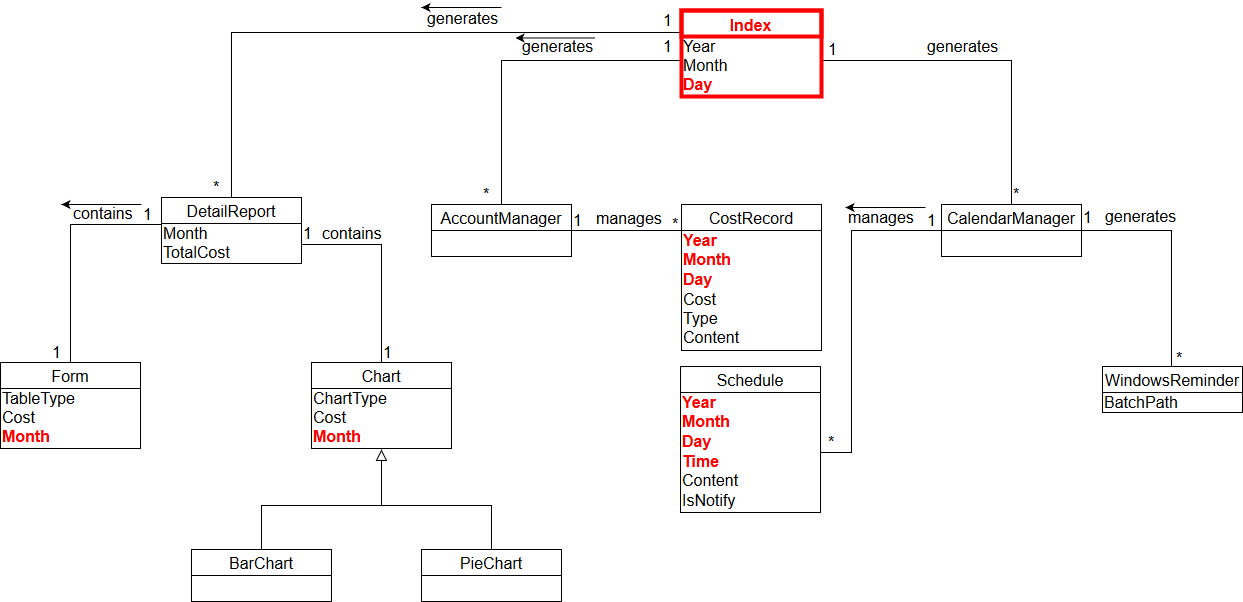
Error Message是一種處理例外的方式，為建構系統的基礎。

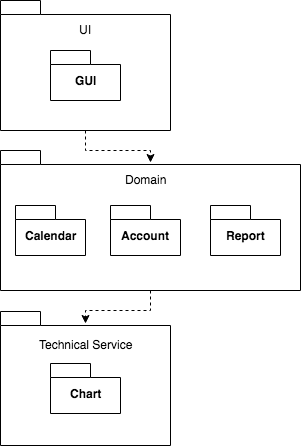
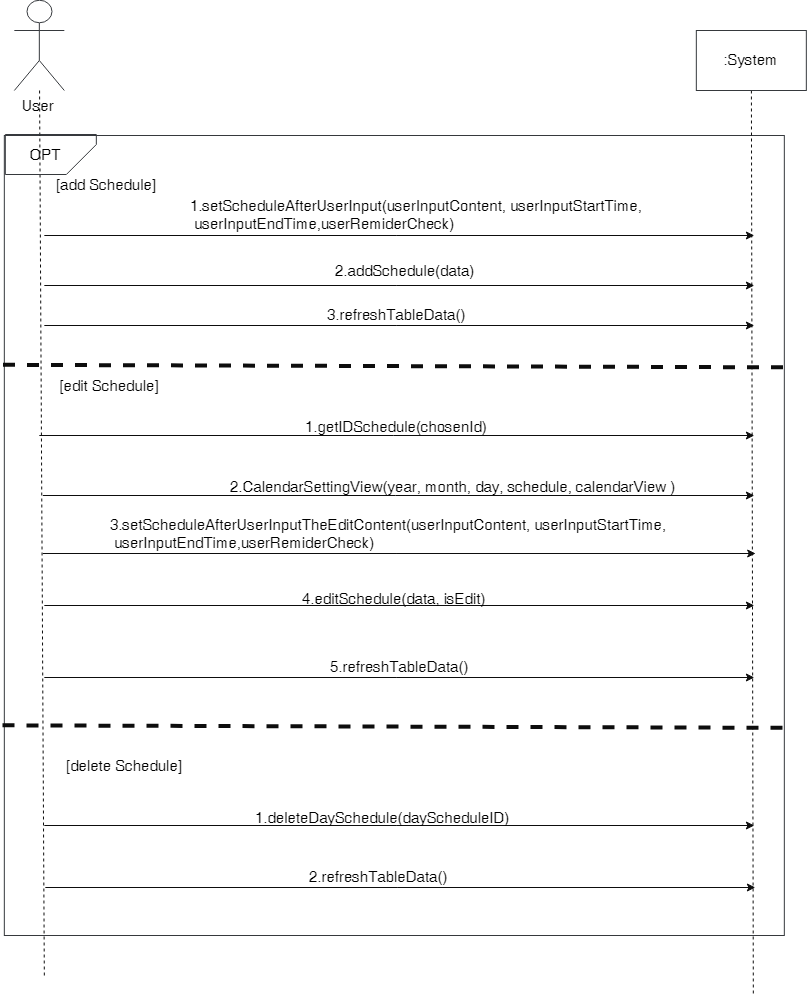
**Inappropriate Scope:**

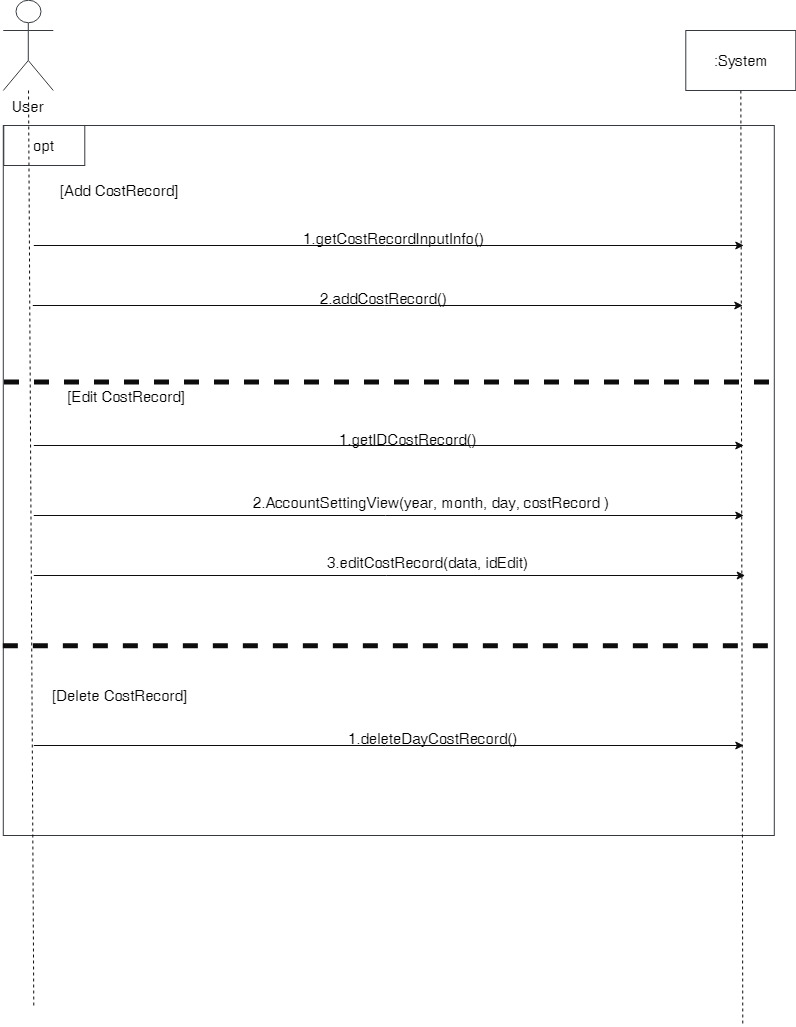
System所表示的範圍過於龐大，不適合直接呈現在Domain Model中，需要進行切割。

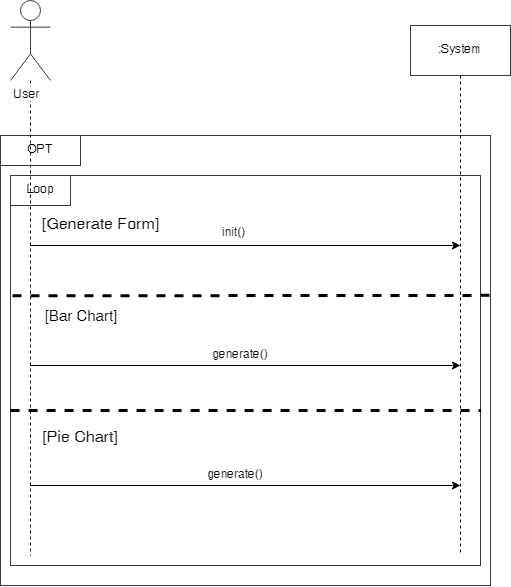
**Redundant:**

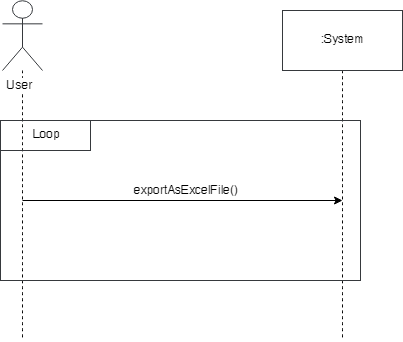
UI為系統的介面，不適合出現在 Domain Model中。

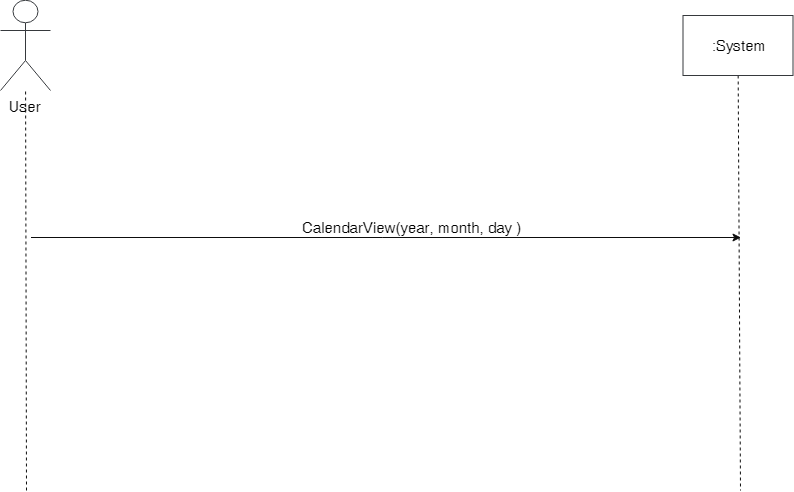
* 1. **Add associations (\*New)**
  2. **Add attributes(\*New)**

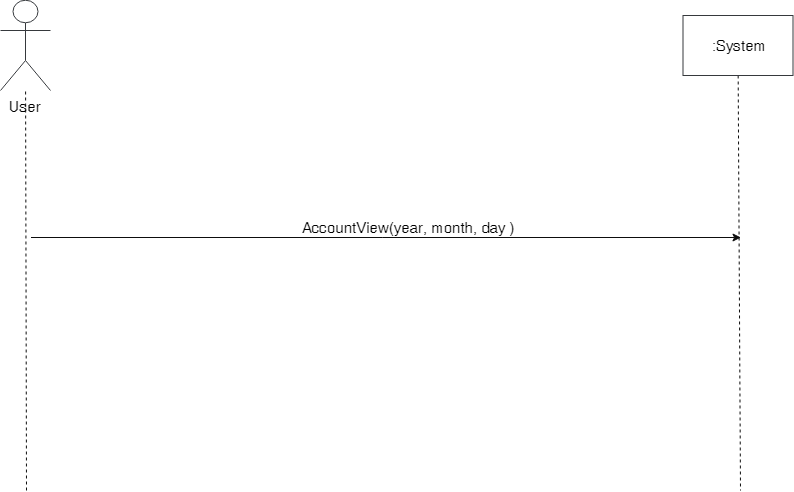
1. **Design**
   1. **Logical Architecture**
   2. **Use-Case Realizations with GRASP Patterns**
      1. **System Sequence Diagram(\*New)  
         CRUD行程**

**CRUD金錢花費**

**查看每月花費明細**

**匯出花費明細報表**

**查看該日行事曆**

**查看該日花費紀錄**

* + 1. **Contract(\*New)**
       1. **setScheduleAfterUserInput**

|  |  |
| --- | --- |
| **Operation** | setScheduleAfterUserInput(userInputContent, userInputStartTime, userInputEndTime, userReminderCheck) |
| **Cross Reference** | FEA-01 |
| **Preconditions** | None |
| **Postcondition** | 產生Schedule物件 |

* + - 1. **addSchedule**

|  |  |
| --- | --- |
| **Operation** | addSchedule(data) |
| **Cross Reference** | FEA-01 |
| **Preconditions** | 已取得使用者輸入之行程資訊 |
| **Postconditions** | 1. 新增一筆行程資訊至資料庫 2. 使用者可於CalendarView查看新增後的行程資訊 |

* + - 1. **refreshTableData**

|  |  |
| --- | --- |
| **Operation** | refreshTableData() |
| **Cross Reference** | FEA-05 |
| **Preconditions** | 1. Schedule物件已建立 2. 使用者進行任一CRUD行事曆的操作 |
| **Postconditions** | 使用者可於CalendarView查看最新的行程資訊 |

* + - 1. **getIdSchedule**

|  |  |
| --- | --- |
| **Operation** | getIdSchedule(dayNumID) |
| **Cross Reference** | FEA-01 |
| **Preconditions** | 1. 已取得使用者輸入之ID號碼 2. 資料庫存在該ID之資料 |
| **Postconditions** | 1. 從資料庫取得建立Schedule物件所需之attribute 2. 建立Schedule物件 |

* + - 1. **CalendarSettingView**

|  |  |
| --- | --- |
| **Operation** | CalendarSettingView(year, month, day, Schedule, calendarView) |
| **Cross Reference** | FEA-05 |
| **Preconditions** | Schedule物件已建立 |
| **Postconditions** | 1. 建立CalendarSettingView物件 2. 顯示欲編輯的行程資訊 |

* + - 1. **setScheduleAfterUserInputTheEditContent**

|  |  |
| --- | --- |
| **Operation** | setScheduleAfterUserInputTheEditContent(userInputContent, userInputStartTime, userInputEndTime, userReminderCheck) |
| **Cross Reference** | FEA-01 |
| **Preconditions** | 1. Schedule物件已建立 2. 已取得使用者編輯後的資訊 |
| **Postconditions** | Schedule物件內的資訊已為編輯後的資訊 |

* + - 1. **editSchedule**

|  |  |
| --- | --- |
| **Operation** | editSchedule(schedule, ID) |
| **Cross Reference** | FEA-01 |
| **Preconditions** | 1. 已取得使用者輸入之行程資訊 2. 已取得使用者輸入之ID |
| **Postconditions** | 1. 將編輯後的資訊存至資料庫  2. 使用者可於CalendarView查看編輯後的行程資訊 |

* + - 1. **deleteDaySchedule**

|  |  |
| --- | --- |
| **Operation** | deleteDaySchedule(dayScheduleID) |
| **Cross Reference** | FEA-01 |
| **Preconditions** | 已取得使用者輸入之dayScheduleID資訊 |
| **Postconditions** | 1.將資料庫中相對應的行程資訊刪除  2.使用者可於CalendarView查看修改後的行程資訊 |

* + - 1. **getCostRecordInputInfo**

|  |  |
| --- | --- |
| **Operation** | getCostRecordInputInfo() |
| **Cross Reference** | FEA-02 |
| **Preconditions** | None |
| **Postconditions** | 1. 從資料庫取得建立CostRecord物件所需之attribute 2. 建立CostRecord物件 |

* + - 1. **addCostRecord**

|  |  |
| --- | --- |
| **Operation** | addCost(data, isEdit) |
| **Cross Reference** | FEA-02 |
| **Preconditions** | 已取得使用者輸入之花費資訊 |
| **Postconditions** | 1. 新增一筆花費資訊至資料庫  2. 使用者可於AccountView查看新增後的花費資訊 |

* + - 1. **getIDCostRecord**

|  |  |
| --- | --- |
| **Operation** | getIDCostRecord(id) |
| **Cross Reference** | FEA-02 |
| **Preconditions** | 1. 已取得使用者輸入之ID號碼 2. 資料庫存在該ID之資料 |
| **Postconditions** | 1. 從資料庫取得建立CostRecord物件所需之attribute 2. 建立CostRecord物件 |

* + - 1. **AccountSettingView**

|  |  |
| --- | --- |
| **Operation** | AccountSettingView(year, month, day) |
| **Cross Reference** | FEA-06 |
| **Preconditions** | CostRecord物件已建立 |
| **Postconditions** | 1. 建立AccountSettingView物件 2. 顯示欲編輯的金錢資訊 |

* + - 1. **editCostRecord**

|  |  |
| --- | --- |
| **Operation** | editCostRecord(data, isEdit) |
| **Cross Reference** | FEA-02 |
| **Preconditions** | 1. 已取得使用者輸入之金錢資訊 2. 已取得使用者輸入之ID |
| **Postconditions** | 1. 將編輯後的資訊存至資料庫  2. 使用者可於AccountView查看編輯後的金錢資訊 |

* + - 1. **deleteDayCostRecord**

|  |  |
| --- | --- |
| **Operation** | deleteDayCostRecord(id) |
| **Cross Reference** | FEA-02 |
| **Preconditions** | 已取得使用者輸入之id資訊 |
| **Postconditions** | 1.將資料庫中相對應的金錢資訊刪除  2.使用者可於AccountView查看修改後的行程資訊 |

* + - 1. **DetailReportView : init**

|  |  |
| --- | --- |
| **Operation** | init() |
| **Cross Reference** | FEA-03 |
| **Preconditions** | None |
| **Postconditions** | 該月報表資訊成功匯出到ReportExport資料夾內 |

* + - 1. **BarChart : generate**

|  |  |
| --- | --- |
| **Operation** | generate() |
| **Cross Reference** | FEA-03 |
| **Preconditions** | None |
| **Postconditions** | 該月花費資訊成功以bar chart形式呈現在畫面中。 |

* + - 1. **PieChart : generate**

|  |  |
| --- | --- |
| **Operation** | generate() |
| **Cross Reference** | FEA-03 |
| **Preconditions** | None |
| **Postconditions** | 該月花費資訊成功以pie chart形式呈現在畫面中。 |

* + - 1. **exportAsExcelFile**

|  |  |
| --- | --- |
| **Operation** | exportAsExcelFile() |
| **Cross Reference** | FEA-04 |
| **Preconditions** | None |
| **Postconditions** | 該月報表資訊成功匯出到ReportExport資料夾內 |

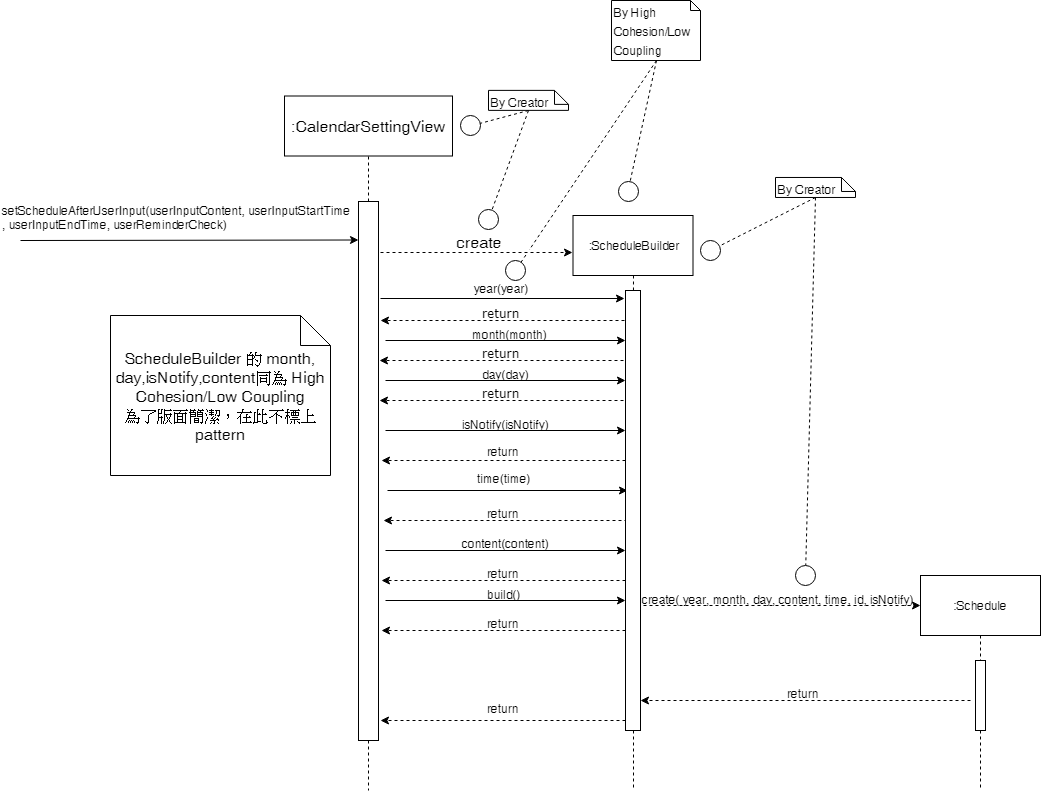
* + - 1. **CalendarView : init**

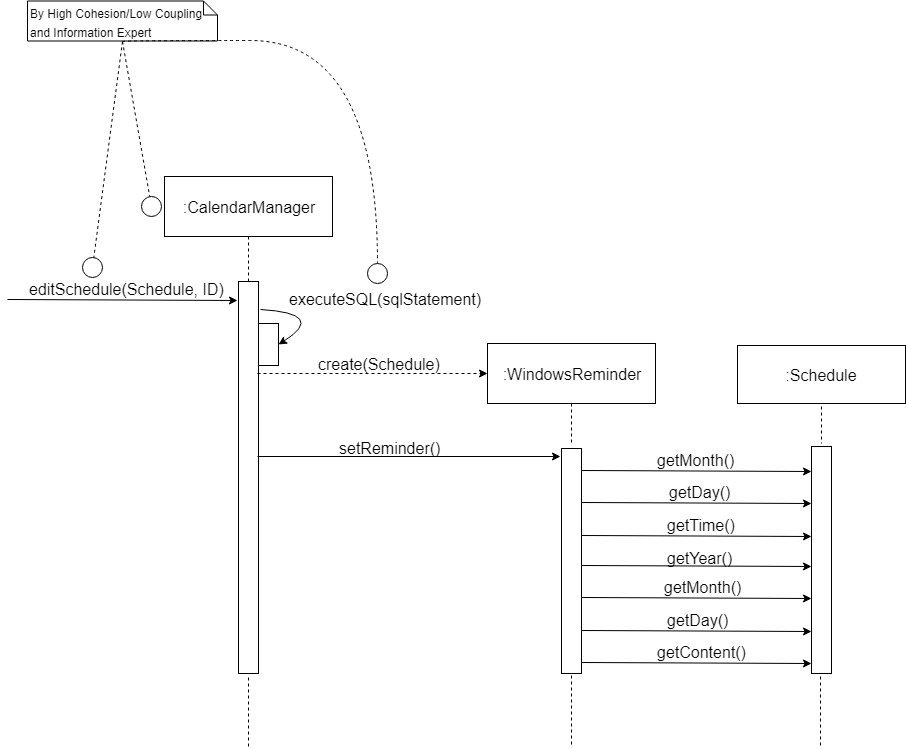
|  |  |
| --- | --- |
| **Operation** | generate() |
| **Cross Reference** | FEA-04 |
| **Preconditions** | None |
| **Postconditions** | 修改DetailReportView中的花費資訊 |

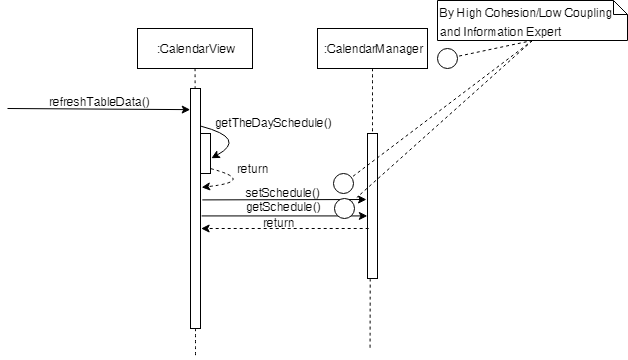
* + - 1. **AccountView : init**

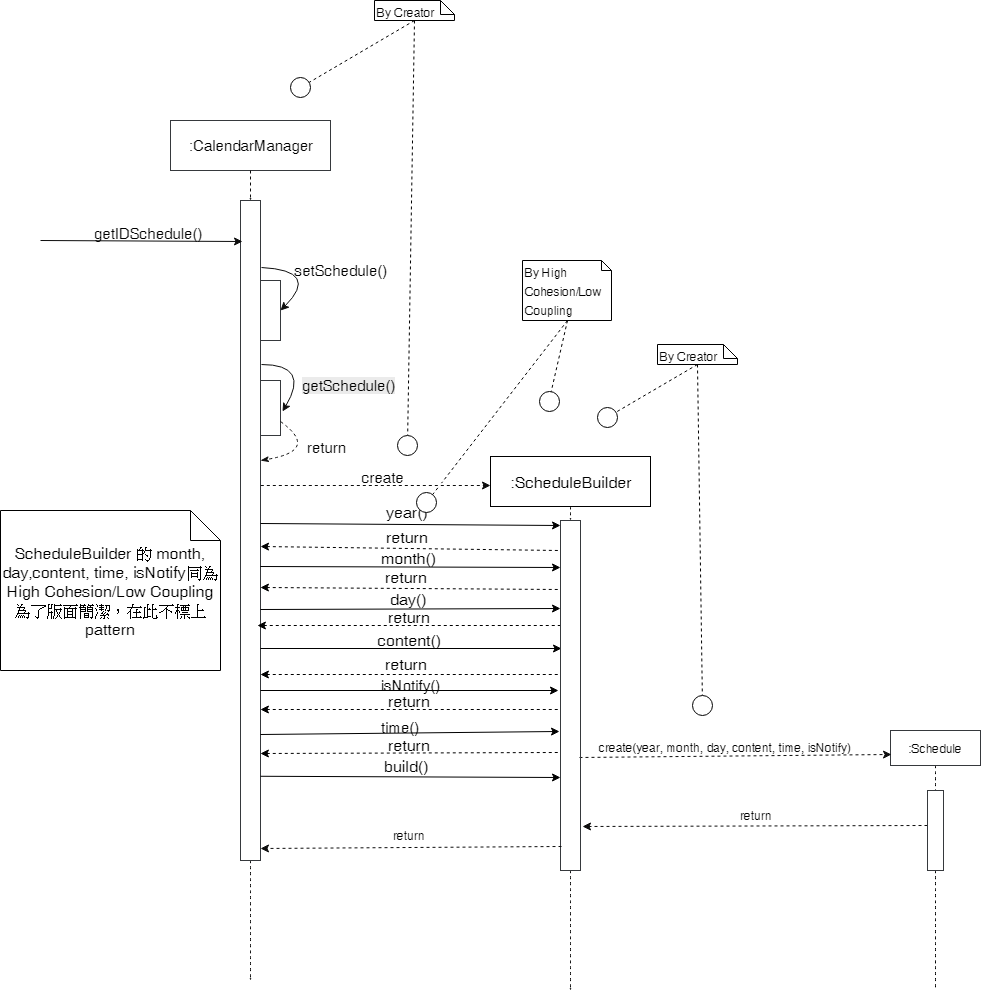
|  |  |
| --- | --- |
| **Operation** | generate() |
| **Cross Reference** | FEA-04 |
| **Preconditions** | None |
| **Postconditions** | 修改DetailReportView中的花費資訊 |

1. **Sequence Diagram(\*New)**

**setScheduleAfterUserInput**

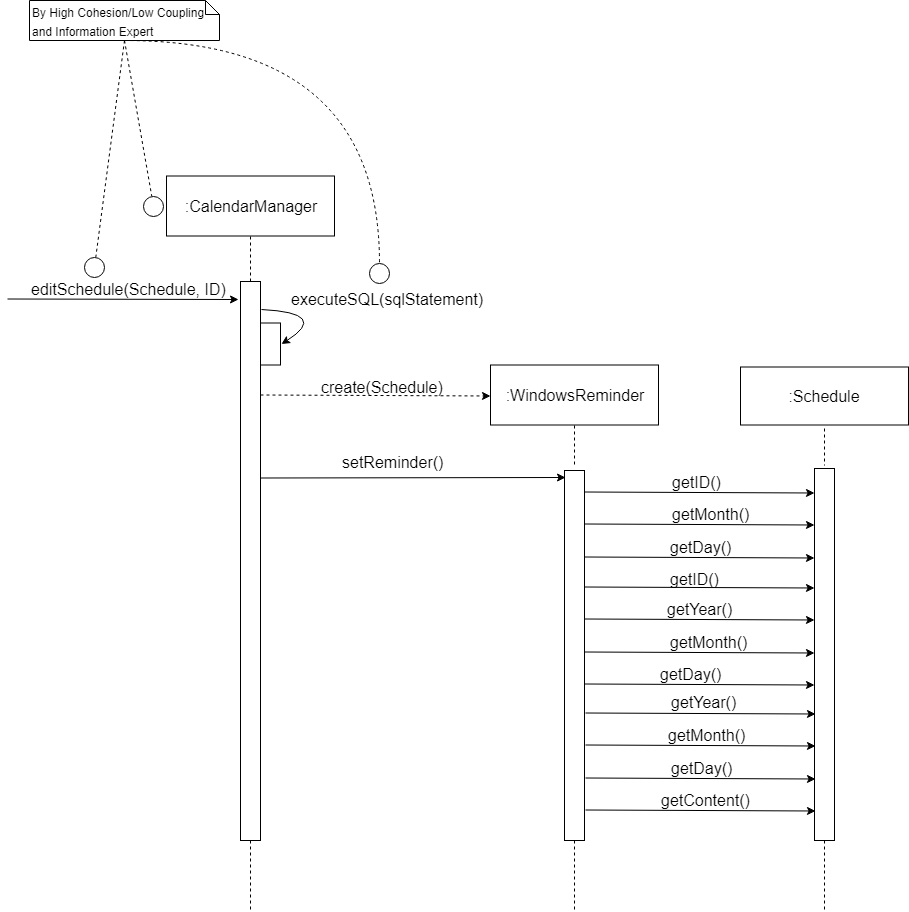
**addSchedule(data)**

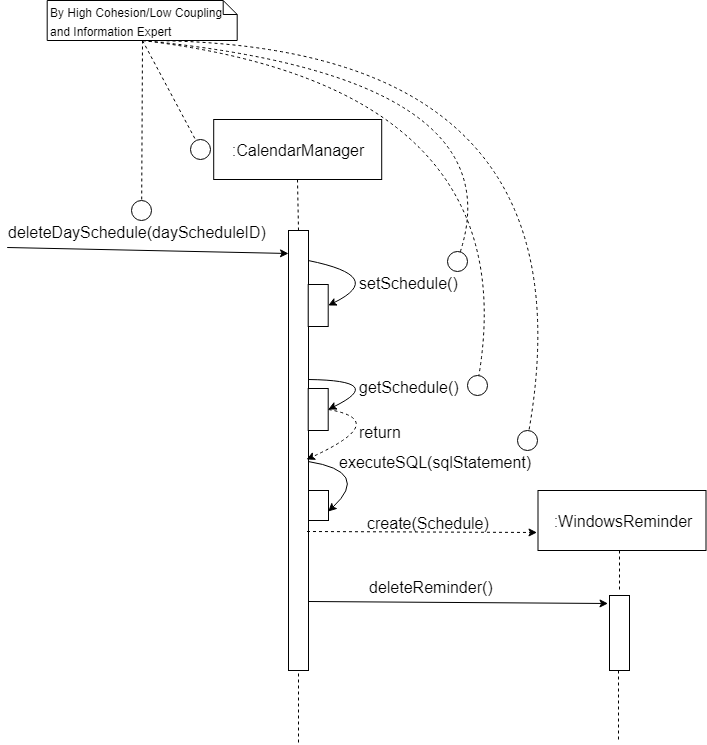
**refreshTableData**

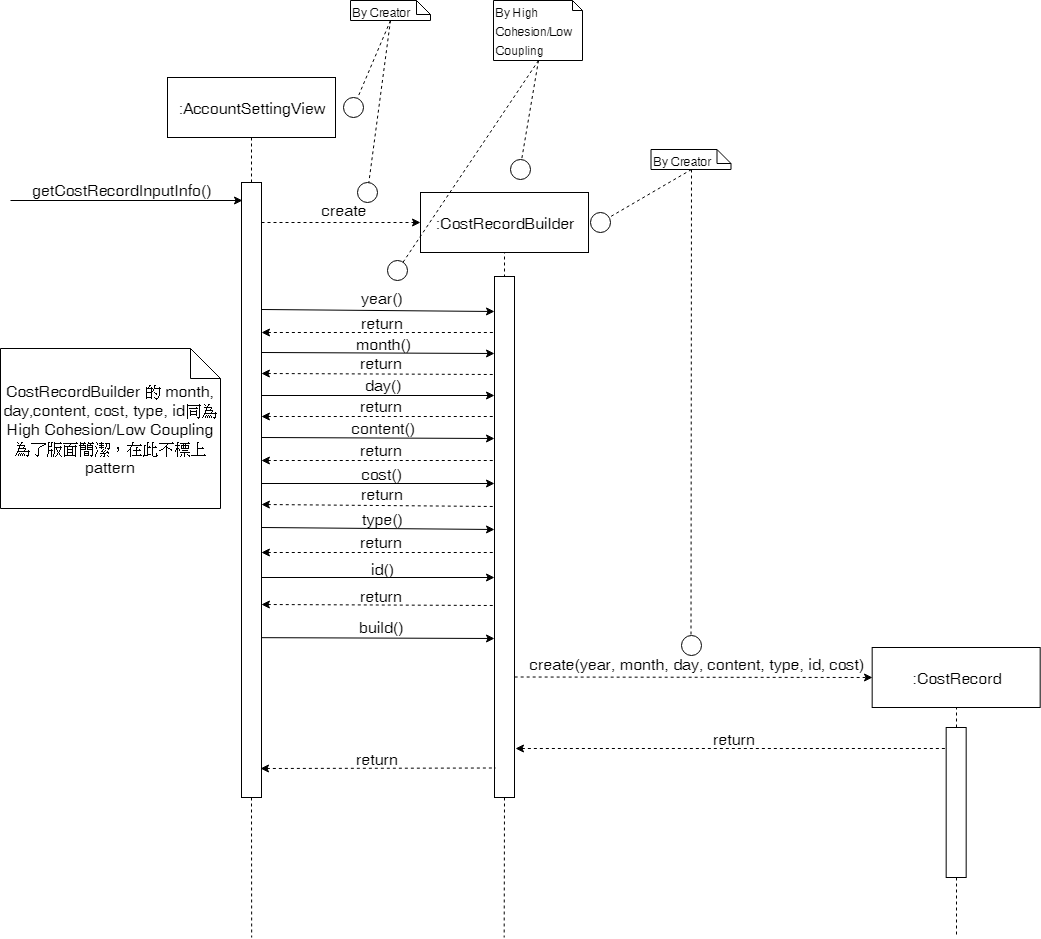
**getIDSchedule**

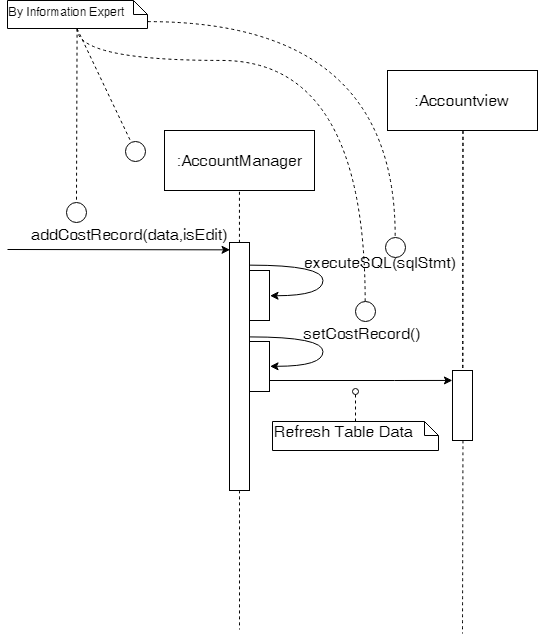
**CalendarSettingView(year, month, day, schedule, calendarView)**

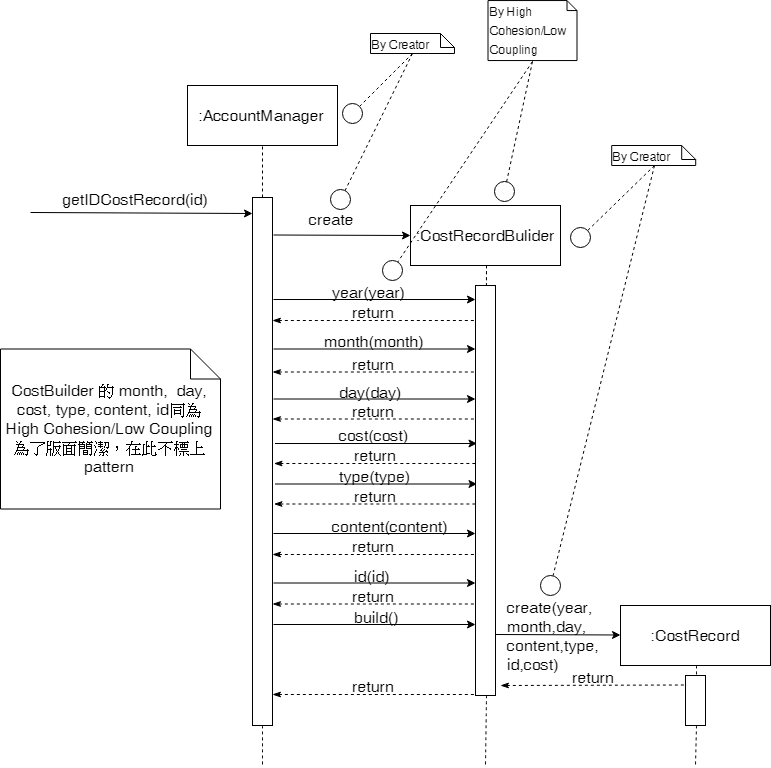
**setScheduleAfterUserInputTheEditContent(userInputContent, userInputStartTime, userInputEndTime, userReminderCheck)**

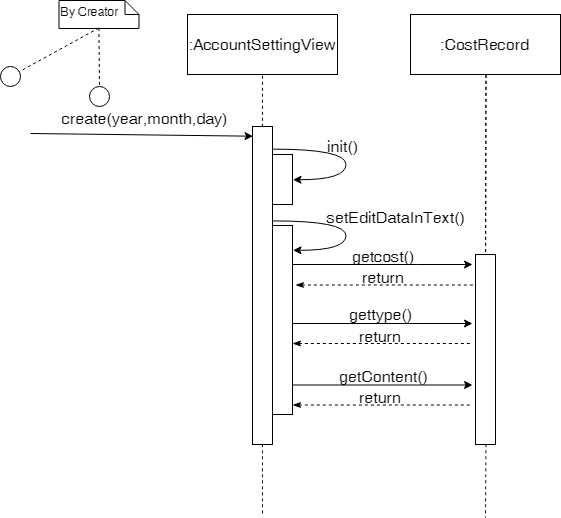
**editSchedule(Schedule, ID)**

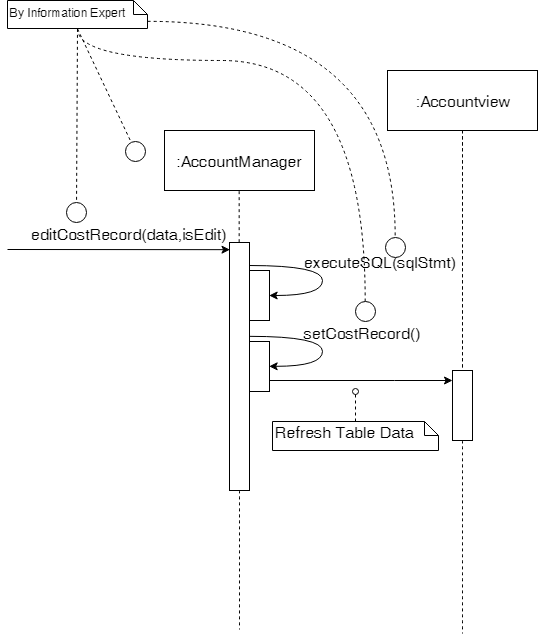
**deleteDaySchedule(dayScheduleID)**

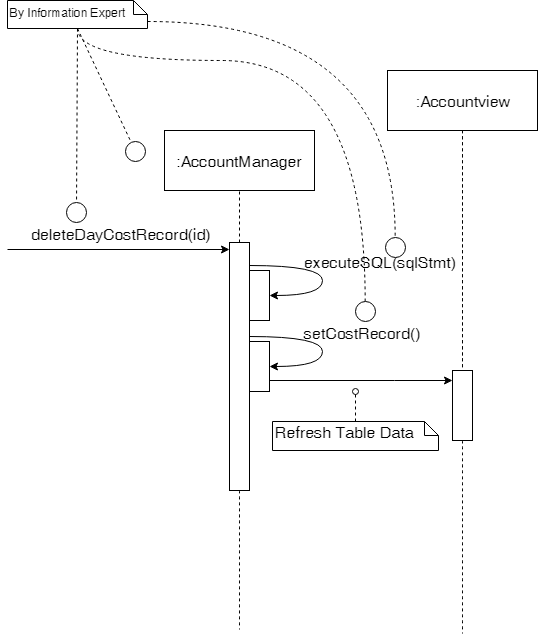
**getCostRecordInputInfo**

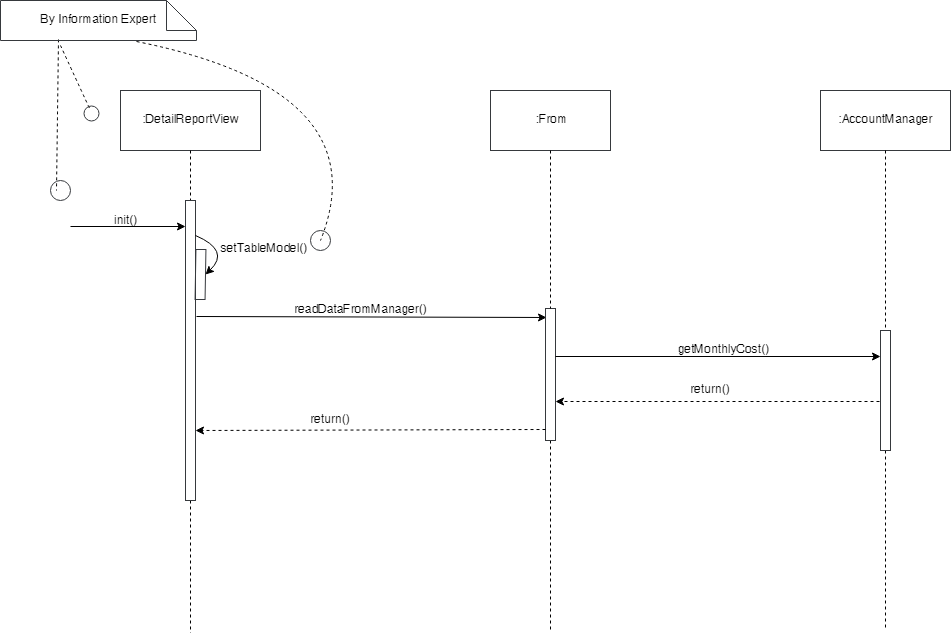
**addCostRecord(data, isEdit)**

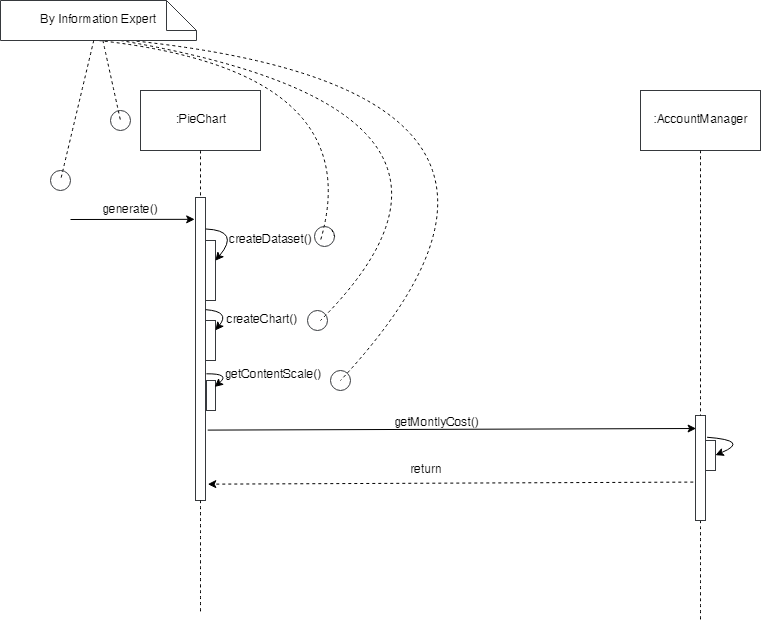
**getIDCostRecord(id)**

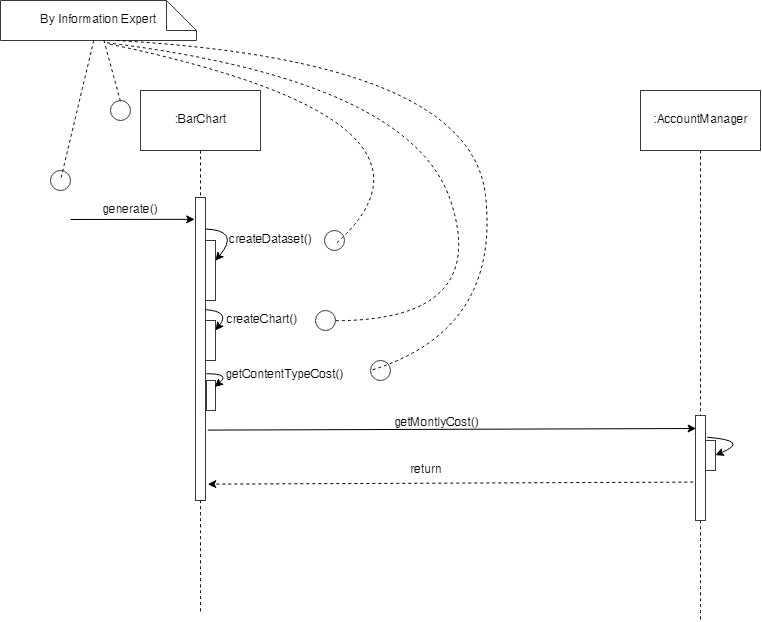
**AccountSettingView(year, month, day)**

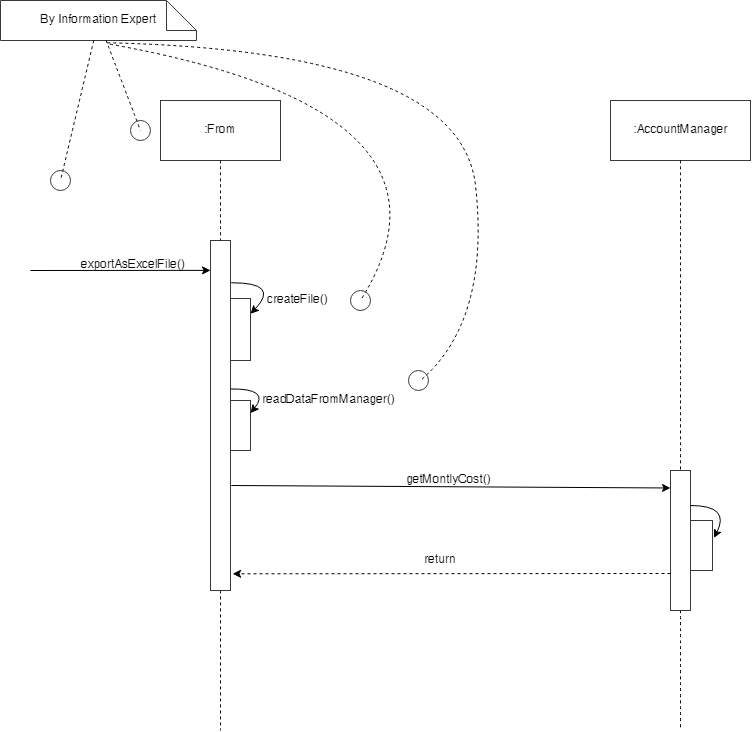
**editCostRecord(data, isEdit)**

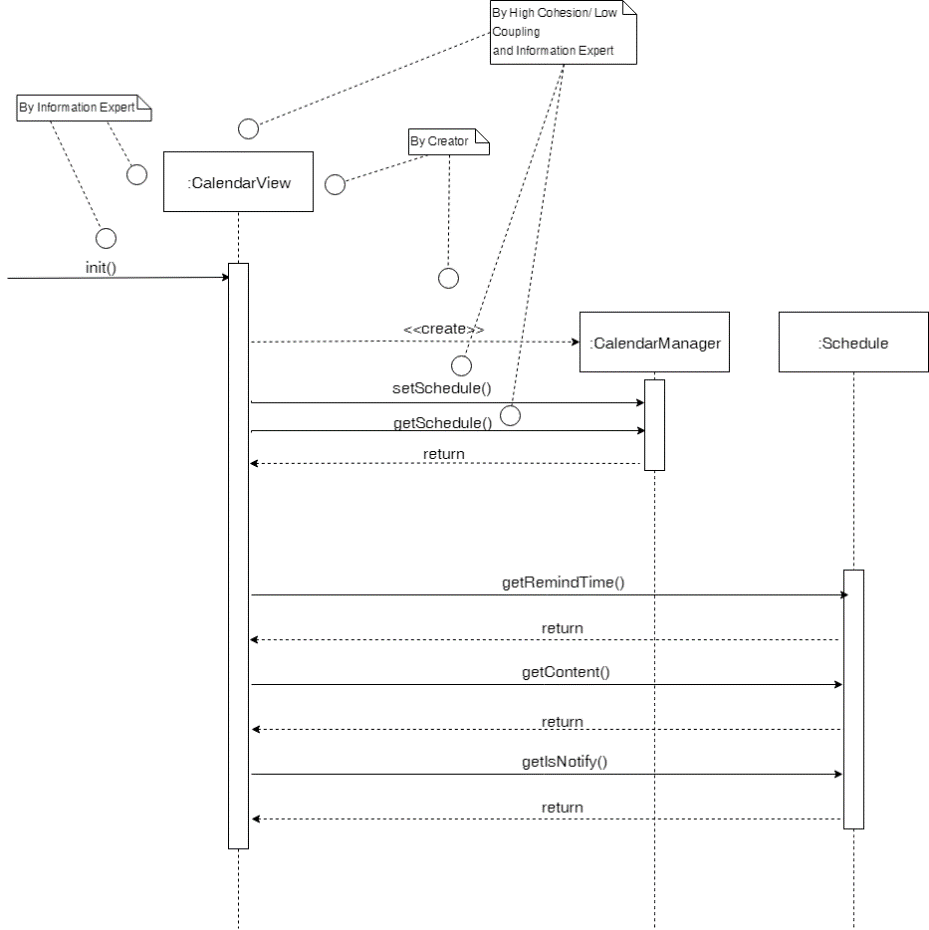
**deleteDayCostRecord(id)**

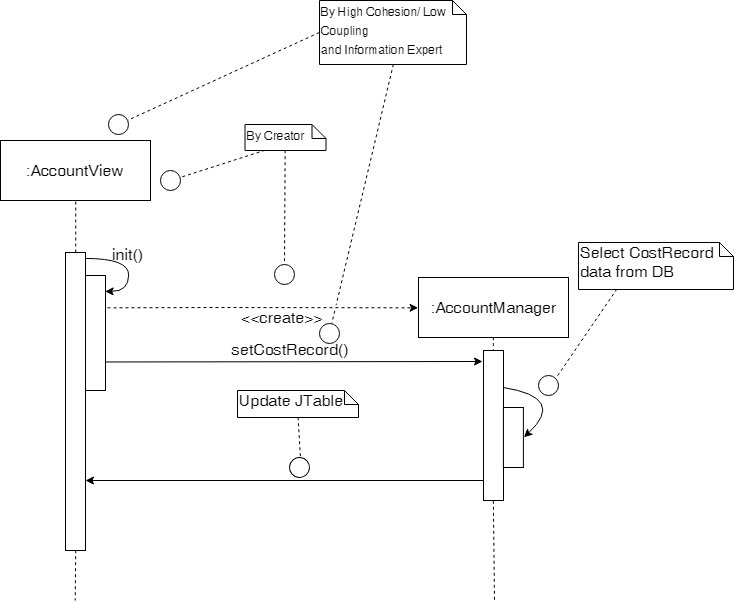
**DetailReportView : init**

**PieChart : generate**

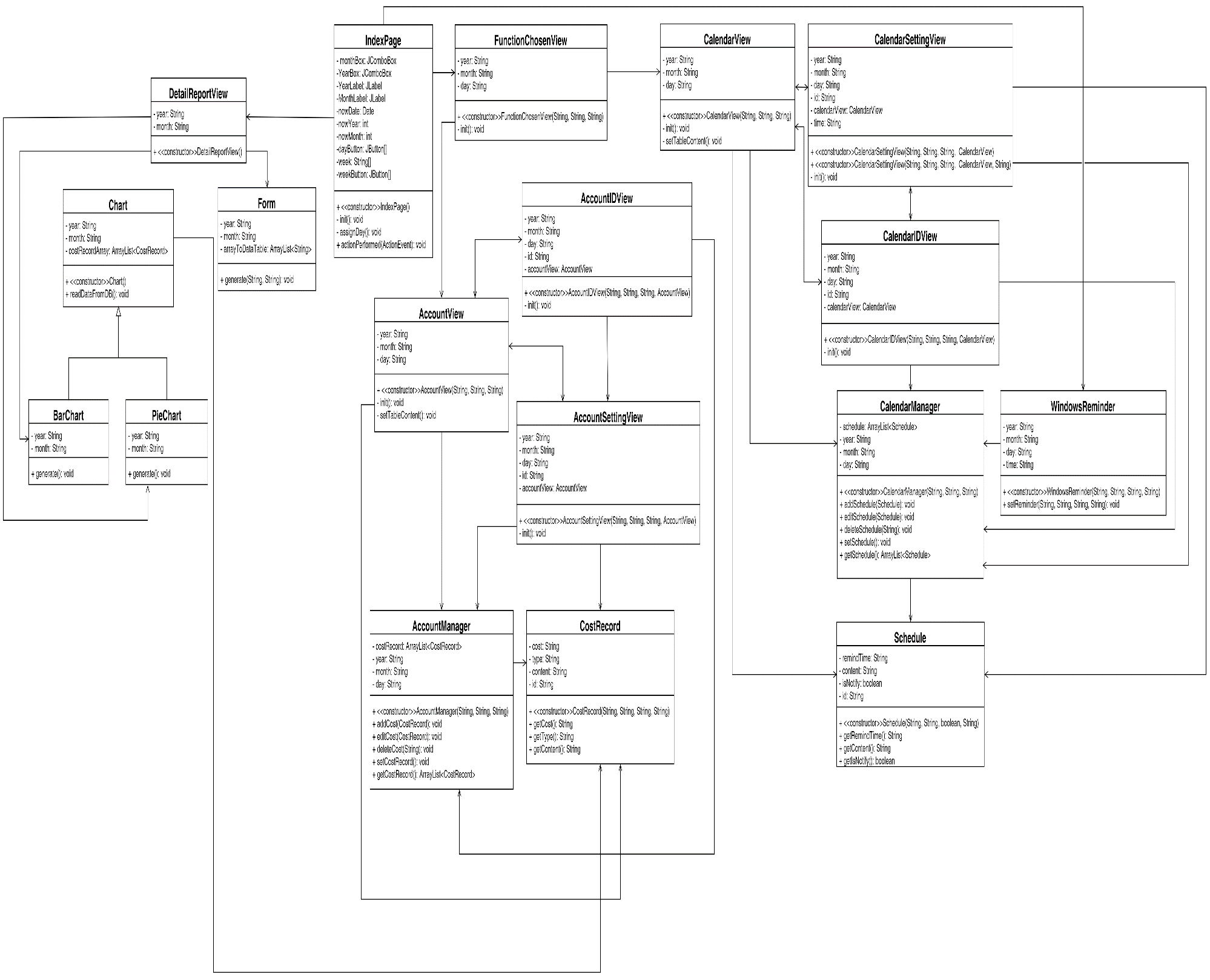
**BarChart : generate**

**exportAsExcelFile()**

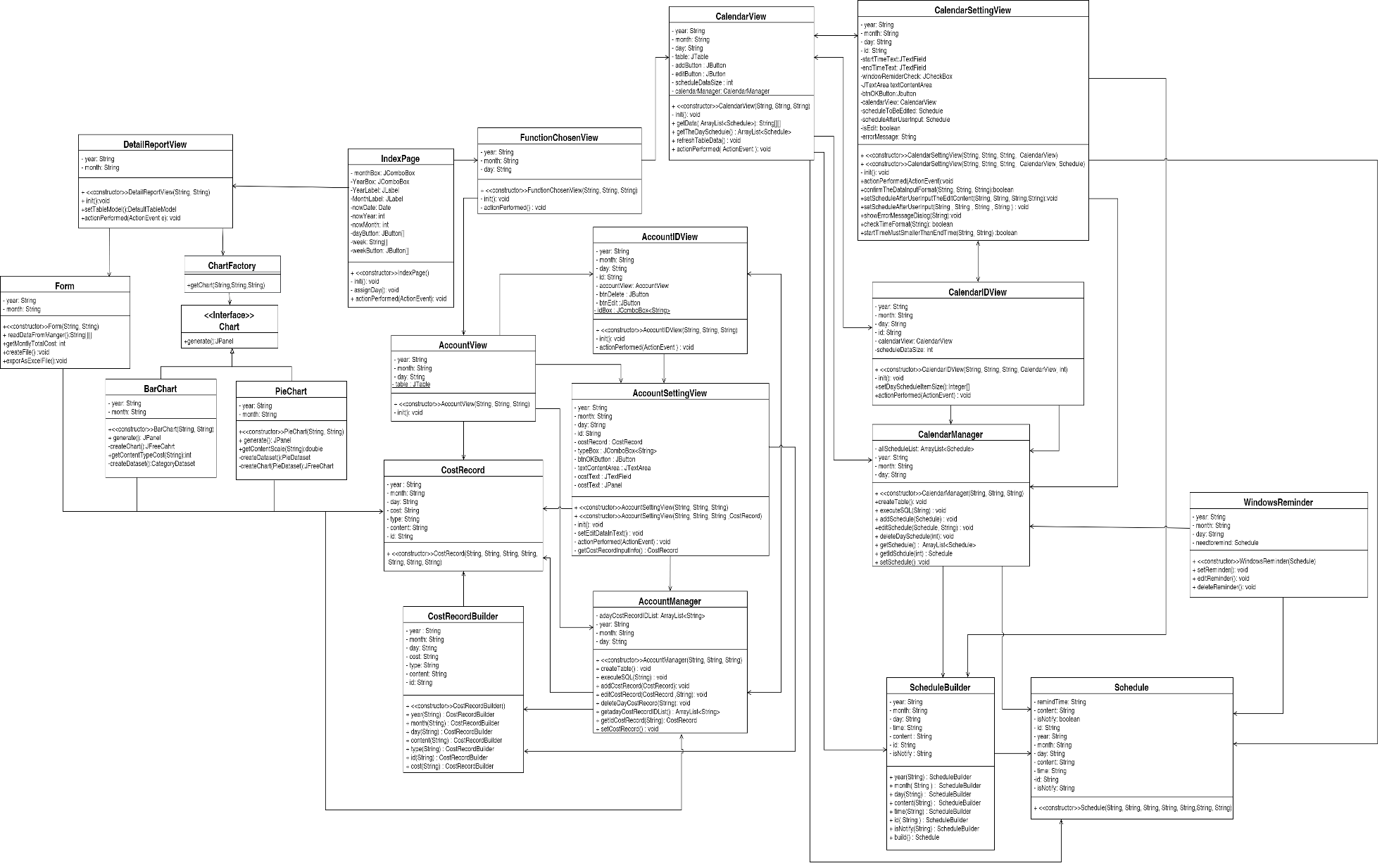
**CalendarView : init**

**AccountView : init**

* 1. **Design Class Model**

****

1. **Implementation Class Model**
   1. **Implementation Class Diagram**

****

* 1. **Implementation Class Model and Design Class Model Difference**
     1. **Comparison with Design and Implementation Class**

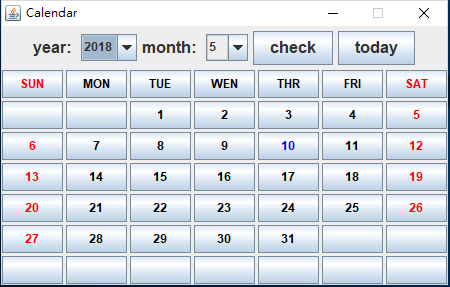
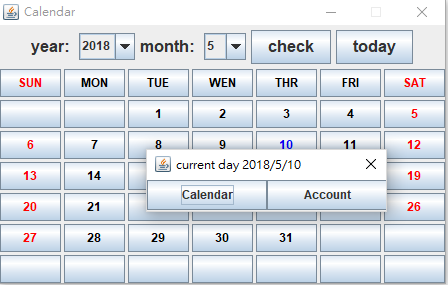
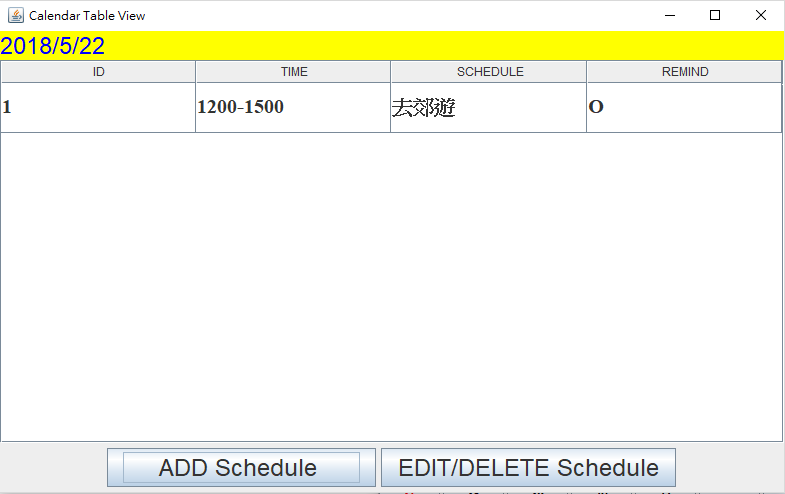
|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **Method** | **Design** | **Imp.** |
| **FunctionChosenView** | init | yes | yes |
| actionPerformed | no | yes |
| **AccountView** | init | yes | yes |
| **AccountIDView** | init | yes | yes |
| actionPerformed | no | yes |
| **AccountSettingView** | init | yes | yes |
| setEditDataInText | no | yes |
| getCostRecordInputInfo | no | yes |
| actionPerformed | no | yes |
| **AccountManager** | addCostRecord | yes | yes |
| editCostRecord | yes | yes |
| deleteDayCostRecord | yes | yes |
| setCostRecord | yes | yes |
| getCostRecord | yes | no |
| createTable | no | yes |
| executeSQL | no | yes |
| getadayCostRecordIDList | no | yes |
| getIdCostRecord | no | yes |
| getMonthlyCost | no | yes |
| getMonthlyTotalCost | no | yes |
| **CalendarManager** | addSchedule | yes | yes |
| editSchedule | yes | yes |
| deleteDaySchedule | yes | yes |
| getSchedule | yes | yes |
| setSchedule | yes | yes |
| createTable | no | yes |
| executeSQL | no | yes |
| getIDSchedule | no | yes |
| **CostRecordBuilder** | year | no | yes |
| month | no | yes |
| day | no | yes |
| content | no | yes |
| type | no | yes |
| id | no | yes |
| build | no | yes |
| **CalendarView** | init | yes | yes |
| setTableContent | yes | no |
| getdata | no | yes |
| getTheDaySchedule | no | yes |
| refreshTableData | no | yes |
| actionPerformed | no | yes |
| **CalendarIDView** | init | yes | yes |
| setDayScheduleItemSize | no | yes |
| actionPerformed | no | yes |
| **CalendarSettingView** | init | yes | yes |
| setEditDataInTextField | no | yes |
| actionPerformed | no | yes |
| confirmTheDataInputFormat | no | yes |
| setScheduleAfterUserInputTheditContent | no | yes |
| startTimeMustSmallerThanEndTime | no | yes |
| **ScheduleBuilder** | year | no | yes |
| month | no | yes |
| day | no | yes |
| content | no | yes |
| time | no | yes |
| id | no | yes |
| isNotify | no | yes |
| build | no | yes |
| **WindowsReminder** | setReminder | yes | yes |
| editReminder | no | yes |
| deleteReminder | no | yes |
| **BarChart** | generate | yes | yes |
| createChart | no | yes |
| getContentTypeCost | no | yes |
| createDataset | no | yes |
| **Chart** | readDataFromDB | yes | no |
| generate | no | yes |
| **ChartFactory** | getChart | no | yes |
| **DetailReportView** | init | no | yes |
| setTableModel | no | yes |
| actionPerformed | no | yes |
| showChart | no | yes |
| **Form** | generate | yes | no |
| readDataFromDB | no | yes |
| getMonthlyTotalCost | no | yes |
| createFile | no | yes |
| exportAsExcelFile | no | yes |

* + 1. **Summary of Implementation of Class/Method Changed**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Number of Added** | **Number of Removed** | **Number of Modified** |
| **class** | 3 | 0 | 12 |
| **method** | 51 | 4 | 51 |

* 1. **Calculate Line of Code**
     1. **Line of Code of Classes**

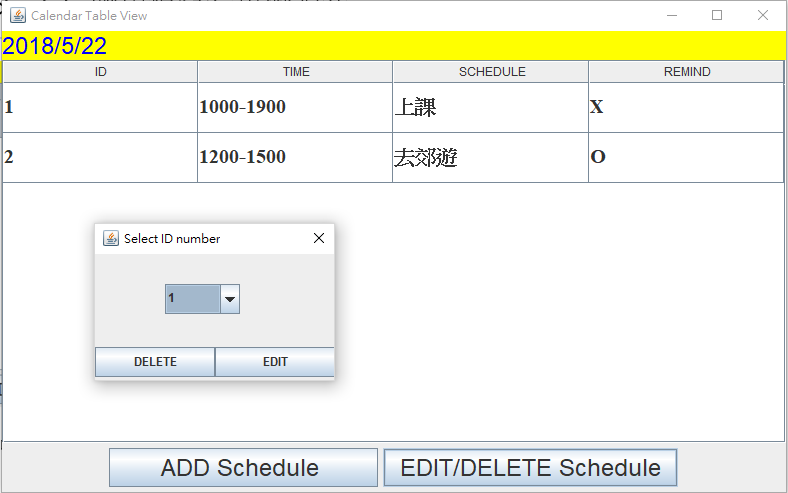
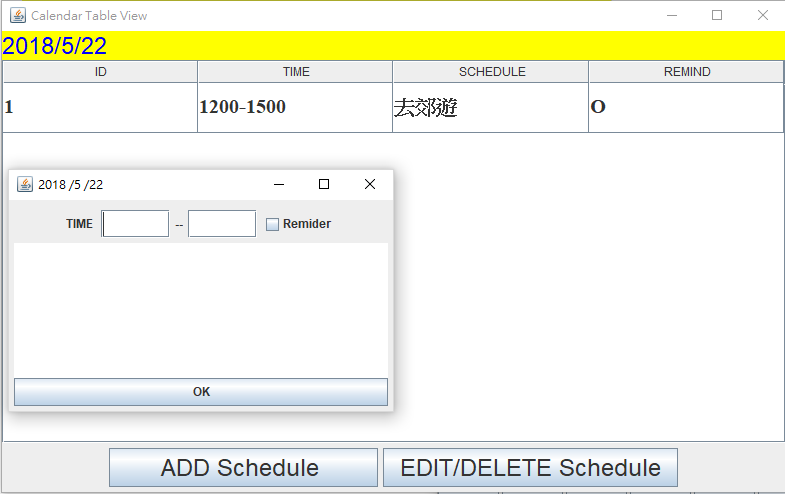
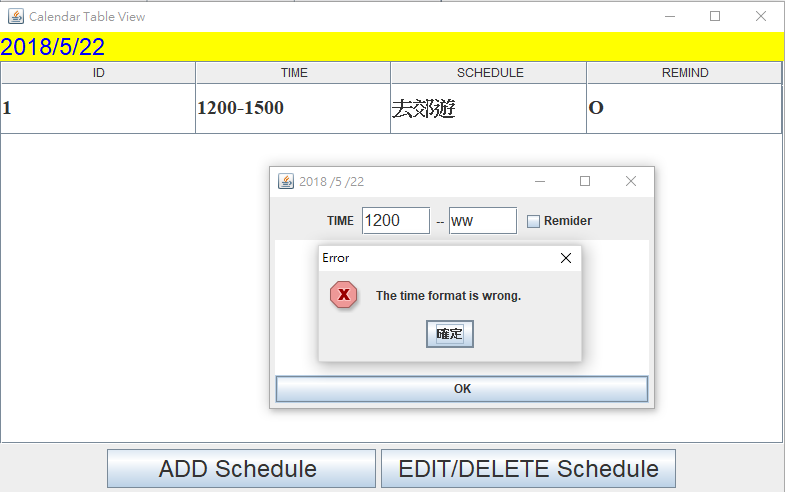
|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Class Name** | **Number of Methods** | **Line of Code in Class** |
| **1** | AccountIDView | 2 | 79 |
| **2** | AccountSettingView | 4 | 128 |
| **3** | AccountView | 1 | 68 |
| **4** | CalendarIDView | 3 | 85 |
| **5** | CalendarSettingView | 9 | 198 |
| **6** | CalendarView | 5 | 134 |
| **7** | WindowsReminder | 3 | 223 |
| **8** | AccountManager | 10 | 256 |
| **9** | CalendarManager | 8 | 260 |
| **10** | CostRecord | 14 | 79 |
| **11** | CostRecordBuilder | 8 | 50 |
| **12** | Schedule | 15 | 83 |
| **13** | ScheduleBuilder | 8 | 51 |
| **14** | BarChart | 2 | 55 |
| **15** | Chart | 1 | 7 |
| **16** | ChartFactory | 1 | 18 |
| **17** | DetailReportView | 4 | 128 |
| **18** | Form | 4 | 106 |
| **19** | PieChart | 4 | 59 |
| **20** | FunctionChosenView | 2 | 77 |
| **21** | IndexPage | 3 | 232 |
| **Sum** | | 111 | 2376 |

1. **Programming**
   1. **Snapshot of System Execution**

點擊行事曆按鈕跳出行事曆視窗

點擊日期跳出選擇行事曆及記帳選擇視窗

月曆畫面

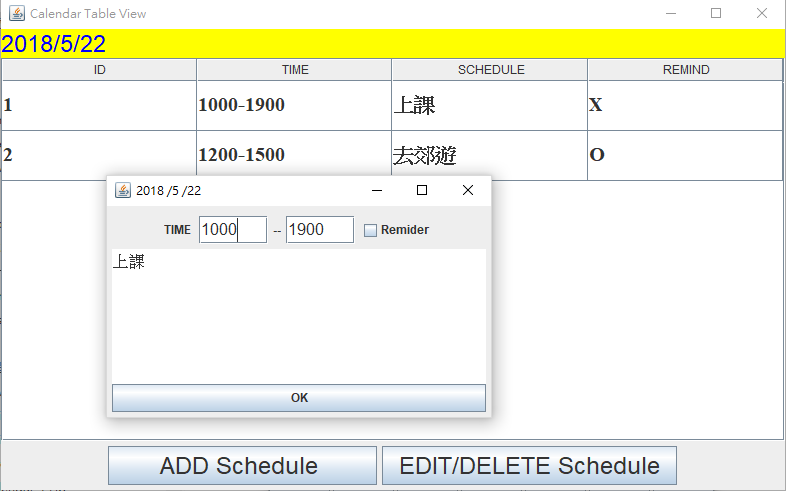
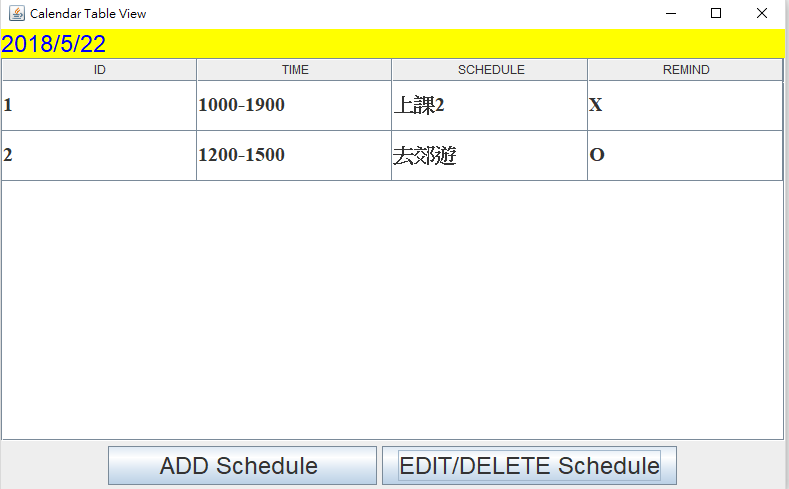
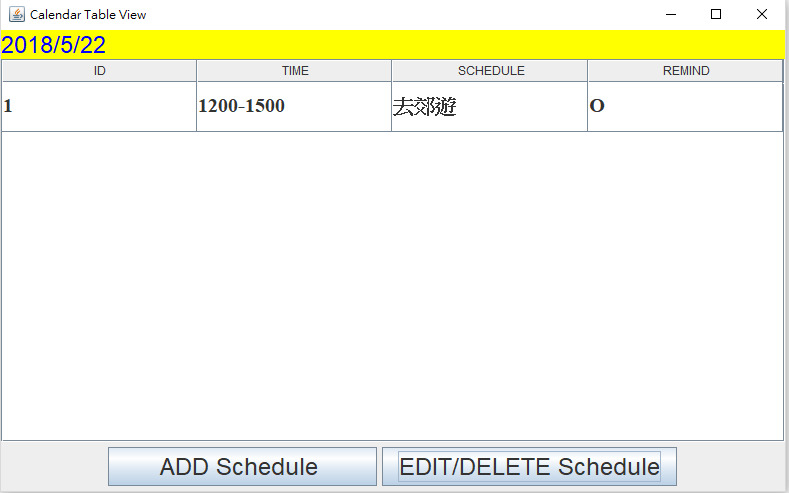


點擊ADD Schedule跳出填寫行事曆視窗

格式不符跳出警告視窗

曆視窗

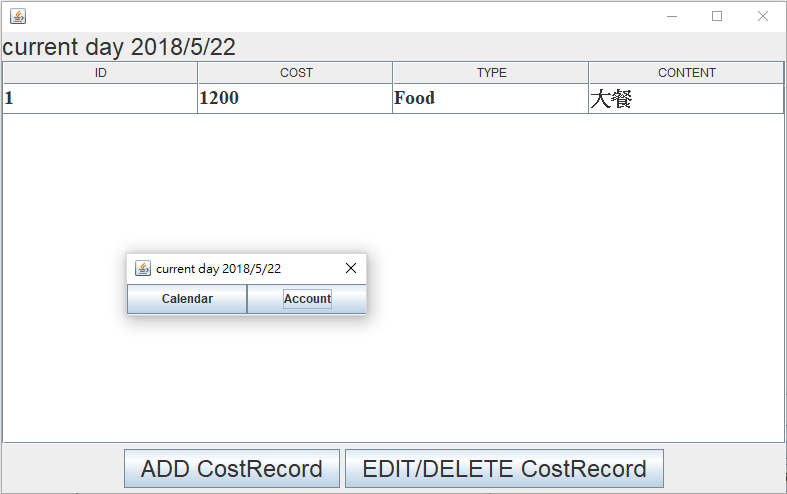
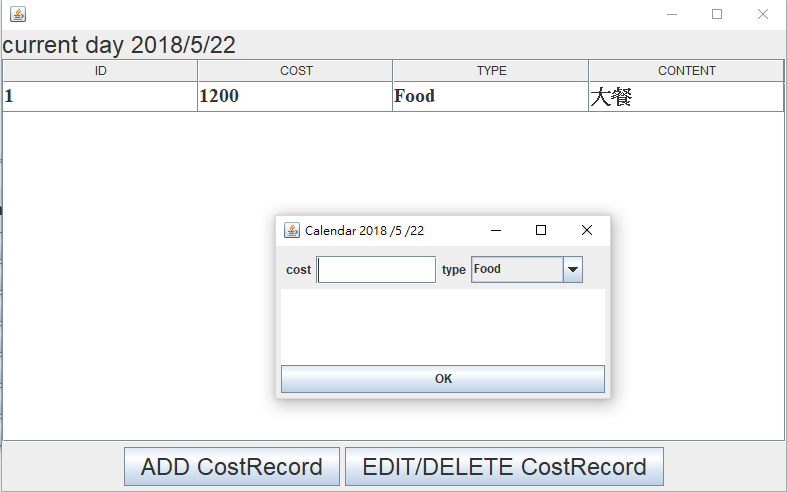
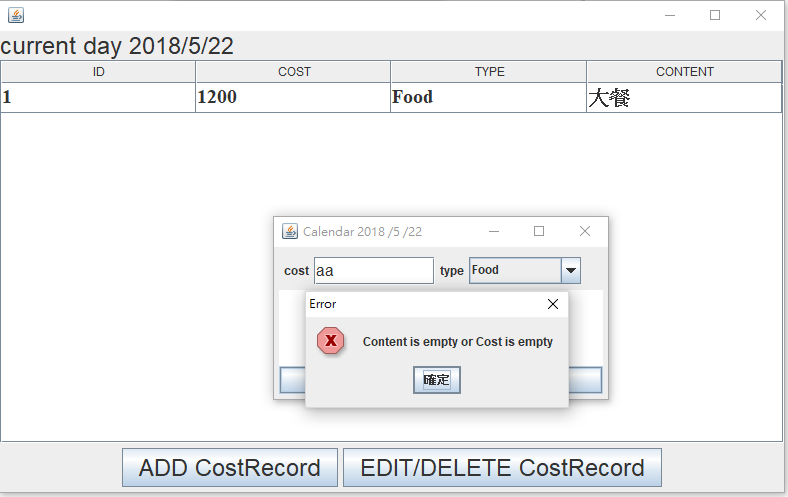
點擊Edit/Delete跳出刪除或編輯行程視窗曆視窗



成功刪除第一筆資料後的畫面

成功編輯第一筆後畫面

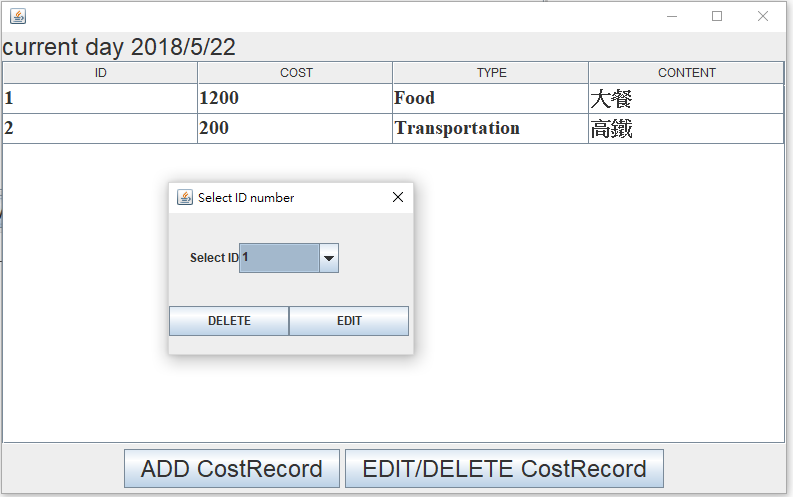
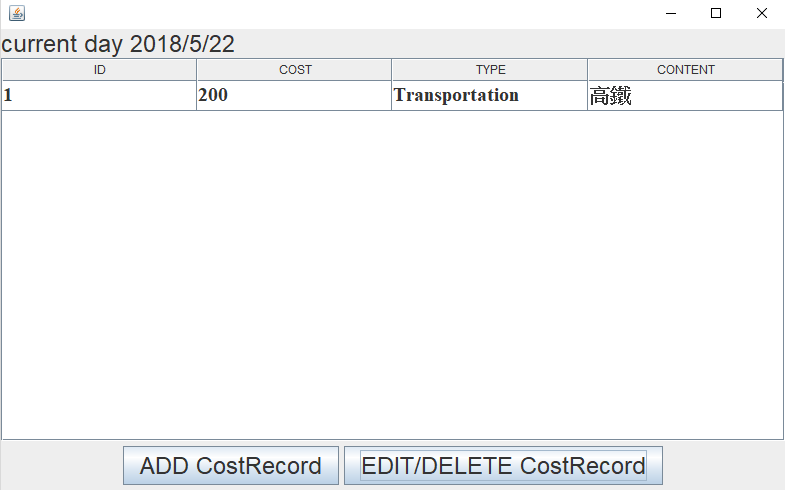
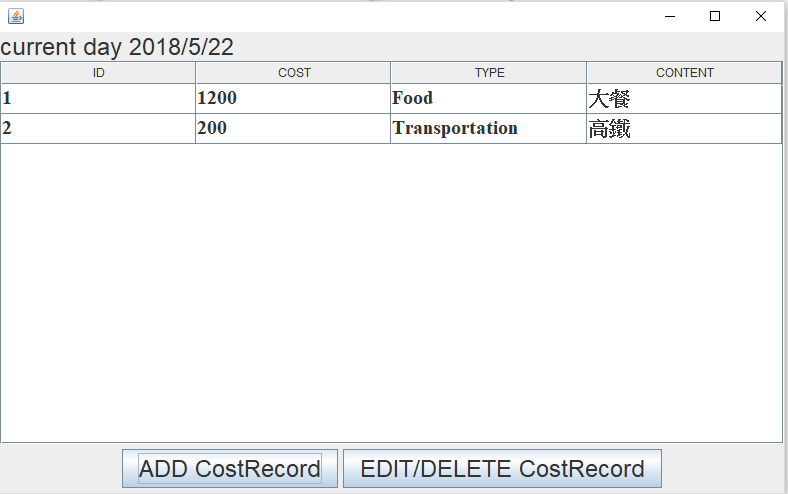
使用者選擇ID後點擊編輯後的視窗



使用者選擇Account後畫面

輸入格式不符合會跳出警告視窗

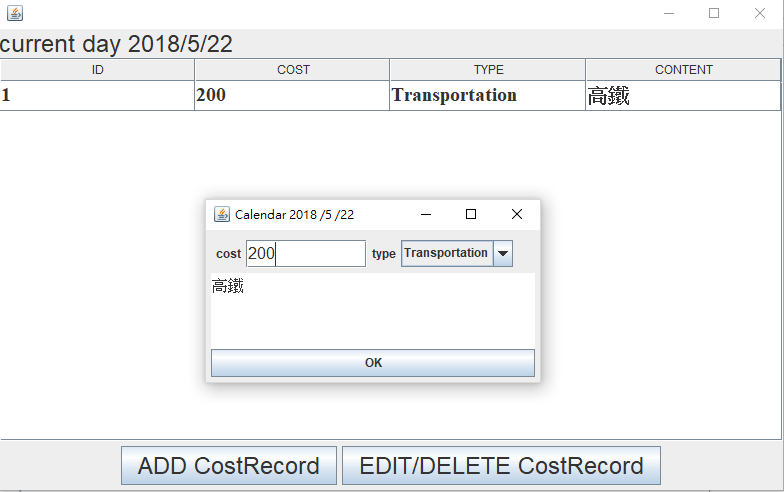
使用者新增一筆新的記帳畫面

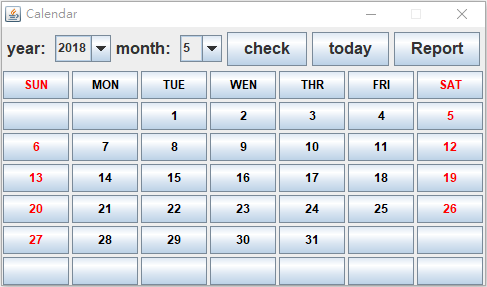
  
  


刪除第一筆資料後畫面

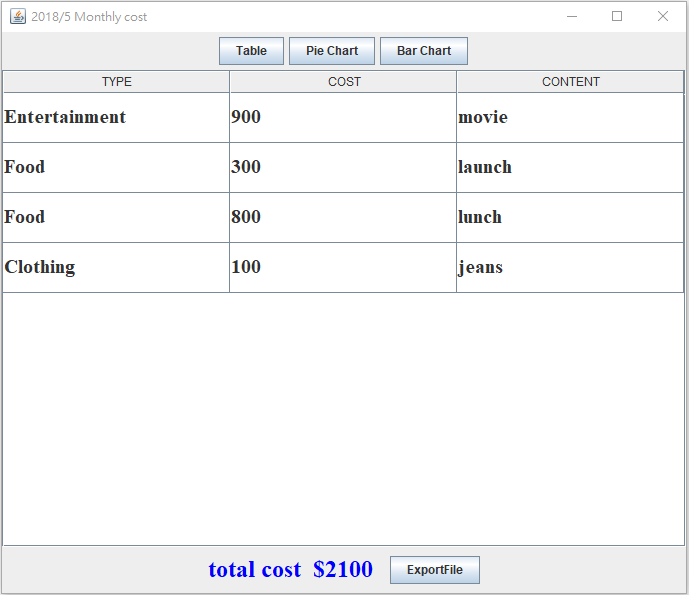
點選EDIT/DELETE後畫面

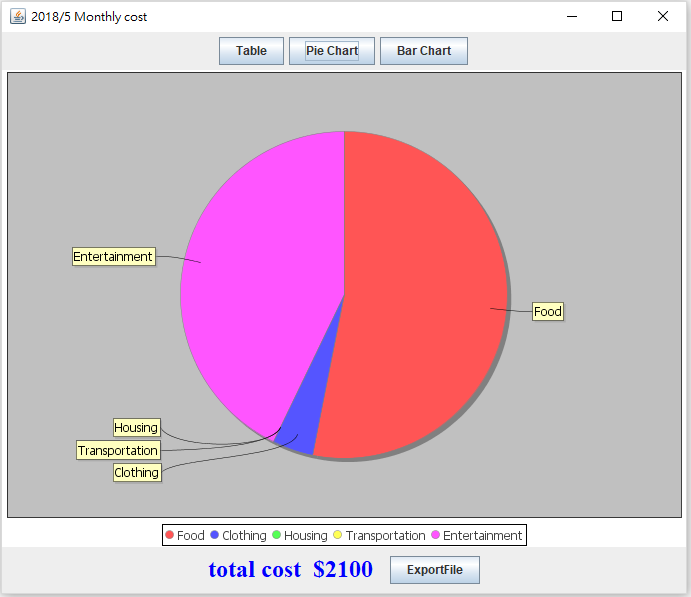
成功新增一筆記帳資料後畫面

  
  
使用者點選主頁面報表按鈕畫面



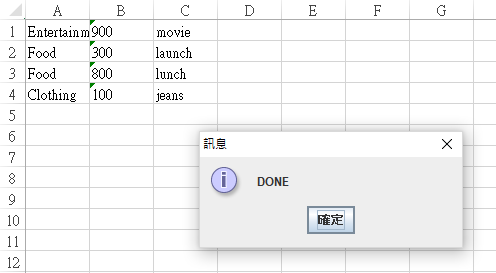
使用者編輯記帳資料的畫面



該月花費明細畫面

以圓餅圖表示該月花費分布



以長條圖表示該月花費分布

匯出之該月花費明細檔案為xlsx格式

* 1. **Source Code Listing**

|  |
| --- |
| **CalendarIDView.java** |
| **package** CalendarView;  **import** java.awt.event.ActionEvent; **import** java.awt.event.ActionListener; **import** java.util.ArrayList;  **import** javax.swing.JButton; **import** javax.swing.JComboBox; **import** javax.swing.JDialog; **import** javax.swing.JFrame; **import** javax.swing.JTextField;  **import** DB.CalendarManager; **import** Model.Schedule;  **public class** CalendarIDView **extends** JDialog **implements** ActionListener {   **private** String **year**;  **private** String **month**;  **private** String **day**;   **private** JButton **btnDelete**;  **private** JButton **btnEdit**;  **private** JComboBox<Integer> **cbEditId**;   **private** CalendarView **calendarView**;  **private int scheduleDataSize**;   **public** CalendarIDView(String year, String month, String day, CalendarView calendarView, **int** scheduleDataSize) {  **this**.**year** = year;  **this**.**month** = month;  **this**.**day** = day;  **this**.**calendarView** = calendarView;  **this**.**scheduleDataSize** = scheduleDataSize;  init();  }   **public** Integer[] setDayScheduleItemSize() {  Integer[] dayScheduleItemSize = **new** Integer[**scheduleDataSize**];  **for** (**int** i = 0; i < **scheduleDataSize**; i++) {  dayScheduleItemSize[i] = i + 1;  }  **return** dayScheduleItemSize;  }   **public void** init() {  setBounds(100, 100, 255, 165);  setTitle(**"Select ID number"**);  setLocationRelativeTo(**null**);  getContentPane().setLayout(**null**);   **cbEditId** = **new** JComboBox<Integer>(setDayScheduleItemSize());  **cbEditId**.setBounds(70, 30, 75, 30);  getContentPane().add(**cbEditId**);   **btnDelete** = **new** JButton(**"DELETE"**);  **btnDelete**.addActionListener(**this**);  **btnDelete**.setBounds(0, 93, 120, 30);  getContentPane().add(**btnDelete**);   **btnEdit** = **new** JButton(**"EDIT"**);  **btnEdit**.addActionListener(**this**);  **btnEdit**.setBounds(120, 93, 120, 30);  getContentPane().add(**btnEdit**);  }   **public void** actionPerformed(ActionEvent e) {  CalendarManager calendarManager = **new** CalendarManager(**year**, **month**, **day**);  **int** chosenId = Integer.*parseInt*(**cbEditId**.getSelectedItem().toString()) ;   **if** (e.getSource() == **btnDelete**) {  calendarManager.deleteDaySchedule(chosenId);  **calendarView**.refreshTableData();  dispose();  } **else if** (e.getSource() == **btnEdit**) {  Schedule schedule = calendarManager.getIdSchdule(chosenId);  CalendarSettingView calendarSettingView = **new** CalendarSettingView(**year**, **month**, **day**, schedule, **calendarView**);  calendarSettingView.setVisible(**true**);  calendarSettingView.setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***);  dispose();  }   }  } |

|  |
| --- |
| **CalendarSettingView.java** |
| **package** CalendarView;  **import** java.awt.BorderLayout;  **import** javax.swing.JFrame; **import** javax.swing.JPanel; **import** javax.swing.border.EmptyBorder;  **import** DB.CalendarManager; **import** Model.Schedule; **import** Model.ScheduleBuilder;  **import** javax.swing.JTextField; **import** javax.swing.JTextArea; **import** javax.swing.JButton; **import** javax.swing.JCheckBox;  **import** java.awt.event.ActionListener; **import** java.awt.event.ActionEvent; **import** javax.swing.JLabel; **import** javax.swing.JOptionPane;   **public class** CalendarSettingView **extends** JFrame **implements** ActionListener {   **private** JTextField **startTimeText**;  **private** JTextField **endTimeText**;  **private** JCheckBox **windowRemiderCheck**;  **private** JTextArea **textContentArea**;  **private** JButton **btnOKButton**;   **private** String **year**;  **private** String **month**;  **private** String **day**;  **private** CalendarView **calendarView**;  **private** Schedule **scheduleToBeEdited** = **null**;  **private** Schedule **scheduleAfterUserInput**;  **boolean isEdit** = **false**;  **private** String **errorMessage**;   **public** CalendarSettingView(String year, String month, String day, CalendarView calendarView) {  **this**.**year** = year;  **this**.**month** = month;  **this**.**day** = day;  **this**.**calendarView** = calendarView;  init();  }   **public** CalendarSettingView(String year, String month, String day, Schedule schedule, CalendarView calendarView) {  **this**.**year** = year;  **this**.**month** = month;  **this**.**day** = day;  **this**.**scheduleToBeEdited** = schedule;  **this**.**calendarView** = calendarView;  init();  setEditDataInTextField();  **isEdit** = **true**;  }   **public void** setEditDataInTextField() {  String[] parts = **scheduleToBeEdited**.getTime().split(**"-"**);  **startTimeText**.setText(parts[0]);  **endTimeText**.setText(parts[1]);  **textContentArea**.setText(**scheduleToBeEdited**.getContent());  **if** (**scheduleToBeEdited**.getIsNotify().equals(**"true"**))  **windowRemiderCheck**.setSelected(**true**);  }   **public void** init() {  setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  setBounds(100, 100, 400, 250);  JPanel contentPane = **new** JPanel();  **this**.setTitle(**year** + **" /"** + **month** + **" /"** + **day**);  contentPane.setBorder(**new** EmptyBorder(5, 5, 5, 5));  contentPane.setLayout(**new** BorderLayout(0, 0));  setContentPane(contentPane);   **textContentArea** = **new** JTextArea();  **textContentArea**.setFont(**textContentArea**.getFont().deriveFont(16f));  contentPane.add(**textContentArea**, BorderLayout.***CENTER***);   **btnOKButton** = **new** JButton(**"OK"**);  **btnOKButton**.addActionListener(**this**);   contentPane.add(**btnOKButton**, BorderLayout.***SOUTH***);  JPanel panel = **new** JPanel();  contentPane.add(panel, BorderLayout.***NORTH***);  JLabel lblNewLabel = **new** JLabel(**"TIME "**);  panel.add(lblNewLabel);   **startTimeText** = **new** JTextField();  **startTimeText**.setFont(**textContentArea**.getFont().deriveFont(16f));  **startTimeText**.setColumns(5);   panel.add(**startTimeText**);  panel.add(**new** JLabel(**"--"**));   **endTimeText** = **new** JTextField();  **endTimeText**.setFont(**endTimeText**.getFont().deriveFont(16f));  panel.add(**endTimeText**);   **endTimeText**.setColumns(5);  **windowRemiderCheck** = **new** JCheckBox(**"Remider"**);  panel.add(**windowRemiderCheck**);  }   **public void** actionPerformed(ActionEvent e) {   String userInputContent = **textContentArea**.getText();  String userInputStartTime = **startTimeText**.getText();  String userInputEndTime = **endTimeText**.getText();  String userRemiderCheck = (**windowRemiderCheck**.isSelected()) ? **"true"** : **"false"**;   **if** (!confirmTheDataInputFormat(userInputContent, userInputStartTime, userInputEndTime))  showErrorMessageDialog(**errorMessage**);  **else** {  CalendarManager calendarManager = **new** CalendarManager(**year**, **month**, **day**);  **if** (**isEdit**) {  setScheduleAfterUserInputTheEditContent(userInputContent, userInputStartTime, userInputEndTime,  userRemiderCheck);  calendarManager.editSchedule(**scheduleToBeEdited**, **scheduleToBeEdited**.getId());  } **else** {  setScheduleAfterUserInput(userInputContent, userInputStartTime, userInputEndTime, userRemiderCheck);  calendarManager.addSchedule(**scheduleAfterUserInput**);  }  **calendarView**.refreshTableData();  dispose();  }  } *// actionPerformed()* **public boolean** confirmTheDataInputFormat(String content, String startTime, String endTime ) {   **if** (!checkTimeFormat(startTime) || !checkTimeFormat(endTime)) {  **errorMessage** = **"The time format is wrong."**;  **return false**;  } **else if** (!startTimeMustSmallerThanEndTime(startTime, endTime)) {  **errorMessage** = **"The start time must smaller than end time."**;  **return false**;  } **else if** (content.length() == 0) {  **errorMessage** = **"Content is empty, please enter the schedule content."**;  **return false**;  } **else if** (content.length() >= 20) {  **errorMessage** = **"Content is too long, please limit to 20 words."**;  **return false**;  }  **return true**;  }   **public void** setScheduleAfterUserInputTheEditContent(String content, String startTime, String endTime,  String RemiderCheck) {  String time = startTime + **"-"** + endTime;  **scheduleToBeEdited**.setContent(content);  **scheduleToBeEdited**.setTime(time);  **scheduleToBeEdited**.setIsNotify(RemiderCheck);  }   **public void** setScheduleAfterUserInput(String content, String startTime, String endTime, String RemiderCheck) {  String time = startTime + **"-"** + endTime;  **scheduleAfterUserInput** = **new** ScheduleBuilder().year(**year**).month(**month**).day(**day**).isNotify(RemiderCheck)  .time(time).content(content).build();  }   **public void** showErrorMessageDialog(String message) {  JOptionPane.*showMessageDialog*(**null**, message, **"Error"**, JOptionPane.***ERROR\_MESSAGE***);  }   **public boolean** checkTimeFormat(String time) { *// not a number* **if** (!time.matches(**"\\d+"**) || time.length() != 4)  **return false**;  **else** {  **int** hour = Integer.*parseInt*(String.*valueOf*(time.charAt(0)) + String.*valueOf*(time.charAt(1)));  **int** min = Integer.*parseInt*(String.*valueOf*(time.charAt(2)) + String.*valueOf*(time.charAt(3)));   **if** (hour > 24)  **return false**;  **else if** (min > 59)  **return false**;  **else if** (hour == 24 && min != 00)  **return false**;  }  **return true**;  }   **public boolean** startTimeMustSmallerThanEndTime(String startTime, String endTime) {  **int** startHour = Integer.*parseInt*(String.*valueOf*(startTime.charAt(0)) + String.*valueOf*(startTime.charAt(1)));  **int** endHour = Integer.*parseInt*(String.*valueOf*(endTime.charAt(0)) + String.*valueOf*(endTime.charAt(1)));   **int** startMin = Integer.*parseInt*(String.*valueOf*(startTime.charAt(2)) + String.*valueOf*(startTime.charAt(3)));  **int** endMin = Integer.*parseInt*(String.*valueOf*(endTime.charAt(2)) + String.*valueOf*(endTime.charAt(3)));   **if** (startHour > endHour)  **return false**;  **else if** (startHour == endHour && startMin > endMin)  **return false**;   **return true**;  } } |

|  |
| --- |
| **CalendarView.java** |
| **package** CalendarView;  **import** javax.swing.\*; **import** javax.swing.table.DefaultTableModel; **import** javax.swing.table.JTableHeader;  **import** DB.CalendarManager; **import** Model.Schedule;  **import** java.awt.\*; **import** java.awt.event.ActionEvent; **import** java.awt.event.ActionListener; **import** java.awt.event.MouseEvent; **import** java.util.ArrayList;  **public class** CalendarView **extends** JFrame **implements** ActionListener {   **private** String **year**;  **private** String **month**;  **private** String **day**;  **private** JTable **table**;  **private** JButton **addButton**;  **private** JButton **editButton**;   **private int scheduleDataSize**;  CalendarManager **calendarManager**;  CalendarSettingView **calendarSettingView** = **null**;   **public** CalendarView(String year, String month, String day) {  **this**.**year** = year;  **this**.**month** = month;  **this**.**day** = day;  **scheduleDataSize** = 0;  **calendarManager** = **new** CalendarManager(year, month, day);  init();  }   **public** String[][] getdata(ArrayList<Schedule> data) {  **scheduleDataSize** = data.size();  String[][] mydata = **new** String[data.size()][4];   **for** (**int** i = 0; i < data.size(); i++) {  mydata[i][0] = Integer.*toString*(i + 1);  mydata[i][1] = data.get(i).getTime();  mydata[i][2] = data.get(i).getContent();  mydata[i][3] = data.get(i).getIsNotify().equals(**"true"**) ? **"O"** : **"X"**;  }  **return** mydata;  }   **public** ArrayList<Schedule> getTheDaySchedule() {  **calendarManager**.setSchedule();  ArrayList<Schedule> data = **calendarManager**.getSchedule();  **return** data;  }   **public void** refreshTableData() {  DefaultTableModel model = (DefaultTableModel) **table**.getModel();  model.setRowCount(0);  ArrayList<Schedule> data = getTheDaySchedule();  **scheduleDataSize** = data.size();  String[] mydata = **new** String[4];  **for** (**int** i = 0; i < data.size(); i++) {  mydata[0] = Integer.*toString*(i + 1);  mydata[1] = data.get(i).getTime();  mydata[2] = data.get(i).getContent();  mydata[3] = data.get(i).getIsNotify().equals(**"true"**) ? **"O"** : **"X"**;  model.addRow(mydata);  }  model.fireTableDataChanged();  }   **public void** init() {   setTitle(**"Calendar Table View"**);  **this**.getContentPane().setBackground(Color.***YELLOW***);  setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  String[] columns = { **"ID"**, **"TIME"**, **"SCHEDULE"**, **"REMIND"** };   Object[][] data = getdata(getTheDaySchedule());   DefaultTableModel model = **new** DefaultTableModel(data, columns);   **table** = **new** JTable(model);  **table**.setEnabled(**false**); *//* JScrollPane scrollPane = **new** JScrollPane(**table**);  **table**.setFillsViewportHeight(**true**);  **table**.setRowHeight(50);  **table**.setFont(**new** Font(**"Serif"**, Font.***BOLD***, 20));   JLabel lblHeading = **new** JLabel(**year** + **"/"** + **month** + **"/"** + **day**);  lblHeading.setForeground(Color.***BLUE***);  lblHeading.setFont(**new** Font(**"Arial"**, Font.***TRUETYPE\_FONT***, 24));   getContentPane().setLayout(**new** BorderLayout());   getContentPane().add(lblHeading, BorderLayout.***PAGE\_START***);  getContentPane().add(scrollPane, BorderLayout.***CENTER***);   setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  setSize(800, 500);  setVisible(**true**);   JPanel panel = **new** JPanel();  add(panel, BorderLayout.***SOUTH***);   **addButton** = **new** JButton(**" ADD Schedule "**);  **editButton** = **new** JButton(**"EDIT/DELETE Schedule"**);  **addButton**.setFont(**new** Font(**"Arial"**, Font.***PLAIN***, 24));  **editButton**.setFont(**new** Font(**"Arial"**, Font.***PLAIN***, 24));  panel.add(**addButton**);  panel.add(**editButton**);   **addButton**.addActionListener(**this**);  **editButton**.addActionListener(**this**);  }     **public void** actionPerformed(ActionEvent e) {  **if** (e.getSource() == **addButton**) {  **if** (**calendarSettingView** == **null** || !**calendarSettingView**.isDisplayable()) {  **calendarSettingView** = **new** CalendarSettingView(**year**, **month**, **day**, **this**);  **calendarSettingView**.setVisible(**true**);  **calendarSettingView**.setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***);  }   } **else if** (e.getSource() == **editButton**) {  CalendarIDView dialog = **new** CalendarIDView(**year**, **month**, **day**, **this**, **scheduleDataSize**);  dialog.setModal(**true**);  dialog.setVisible(**true**);  }  } } |

|  |
| --- |
| **AccountIDView.java** |
| **package** AccountView;  **import** java.awt.event.ActionEvent; **import** java.awt.event.ActionListener; **import** java.util.ArrayList;  **import** javax.swing.DefaultComboBoxModel; **import** javax.swing.JButton; **import** javax.swing.JComboBox; **import** javax.swing.JDialog; **import** javax.swing.JFrame; **import** javax.swing.JLabel;  **import** DB.AccountManager; **import** Model.CostRecord;  **public class** AccountIDView **extends** JDialog **implements** ActionListener{   **private** String **year**;  **private** String **month**;  **private** String **day**;  **private** String **id**;  **public static** JComboBox<String> *idBox* = **new** JComboBox<String>();  **private** JButton **btnDelete**;  **private** JButton **btnEdit**;  **private** ArrayList<String> **adayCostRecordIDList**;  **public** AccountIDView(String year, String month, String day) {  **this**.**year** = year;  **this**.**month** = month;  **this**.**day** = day;  init();  }   **public void** init(){  AccountManager accountManager = **new** AccountManager(**year**, **month**, **day**);  **adayCostRecordIDList** = accountManager.getadayCostRecordIDList();  setBounds(100, 100, 260, 180);  setTitle(**"Select ID number"**);  setLocationRelativeTo(**null**);  getContentPane().setLayout(**null**);  *idBox*.setBounds(70, 30, 100, 30);  getContentPane().add(*idBox*);  JLabel label = **new** JLabel(**"Select ID"**);  label.setBounds(20, 30, 50, 30);  getContentPane().add(label);  **btnDelete** = **new** JButton(**"DELETE"**);  **btnDelete**.addActionListener(**this**);  **btnDelete**.setBounds(0, 93, 120, 30);  getContentPane().add(**btnDelete**);   **btnEdit** = **new** JButton(**"EDIT"**);  **btnEdit**.addActionListener(**this**);  **btnEdit**.setBounds(120, 93, 120, 30);  getContentPane().add(**btnEdit**);  }   **public static void** main(String[] args) {  AccountIDView test = **new** AccountIDView(**"2018"**,**"5"**,**"5"**);  }   **public void** actionPerformed(ActionEvent e) {  AccountManager accountManager = **new** AccountManager(**year**, **month**, **day**);  **adayCostRecordIDList** = accountManager.getadayCostRecordIDList();  System.***out***.println(*idBox*.getSelectedIndex());  String id = **adayCostRecordIDList**.get(*idBox*.getSelectedIndex());  **if** (e.getSource() == **btnDelete**) {  accountManager.deleteDayCostRecord(id);  dispose();  }  **else if**( e.getSource() == **btnEdit** ) {  CostRecord costRecord = accountManager.getIdCostRecord(id);  AccountSettingView accountSettingView = **new** AccountSettingView(**year**, **month**, **day**, costRecord);  accountSettingView.setVisible(**true**);  accountSettingView.setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***);  dispose();  }  }  } |

|  |
| --- |
| **AccountSettingView.java** |
| **package** AccountView;  **import** java.awt.BorderLayout; **import** java.awt.FlowLayout;  **import** javax.swing.JFrame; **import** javax.swing.JPanel; **import** javax.swing.border.EmptyBorder; **import** DB.AccountManager; **import** Model.CostRecord; **import** Model.CostRecordBuilder; **import** javax.swing.JTextField; **import** javax.swing.JTextArea; **import** javax.swing.JButton; **import** javax.swing.JComboBox; **import** java.awt.event.ActionListener; **import** java.awt.event.ActionEvent; **import** javax.swing.JLabel; **import** javax.swing.JOptionPane;  **public class** AccountSettingView **extends** JFrame **implements** ActionListener {   **private** JPanel **contentPane**;  **private** JTextField **costText**;  **private** JTextArea **textContentArea**;  **private** JButton **btnOKButton**;  **private** JComboBox<String> **typeBox** = **new** JComboBox<String>();  **private** String **year**;  **private** String **month**;  **private** String **day**;  **private** CostRecord **costRecord** = **null**;  **boolean isEdit** = **false**;   **public** AccountSettingView(String year, String month, String day) {  **this**.**year** = year;  **this**.**month** = month;  **this**.**day** = day;  init();  }   **public** AccountSettingView(String year, String month, String day ,CostRecord costRecord) {  **this**.**year** = year;  **this**.**month** = month;  **this**.**day** = day;  **this**.**costRecord** = costRecord;  **this**.**isEdit** = **true**;  init();  setEditDataInText();  }  **public void** setEditDataInText() {  **costText**.setText(**costRecord**.getcost());  **typeBox**.setSelectedItem(**costRecord**.gettype());  **textContentArea**.setText(**costRecord**.getContent());  }   **public void** init() {  **typeBox**.addItem(**"Food"**);  **typeBox**.addItem(**"Clothing"**);  **typeBox**.addItem(**"Housing"**);  **typeBox**.addItem(**"Transportation"**);  **typeBox**.addItem(**"Entertainment"**);  setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  setBounds(100, 100, 350, 191);  **contentPane** = **new** JPanel();  **this**.setTitle(**"Calendar "** + **year** + **" /"** + **month** + **" /"** + **day**);  **contentPane**.setBorder(**new** EmptyBorder(5, 5, 5, 5));  **contentPane**.setLayout(**new** BorderLayout(0, 0));  setContentPane(**contentPane**);   **textContentArea** = **new** JTextArea();  **textContentArea**.setFont(**textContentArea**.getFont().deriveFont(16f));  **contentPane**.add(**textContentArea**, BorderLayout.***CENTER***);   **btnOKButton** = **new** JButton(**"OK"**);  **btnOKButton**.addActionListener(**this**);   **contentPane**.add(**btnOKButton**, BorderLayout.***SOUTH***);  JPanel panel = **new** JPanel();  **contentPane**.add(panel, BorderLayout.***NORTH***);  JLabel lblNewLabel = **new** JLabel(**"cost"**);  panel.setLayout(**new** FlowLayout(FlowLayout.***LEFT***));  panel.add(lblNewLabel);  **costText** = **new** JTextField();  **costText**.setFont(**textContentArea**.getFont().deriveFont(16f));  **costText**.setColumns(9);   panel.add(**costText**);  panel.add(**new** JLabel(**"type"**));  panel.add(**typeBox**);  }   **public void** actionPerformed(ActionEvent e) {   AccountManager accountManager = **new** AccountManager(**year**, **month**, **day**);  **if**(!getCostRecordInputInfo().getContent().isEmpty() && !getCostRecordInputInfo().getcost().isEmpty() && getCostRecordInputInfo().getcost().length() <= 12 && getCostRecordInputInfo().getContent().length() <= 20 && getCostRecordInputInfo().getcost().matches(**"\\d+"**)){  **if** (**isEdit**) {  accountManager.editCostRecord(getCostRecordInputInfo(),**costRecord**.getId());  dispose();  } **else** {  accountManager.addCostRecord(getCostRecordInputInfo());  dispose();  }  }  **else** {  **if**(getCostRecordInputInfo().getContent().isEmpty() || getCostRecordInputInfo().getcost().isEmpty())  JOptionPane.*showMessageDialog*(**null**, **"Content is empty or Cost is empty"**, **"Error"**, JOptionPane.***ERROR\_MESSAGE***);  **else if**(! getCostRecordInputInfo().getcost().matches(**"\\d+"**))  JOptionPane.*showMessageDialog*(**null**, **"Cost format error"**, **"Error"**, JOptionPane.***ERROR\_MESSAGE***);  **else** JOptionPane.*showMessageDialog*(**null**, **"Cost length > 12 or Content length > 20"**, **"Error"**, JOptionPane.***ERROR\_MESSAGE***);  }  }   **public** CostRecord getCostRecordInputInfo() {  String content = **textContentArea**.getText();  String cost = **costText**.getText();  String type = **typeBox**.getSelectedItem().toString();  CostRecord costRecord = **new** CostRecordBuilder().year(**year**).month(**month**).day(**day**).cost(cost).type(type).content(content).build();  **return** costRecord;  }   **public static void** main(String[] args) {  AccountSettingView accountSettingView = **new** AccountSettingView(**"2018"**, **"4"**, **"26"**);  accountSettingView.setVisible(**true**);   }  } |

|  |
| --- |
| **AccountView.java** |
| **package** AccountView;  **import** javax.swing.\*; **import** DB.AccountManager; **import** java.awt.\*; **import** java.awt.event.ActionEvent; **import** java.awt.event.ActionListener; **public class** AccountView **extends** JFrame{  **private** String **year**;  **private** String **month**;  **private** String **day**;  **public static** JTable *table*;  **public** AccountView(**final** String year, **final** String month, **final** String day) {  **this**.**year** = year;  **this**.**month** = month;  **this**.**day** = day;  *table* = **new** JTable();  JScrollPane scrollPane = **new** JScrollPane(*table*);  *table*.setFillsViewportHeight(**true**);  *table*.setRowHeight(30);  *table*.setFont(**new** Font(**"Serif"**, Font.***BOLD***, 20));  *table*.setEnabled(**false**);  JLabel lblHeading = **new** JLabel(**"current day "** + year + **"/"**+ month + **"/"** + day);  lblHeading.setFont(**new** Font(**"Arial"**,Font.***TRUETYPE\_FONT***,24));  getContentPane().setLayout(**new** BorderLayout());  getContentPane().add(lblHeading,BorderLayout.***PAGE\_START***);  getContentPane().add(scrollPane,BorderLayout.***CENTER***);  setSize(800, 500);  setVisible(**true**);   JPanel panel=**new** JPanel();  add(panel, BorderLayout.***SOUTH***);  JButton addButton = **new** JButton(**"ADD CostRecord"**);  JButton okButton = **new** JButton(**"EDIT/DELETE CostRecord"**);  addButton.setFont(**new** Font(**"Arial"**, Font.***PLAIN***, 24));  okButton.setFont(**new** Font(**"Arial"**, Font.***PLAIN***, 24));  panel.add(addButton);  panel.add(okButton);  init();  addButton.addActionListener(**new** ActionListener() {  **public void** actionPerformed(ActionEvent e) {  AccountSettingView accountSettingView = **new** AccountSettingView(year,month,day);  accountSettingView.setVisible(**true**);  accountSettingView.setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***);  }  });  okButton.addActionListener(**new** ActionListener() {  **public void** actionPerformed(ActionEvent e) {  AccountIDView accountIDView = **new** AccountIDView(year,month,day);  accountIDView.setVisible(**true**);  accountIDView.setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***);  }  });  }   **public void** init() {  AccountManager test = **new** AccountManager(**year**,**month**,**day**);  test.setCostRecord();  }  **public static void** main(String[] args) {  AccountView AccountViewtest = **new** AccountView(**"2018"**,**"4"**,**"27"**);  } } |

|  |
| --- |
| **AccountManager.java** |
| **package** DB;  **import** java.sql.Connection;  **import** java.sql.DriverManager; **import** java.sql.ResultSet; **import** java.sql.ResultSetMetaData; **import** java.sql.SQLException; **import** java.sql.Statement; **import** java.util.ArrayList;  **import** javax.swing.DefaultComboBoxModel; **import** javax.swing.table.DefaultTableModel;  **import** AccountView.AccountIDView; **import** AccountView.AccountView; **import** Model.CostRecord; **import** Model.CostRecordBuilder;  **public class** AccountManager {    **private** String **year**;  **private** String **month**;  **private** String **day**;    **private** ArrayList<String> **adayCostRecordIDList** = **new** ArrayList<String>();    **public** AccountManager(String year, String month , String day) {  **this**.**year** = year;  **this**.**month** = month;  **this**.**day** = day;  }   **public void** createTable() {  Connection c = **null**;  Statement stmt = **null**;  **try** {  Class.*forName*(**"org.sqlite.JDBC"**);  c = DriverManager.*getConnection*(**"jdbc:sqlite:Calendar.db"**);  System.***out***.println(**"Opened database successfully"**);   stmt = c.createStatement();  String sql = **"CREATE TABLE Account( ID INTEGER PRIMARY KEY AUTOINCREMENT ,YEAR TEXT NOT NULL,MONTH TEXT NOT NULL,DAY TEXT NOT NULL,CONTENT TEXT NOT NULL,COST TEXT NOT NULL, TYPE TEXT NOT NULL);"**;    stmt.executeUpdate(sql);  c.setAutoCommit(**false**);  c.commit();  stmt.close();  c.close();  } **catch** (Exception e) {  System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  System.*exit*(0);  }  System.***out***.println(**"Table created successfully"**);  }   **public void** executeSQL(String sqlStmt) {  Connection c = **null**;  Statement stmt = **null**;  **try** {  Class.*forName*(**"org.sqlite.JDBC"**);  c = DriverManager.*getConnection*(**"jdbc:sqlite:Calendar.db"**);  c.setAutoCommit(**false**);  System.***out***.println(**"Opened database successfully"**);   stmt = c.createStatement();  stmt.executeUpdate(sqlStmt);  c.commit();  stmt.close();  c.close();  } **catch** (Exception e) {  System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  System.*exit*(0);  }  System.***out***.println(**"Operation done successfully"**);  }   **public void** addCostRecord(CostRecord data) {  String sql = **"insert into Account( YEAR, MONTH, DAY,CONTENT, COST, TYPE )"** + **"VALUES(\'"** + data.getYear() + **"','"** + data.getMonth() + **"','"** + data.getDay() + **"','"** + data.getContent() + **"','"** + data.getcost() +   **"','"**+ data.gettype()+ **"\');"**;  executeSQL(sql);  setCostRecord();  }   **public void** editCostRecord(CostRecord data, String idEdit) {  String sql = **"update Account set CONTENT='"** + data.getContent() + **"',COST = '"** + data.getcost() +**"',TYPE='"** + data.gettype() +**"'where id = '"** +idEdit + **"';"**;   executeSQL(sql);  setCostRecord();  }    **public void** deleteDayCostRecord(String id) {  String sql = **"delete from Account where id ='"** + id + **"'"**;  executeSQL(sql);  setCostRecord();  }    **public** ArrayList<String> getadayCostRecordIDList(){  Connection c = **null**;  Statement stmt = **null**;  **try** {  Class.*forName*(**"org.sqlite.JDBC"**);  c = DriverManager.*getConnection*(**"jdbc:sqlite:Calendar.db"**);  c.setAutoCommit(**false**);  System.***out***.println(**"Opened database successfully"**);  stmt = c.createStatement();  System.***out***.println(**year**);  String sql = **"SELECT \* FROM Account where year='"** + **year** +**"'AND month='"** + **month** +**"'AND day='"** + **day** + **"';"**;  ResultSet rs = stmt.executeQuery(sql);  **adayCostRecordIDList**.clear();   **while** (rs.next()) {  **adayCostRecordIDList**.add( rs.getString(**"id"**));  }  rs.close();  stmt.close();  c.close();  } **catch** (Exception e) {  System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  System.*exit*(0);  }  **return adayCostRecordIDList**;  }    **public** CostRecord getIdCostRecord(String id){  Connection c = **null**;  Statement stmt = **null**;  CostRecord costRecord = **null**;  **try** {  Class.*forName*(**"org.sqlite.JDBC"**);  c = DriverManager.*getConnection*(**"jdbc:sqlite:Calendar.db"**);  c.setAutoCommit(**false**);  System.***out***.println(**"Opened database successfully"**);   stmt = c.createStatement();  String sql = **"SELECT \* FROM Account where id='"** + id +**"';"**;    ResultSet rs = stmt.executeQuery(sql);  costRecord = **new** CostRecordBuilder().year(rs.getString(**"year"**)).month(rs.getString(**"month"**)).day(rs.getString(**"day"**)).cost(rs.getString(**"cost"**)).type(rs.getString(**"type"**)).content(rs.getString(**"CONTENT"**)).id(rs.getString(**"id"**)).build();   rs.close();  stmt.close();  c.close();  } **catch** (Exception e) {  System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  System.*exit*(0);  }  System.***out***.println(**"Operation done successfully----------"**);  **return** costRecord;  }    **public void** setCostRecord(){  Connection c = **null**;  Statement stmt = **null**;  **try** {  Class.*forName*(**"org.sqlite.JDBC"**);  c = DriverManager.*getConnection*(**"jdbc:sqlite:Calendar.db"**);  c.setAutoCommit(**false**);  System.***out***.println(**"Opened database successfully"**);   stmt = c.createStatement();  System.***out***.println(**year**);  String sql = **"SELECT \* FROM Account where year='"** + **year** +**"'AND month='"** + **month** +**"'AND day='"** + **day** + **"';"**;    ResultSet rs = stmt.executeQuery(sql);  AccountIDView.*idBox*.removeAllItems();  DefaultComboBoxModel model = (DefaultComboBoxModel)AccountIDView.*idBox*.getModel();  DefaultTableModel tm = (DefaultTableModel)AccountView.*table*.getModel();  ResultSetMetaData rsmd = rs.getMetaData();  **adayCostRecordIDList**.clear();  tm.setColumnCount(0);  tm.setRowCount(0);  tm.addColumn(**"ID"**);   tm.addColumn(**"COST"**);   tm.addColumn(**"TYPE"**);   tm.addColumn(**"CONTENT"**);   **int** i = 1;  **while** (rs.next()) {  String[] a = **new** String[4];  a[0] = Integer.*toString*(i);  a[1] = rs.getString(**"cost"**);  a[2] = rs.getString(**"type"**);  a[3] = rs.getString(**"CONTENT"**);  tm.addRow(a);  i++;  **adayCostRecordIDList**.add( rs.getString(**"id"**));  }  **for** (**int** x = 0; x < **adayCostRecordIDList**.size(); x++) {  model.addElement(Integer.*toString*(x+1));  }  rs.close();  stmt.close();  c.close();  } **catch** (Exception e) {  System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  System.*exit*(0);  }  System.***out***.println(**"Operation done successfully"**);  }   **public static void** main(String args[]) { } } |

|  |
| --- |
| **CalendarManager.java** |
| **package** DB;  **import** java.sql.Connection;  **import** java.sql.DriverManager; **import** java.sql.ResultSet; **import** java.sql.SQLException; **import** java.sql.Statement; **import** java.util.ArrayList;  **import** CalendarView.WindowsReminder; **import** Model.Schedule; **import** Model.ScheduleBuilder;  **public class** CalendarManager {   **private** String **year**;  **private** String **month**;  **private** String **day**;   **private** ArrayList<Schedule> **allScheduleList** = **new** ArrayList<Schedule>();   **public** CalendarManager(String year, String month, String day) {  **this**.**year** = year;  **this**.**month** = month;  **this**.**day** = day;  }   **public void** createTable() {  Connection connection = **null**;  Statement stmt = **null**;  **try** {  Class.*forName*(**"org.sqlite.JDBC"**);  connection = DriverManager.*getConnection*(**"jdbc:sqlite:Calendar.db"**);  System.***out***.println(**"Opened database successfully"**);   stmt = connection.createStatement();  String sql = **"CREATE TABLE Schedule( ID INTEGER PRIMARY KEY AUTOINCREMENT ,YEAR TEXT NOT NULL,MONTH TEXT NOT NULL,DAY TEXT NOT NULL,CONTENT TEXT NOT NULL,TIME TEXT NOT NULL, NOTIFY BOOLEAN NOT NULL);"**;   stmt.executeUpdate(sql);  connection.commit();  stmt.close();  connection.close();  } **catch** (Exception e) {  System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  System.*exit*(0);  }  System.***out***.println(**"Table created successfully"**);  }   **public void** executeSQL(String sqlStmt) {  Connection connection = **null**;  Statement stmt = **null**;  **try** {  Class.*forName*(**"org.sqlite.JDBC"**);  connection = DriverManager.*getConnection*(**"jdbc:sqlite:Calendar.db"**);  connection.setAutoCommit(**false**);  System.***out***.println(**"Opened database successfully"**);   stmt = connection.createStatement();  stmt.executeUpdate(sqlStmt);  connection.commit();  stmt.close();  connection.close();  } **catch** (Exception e) {  System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  System.*exit*(0);  }  System.***out***.println(**"Operation done successfully"**);  }   **public void** addSchedule(Schedule data) {  String sql = **"insert into Schedule( YEAR, MONTH, DAY,CONTENT, TIME, NOTIFY )"** + **"VALUES(\'"** + data.getYear()  + **"','"** + data.getMonth() + **"','"** + data.getDay() + **"','"** + data.getContent() + **"','"** + data.getTime()  + **"','"** + data.getIsNotify() + **"\');"**;  executeSQL(sql);  **if**(data.getIsNotify() == **"true"**) {  Connection c = **null**;  Statement stmt = **null**;  Schedule reminderdata = **null**;  WindowsReminder testing = **null**;  **try** {  Class.*forName*(**"org.sqlite.JDBC"**);  c = DriverManager.*getConnection*(**"jdbc:sqlite:Calendar.db"**);  c.setAutoCommit(**false**);  System.***out***.println(**"Opened database successfully"**);   stmt = c.createStatement();  String sql2 = **"SELECT \* FROM Schedule"**;   ResultSet rs = stmt.executeQuery(sql2);  **while** (rs.next()) {  reminderdata = **new** ScheduleBuilder().year(**year**).month(**month**).day(**day**)  .isNotify(rs.getString(**"notify"**)).time(rs.getString(**"time"**)).id(rs.getString(**"id"**))  .content(rs.getString(**"CONTENT"**)).build();   }  rs.close();  stmt.close();  c.close();  } **catch** (Exception e) {  System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  System.*exit*(0);  }  System.***out***.println(reminderdata.getId());  testing = **new** WindowsReminder(reminderdata);  testing.setReminder();  }  }   **public void** editSchedule(Schedule data, String idEdit) {  Connection c = **null**;  Statement stmt = **null**;  Schedule originaldata = **null**;  WindowsReminder testing = **null**;  **try** {  Class.*forName*(**"org.sqlite.JDBC"**);  c = DriverManager.*getConnection*(**"jdbc:sqlite:Calendar.db"**);  c.setAutoCommit(**false**);  System.***out***.println(**"Opened database successfully"**);   stmt = c.createStatement();  String sqlOriginal = **"SELECT \* FROM Schedule where id ='"** + idEdit + **"'"**;   ResultSet rs = stmt.executeQuery(sqlOriginal);  **while** (rs.next()) {  originaldata = **new** ScheduleBuilder().year(**year**).month(**month**).day(**day**)  .isNotify(rs.getString(**"notify"**)).time(rs.getString(**"time"**)).id(rs.getString(**"id"**))  .content(rs.getString(**"CONTENT"**)).build();  }  rs.close();  stmt.close();  c.close();  } **catch** (Exception e) {  System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  System.*exit*(0);  }    **if**(!(originaldata.getIsNotify().equals(data.getIsNotify()))){  testing = **new** WindowsReminder(originaldata);  testing.editReminder();  }    String sql = **"update Schedule set CONTENT='"** + data.getContent() + **"',TIME = '"** + data.getTime() + **"',NOTIFY='"** + data.getIsNotify() + **"'where id = '"** + idEdit + **"';"**;  executeSQL(sql);    }   **public void** deleteDaySchedule(**int** dayNumId) {  setSchedule();  String id = getSchedule().get(dayNumId-1).getId();    String sql = **"delete from Schedule where id ='"** + id + **"' AND year='"** + **year** + **"'AND month='"** + **month** + **"'AND day='"** + **day** + **"';"**;  Connection c = **null**;  Statement stmt = **null**;  Schedule reminderdata = **null**;  WindowsReminder testing = **null**;  **try** {  Class.*forName*(**"org.sqlite.JDBC"**);  c = DriverManager.*getConnection*(**"jdbc:sqlite:Calendar.db"**);  c.setAutoCommit(**false**);  System.***out***.println(**"Opened database successfully"**);   stmt = c.createStatement();  String sql2 = **"SELECT \* FROM Schedule where id ='"** + id + **"'"**;   ResultSet rs = stmt.executeQuery(sql2);  **while** (rs.next()) {  reminderdata = **new** ScheduleBuilder().year(**year**).month(**month**).day(**day**)  .isNotify(rs.getString(**"notify"**)).time(rs.getString(**"time"**)).id(rs.getString(**"id"**))  .content(rs.getString(**"CONTENT"**)).build();   }  rs.close();  stmt.close();  c.close();  } **catch** (Exception e) {  System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  System.*exit*(0);  }  testing = **new** WindowsReminder(reminderdata);  testing.deleteReminder();  executeSQL(sql);  }   **public** ArrayList<Schedule> getSchedule() {  **return allScheduleList**;  }     **public** Schedule getIdSchdule(**int** dayNumId) {  Connection c = **null**;  Statement stmt = **null**;  Schedule schedule = **null** ;  setSchedule();  String id = getSchedule().get(dayNumId-1).getId();  **try** {  Class.*forName*(**"org.sqlite.JDBC"**);  c = DriverManager.*getConnection*(**"jdbc:sqlite:Calendar.db"**);  c.setAutoCommit(**false**);  System.***out***.println(**"Opened database successfully"**);   stmt = c.createStatement();  String sql = **"SELECT \* FROM Schedule where id='"** + id + **"' AND year='"** + **year** + **"'AND month='"** + **month** + **"'AND day='"** + **day** + **"';"**;  System.***out***.println(**"sql = "** + sql);   ResultSet rs = stmt.executeQuery(sql);  **while** (rs.next()) {  schedule = **new** ScheduleBuilder().year(**year**).month(**month**).day(**day**)  .isNotify(rs.getString(**"notify"**)).time(rs.getString(**"time"**)).id(rs.getString(**"id"**))  .content(rs.getString(**"CONTENT"**)).build();  }  rs.close();  stmt.close();  c.close();  } **catch** (Exception e) {  System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  System.*exit*(0);  }  System.***out***.println(**"Operation done successfully"**);  **return** schedule;  }   **public void** setSchedule() {  Connection c = **null**;  Statement stmt = **null**;  **allScheduleList**.clear();  **try** {  Class.*forName*(**"org.sqlite.JDBC"**);  c = DriverManager.*getConnection*(**"jdbc:sqlite:Calendar.db"**);  c.setAutoCommit(**false**);  System.***out***.println(**"Opened database successfully"**);   stmt = c.createStatement();  String sql = **"SELECT \* FROM Schedule where year='"** + **year** + **"'AND month='"** + **month** + **"'AND day='"** + **day** + **"' order by time ASC;"**;   ResultSet rs = stmt.executeQuery(sql);  **while** (rs.next()) {  Schedule schedule = **new** ScheduleBuilder().year(**year**).month(**month**).day(**day**)  .isNotify(rs.getString(**"notify"**)).time(rs.getString(**"time"**)).id(rs.getString(**"id"**))  .content(rs.getString(**"CONTENT"**)).build();   **allScheduleList**.add(schedule);  }  rs.close();  stmt.close();  c.close();  } **catch** (Exception e) {  System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  System.*exit*(0);  }  System.***out***.println(**"Operation done successfully"**);  }  } |

|  |
| --- |
| **CostRecord.java** |
| **package** Model;  **public class** CostRecord {  **private** String **year**;  **private** String **month**;  **private** String **day**;  **private** String **content**;  **private** String **type**;  **private** String **id**;  **private** String **cost**;    **public** CostRecord( String year,String month,String day,String content,String type,String id,String cost ){  **this**.**year** = year;  **this**.**month** = month;  **this**.**content** = content;  **this**.**type** = type;  **this**.**id** = id;  **this**.**cost** = cost;  **this**.**day** = day;  }  **public** String getId(){  **return id**;  }  **public void** setId(String id){  **this**.**id** = id;  }  **public** String gettype(){  **return type**;  }  **public void** settype(String type){  **this**.**type** = type;  }  **public** String getYear() {  **return year**;  }  **public void** setYear(String year) {  **this**.**year** = year;  }  **public** String getMonth() {  **return month**;  }  **public void** setMonth(String month) {  **this**.**month** = month;  }  **public** String getDay() {  **return day**;  }  **public void** setDay(String day) {  **this**.**day** = day;  }  **public** String getContent() {  **return content**;  }  **public void** setContent(String content) {  **this**.**content** = content;  }  **public** String getcost() {  **return cost**;  }  **public void** setcost(String cost) {  **this**.**cost** = cost;  } } |

|  |
| --- |
| **CostRecordBuilder.java** |
| **package** Model;  **public class** CostRecordBuilder {  **private** String **year**;  **private** String **month**;  **private** String **day**;  **private** String **content**;  **private** String **cost**;  **private** String **id**;  **private** String **type**;   **public** CostRecordBuilder year(String year) {  **this**.**year** = year;  **return this**;  }   **public** CostRecordBuilder month(String month) {  **this**.**month** = month;  **return this**;  }   **public** CostRecordBuilder day(String day) {  **this**.**day** = day;  **return this**;  }   **public** CostRecordBuilder content(String content) {  **this**.**content** = content;  **return this**;  }   **public** CostRecordBuilder cost(String cost) {  **this**.**cost** = cost;  **return this**;  }   **public** CostRecordBuilder type(String type) {  **this**.**type** = type;  **return this**;  }   **public** CostRecordBuilder id(String id) {  **this**.**id** = id;  **return this**;  }  **public** CostRecord build() {  **return new** CostRecord(**year**, **month**, **day**, **content**, **type**, **id**, **cost**);  }  } |

|  |
| --- |
| **Schedule.java** |
| **package** Model;  **public class** Schedule {  **private** String **year**;  **private** String **month**;  **private** String **day**;  **private** String **content**;  **private** String **time**;  **private** String **id**;  **private** String **isNotify**;    **public** Schedule( String year,String month,String day,String content,String time,String id,String isNotify ){  **this**.**year** = year;  **this**.**month** = month;  **this**.**content** = content;  **this**.**time** = time;  **this**.**id** = id;  **this**.**isNotify** = isNotify;  **this**.**day** = day;  }  **public** String getId(){  **return id**;  }   **public** String getYear() {  **return year**;  }  **public** String getMonth() {  **return month**;  }  **public** String getDay() {  **return day**;  }  **public** String getContent() {  **return content**;  }  **public** String getTime() {  **return time**;  }  **public** String getIsNotify() {  **return isNotify**;  }  **public void** setYear(String year) {  **this**.**year** = year;  }  **public void** setMonth(String month) {  **this**.**month** = month;  }  **public void** setDay(String day) {  **this**.**day** = day;  }  **public void** setContent(String content) {  **this**.**content** = content;  }  **public void** setTime(String time) {  **this**.**time** = time;  }  **public void** setId(String id) {  **this**.**id** = id;  }  **public void** setIsNotify(String isNotify) {  **this**.**isNotify** = isNotify;  } } |

|  |
| --- |
| **ScheduleBuilder.java** |
| **package** Model;  **public class** ScheduleBuilder {  **private** String **year**;  **private** String **month**;  **private** String **day**;  **private** String **content**;  **private** String **time**;  **private** String **id**;  **private** String **isNotify**;   **public** ScheduleBuilder year(String year) {  **this**.**year** = year;  **return this**;  }   **public** ScheduleBuilder month(String month) {  **this**.**month** = month;  **return this**;  }   **public** ScheduleBuilder day(String day) {  **this**.**day** = day;  **return this**;  }   **public** ScheduleBuilder content(String content) {  **this**.**content** = content;  **return this**;  }   **public** ScheduleBuilder time(String time) {  **this**.**time** = time;  **return this**;  }   **public** ScheduleBuilder id(String id) {  **this**.**id** = id;  **return this**;  }   **public** ScheduleBuilder isNotify(String isNotify) {  **this**.**isNotify** = isNotify;  **return this**;  }   **public** Schedule build() {  **return new** Schedule(**year**, **month**, **day**, **content**, **time**, **id**, **isNotify**);  }  } |

|  |
| --- |
| **FunctionChosenView.java** |
| **package** View;  **import** java.awt.event.ActionEvent; **import** java.awt.event.ActionListener; **import** java.util.ArrayList;  **import** javax.swing.JButton; **import** javax.swing.JComboBox; **import** javax.swing.JDialog; **import** javax.swing.JFrame; **import** javax.swing.JLabel;  **import** AccountView.AccountView; **import** CalendarView.CalendarView; **import** DB.AccountManager; **import** Model.CostRecord;  **public class** FunctionChosenView **extends** JDialog **implements** ActionListener{   **private** String **year**;  **private** String **month**;  **private** String **day**;  **private** String **id**;  **private** JButton **btncalendar**;  **private** JButton **btnaccount**;  **public** FunctionChosenView(String year, String month, String day) {  **this**.**year** = year;  **this**.**month** = month;  **this**.**day** = day;  init();  }   **public void** init(){  setBounds(100, 100, 255, 70);  setTitle(**"current day "** + **year** + **"/"** + **month** + **"/"** + **day**);  setLocationRelativeTo(**null**);  getContentPane().setLayout(**null**);  **btncalendar** = **new** JButton(**"Calendar"**);  **btncalendar**.addActionListener(**this**);  **btncalendar**.setBounds(0, 0, 120, 30);  getContentPane().add(**btncalendar**);   **btnaccount** = **new** JButton(**"Account"**);  **btnaccount**.addActionListener(**this**);  **btnaccount**.setBounds(120, 0, 120, 30);  getContentPane().add(**btnaccount**);  }   **public static void** main(String[] args) {  FunctionChosenView test = **new** FunctionChosenView(**"2018"**,**"5"**,**"5"**);  }   **public void** actionPerformed(ActionEvent e) {  **if** (e.getSource() == **btncalendar**) {  **try** {  CalendarView frame = **new** CalendarView(**year**, **month**, **day**);  frame.setLocationRelativeTo(**null**);  frame.setVisible(**true**);  frame.setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***);  *//* } **catch** (Exception ex) {  ex.printStackTrace();  }  }  **else if**( e.getSource() == **btnaccount** ) {  **try** {  AccountView frame = **new** AccountView(**year**, **month**, **day**);  frame.setLocationRelativeTo(**null**);  frame.setVisible(**true**);  frame.setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***);  *//* } **catch** (Exception ex) {  ex.printStackTrace();  }  }  }  } |

|  |
| --- |
| **IndexPage.java** |
| **package** View;  **import** java.awt.BorderLayout; **import** java.awt.Color; **import** java.awt.EventQueue; **import** java.awt.Font; **import** java.awt.GridLayout; **import** java.awt.event.ActionEvent; **import** java.awt.event.ActionListener; **import** java.awt.event.WindowAdapter; **import** java.awt.event.WindowEvent; **import** java.util.Arrays; **import** java.util.Date; **import** java.util.GregorianCalendar; **import** javax.swing.JButton; **import** javax.swing.JComboBox; **import** javax.swing.JFrame; **import** javax.swing.JLabel; **import** javax.swing.JPanel;  **public class** IndexPage **extends** JFrame **implements** ActionListener {   **private** JComboBox **MonthBox** = **new** JComboBox();  **private** JComboBox **YearBox** = **new** JComboBox();   **private** JLabel **YearLabel** = **new** JLabel(**"year: "**);  **private** JLabel **MonthLabel** = **new** JLabel(**"month: "**);   **private** JButton **button\_ok** = **new** JButton(**"check"**);  **private** JButton **button\_today** = **new** JButton(**"today"**);   **private** Date **now\_date** = **new** Date();   **private int now\_year** = **now\_date**.getYear() + 1900;  **private int now\_month** = **now\_date**.getMonth();  **private boolean todayFlag** = **false**;   **private** JButton[] **button\_day** = **new** JButton[42];  **private final** String[] **week** = { **"SUN"**, **"MON"**, **"TUE"**, **"WEN"**, **"THR"**, **"FRI"**, **"SAT"** };  **private** JButton[] **button\_week** = **new** JButton[7];  **private** String **year\_int** = **null**;  **private int month\_int**;   **public** IndexPage() {  **super**();  **this**.setTitle(**"Calendar"**);  **this**.init();  **this**.setLocation(500, 300);   **this**.setResizable(**false**);  pack();   }   **private void** init() {  Font font = **new** Font(**"Dialog"**, Font.***BOLD***, 16);  **YearLabel**.setFont(font);  **MonthLabel**.setFont(font);  **button\_ok**.setFont(font);  **button\_today**.setFont(font);   **for** (**int** i = **now\_year**; i <= **now\_year** + 1; i++) {  **YearBox**.addItem(i + **""**);  }  **YearBox**.setSelectedIndex(0);   **for** (**int** i = 1; i <= 12; i++) {  **MonthBox**.addItem(i + **""**);  }  **MonthBox**.setSelectedIndex(**now\_month**);   JPanel panel\_ym = **new** JPanel();  panel\_ym.add(**YearLabel**);  panel\_ym.add(**YearBox**);  panel\_ym.add(**MonthLabel**);  panel\_ym.add(**MonthBox**);  panel\_ym.add(**button\_ok**);  panel\_ym.add(**button\_today**);   **button\_ok**.addActionListener(**this**);  **button\_today**.addActionListener(**this**);   JPanel panel\_day = **new** JPanel();  *// 7\*7* panel\_day.setLayout(**new** GridLayout(7, 7, 3, 3));  **for** (**int** i = 0; i < 7; i++) {  **button\_week**[i] = **new** JButton(**" "**);  **button\_week**[i].setText(**week**[i]);  **button\_week**[i].setForeground(Color.***black***);  panel\_day.add(**button\_week**[i]);  }  **button\_week**[0].setForeground(Color.***red***);  **button\_week**[6].setForeground(Color.***red***);   **for** (**int** i = 0; i < 42; i++) {  **button\_day**[i] = **new** JButton(**" "**);   **button\_day**[i].addActionListener(**this**); *// \*\*\*\** panel\_day.add(**button\_day**[i]);  }   **this**.paintDay(); *// show current day* JPanel panel\_main = **new** JPanel();  panel\_main.setLayout(**new** BorderLayout());  panel\_main.add(panel\_day, BorderLayout.***SOUTH***);  panel\_main.add(panel\_ym, BorderLayout.***NORTH***);  getContentPane().add(panel\_main);   }   **private void** paintDay() {  System.***out***.println(**"todayflag = "** + **todayFlag**);  **if** (**todayFlag**) {  **year\_int** = **now\_year** + **""**;  **month\_int** = **now\_month**;  } **else** {  **year\_int** = **YearBox**.getSelectedItem().toString();  **month\_int** = **MonthBox**.getSelectedIndex();  System.***out***.println(**"yer: "** + **year\_int** + **"month = "** + **month\_int**);  }   **int** year\_sel = Integer.*parseInt*(**year\_int**) - 1900;  Date firstDay = **new** Date(year\_sel, **month\_int**, 1);  GregorianCalendar cal = **new** GregorianCalendar();  cal.setTime(firstDay);  **int** days = 0;  **int** day\_week = 0;   **if** (**month\_int** == 0 || **month\_int** == 2 || **month\_int** == 4 || **month\_int** == 6 || **month\_int** == 7 || **month\_int** == 9  || **month\_int** == 11) {  days = 31;  } **else if** (**month\_int** == 3 || **month\_int** == 5 || **month\_int** == 8 || **month\_int** == 10) {  days = 30;  } **else** {  **if** (cal.isLeapYear(year\_sel)) {  days = 29;  } **else** {  days = 28;  }  }   day\_week = firstDay.getDay();  **int** count = 1;   **for** (**int** i = day\_week; i < day\_week + days; count++, i++) {  **if** (i % 7 == 0 || (i + 1) % 7 == 0) {  **if** ((i == day\_week + **now\_date**.getDate() - 1) && **month\_int** == **now\_month** && (year\_sel == **now\_year** - 1900)) {  **button\_day**[i].setForeground(Color.***BLUE***);  **button\_day**[i].setText(count + **""**);  } **else** {  **button\_day**[i].setForeground(Color.***RED***);  **button\_day**[i].setText(count + **""**);  }  } **else** {  **if** ((i == day\_week + **now\_date**.getDate() - 1) && **month\_int** == **now\_month** && (year\_sel == **now\_year** - 1900)) {  **button\_day**[i].setForeground(Color.***BLUE***);  **button\_day**[i].setText(count + **""**);  } **else** {  **button\_day**[i].setForeground(Color.***BLACK***);  **button\_day**[i].setText(count + **""**);  }  }   }  **if** (day\_week == 0) {  **for** (**int** i = days; i < 42; i++) {  **button\_day**[i].setText(**""**);  }  } **else** {  **for** (**int** i = 0; i < day\_week; i++) {  **button\_day**[i].setText(**""**);  }  **for** (**int** i = day\_week + days; i < 42; i++) {  **button\_day**[i].setText(**""**);  }  }   }   **public void** actionPerformed(ActionEvent e) {  **if** (e.getSource() == **button\_ok**) {  **todayFlag** = **false**;  **this**.paintDay();  } **else if** (e.getSource() == **button\_today**) {  *// System.out.println( e.getSource() );* **todayFlag** = **true**;  **YearBox**.setSelectedIndex(0);  **MonthBox**.setSelectedIndex(**now\_month**);  **this**.paintDay();  } **else** {  **for** (**int** i = 0; i < 42; i++) {  **if** (e.getSource() == **button\_day**[i]) {   System.***out***.println(  **YearBox**.getSelectedItem().toString() + **" "** + **MonthBox**.getSelectedItem().toString());  Object source = e.getSource();  **if** (source **instanceof** JButton) {  JButton btn = (JButton) source;  **final** String butSrcTxt = btn.getText();  System.***out***.println(**"i = "** + butSrcTxt);  **if** (butSrcTxt != **""**) {  **try** {  FunctionChosenView frame = **new** FunctionChosenView(**YearBox**.getSelectedItem().toString(),  **MonthBox**.getSelectedItem().toString(), butSrcTxt);  frame.setVisible(**true**);  frame.setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***);  *//* } **catch** (Exception ex) {  ex.printStackTrace();  }  }  } *// if* }  } *// for* } *// else* }   **public static void** main(String[] args) {  **final** IndexPage ct = **new** IndexPage();  ct.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);   ct.setVisible(**true**);  } } |

|  |
| --- |
| **BarChart.java** |
| **package** Report;  **import** java.util.ArrayList;  **import** javax.swing.JPanel;  **import** org.jfree.chart.ChartFactory; **import** org.jfree.chart.ChartPanel; **import** org.jfree.chart.JFreeChart; **import** org.jfree.chart.plot.PlotOrientation; **import** org.jfree.data.category.CategoryDataset; **import** org.jfree.data.category.DefaultCategoryDataset;  **import** DB.AccountManager; **import** Model.CostRecord;  **public class** BarChart **implements** Chart {   AccountManager **accountManager**;    **public** BarChart(String year, String month) {  **accountManager** = **new** AccountManager(year, month, **""**);  }   **private** JFreeChart createChart() {  JFreeChart barChart = ChartFactory.*createBarChart*(**""**, **"Category"**, **"Cost"**, createDataset(),  PlotOrientation.***VERTICAL***, **true**, **true**, **false**);  **return** barChart;  }    **public int** getContentTypeCost(String searchType) {  ArrayList<CostRecord> montlyCostList = **accountManager**.getMonthlyCost();  **int** partCost = 0;  **for** (CostRecord costRecord : montlyCostList) {  **if** (costRecord.gettype().equals(searchType))  partCost += Integer.*parseInt*(costRecord.getcost());  }  **return** partCost;  }   **private** CategoryDataset createDataset() {  **final** DefaultCategoryDataset dataset = **new** DefaultCategoryDataset();  dataset.addValue(getContentTypeCost(**"Food"**), **"Food"**, **""**);  dataset.addValue(getContentTypeCost(**"Clothing"**), **"Clothing"**, **""**);  dataset.addValue(getContentTypeCost(**"Housing"**), **"Housing"**, **""**);  dataset.addValue(getContentTypeCost(**"Transportation"**), **"Transportation"**, **""**);  dataset.addValue(getContentTypeCost(**"Entertainment"**), **"Entertainment"**, **""**);  **return** dataset;  }    **public** JPanel generate() {  ChartPanel chartPanel = **new** ChartPanel(createChart());  **return** chartPanel;  } } |

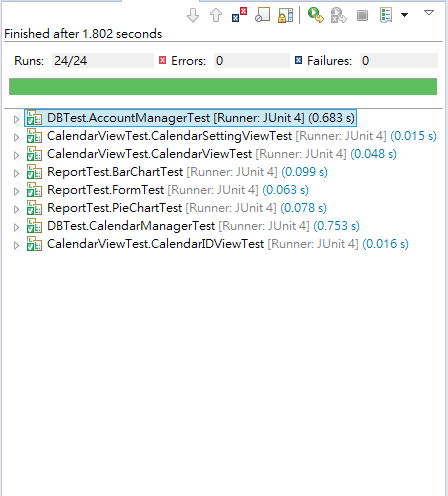
|  |
| --- |
| **Chart.java** |
| **package** Report;  **import** javax.swing.JPanel;  **interface** Chart {  **public** JPanel generate();  } |

|  |
| --- |
| **ChartFactory.java** |
| **package** Report;  **public class** ChartFactory {    **public** Chart getChart( String chartType, String year, String month){    **if**( chartType == **null** ) **return null**;  **else** {   **if**( chartType.equalsIgnoreCase(**"PieChart"**) )   **return new** PieChart(year, month);  **else if**( chartType.equalsIgnoreCase(**"BarChart"**) )  **return new** BarChart(year, month);  }    **return null**;    } } |

|  |
| --- |
| **DetailReportView.java** |
| **package** Report;  **import** java.awt.BorderLayout; **import** java.awt.Color; **import** java.awt.Font; **import** java.awt.event.ActionEvent; **import** java.awt.event.ActionListener;  **import** javax.swing.JButton; **import** javax.swing.JFrame; **import** javax.swing.JLabel; **import** javax.swing.JPanel; **import** javax.swing.JScrollPane; **import** javax.swing.JTable; **import** javax.swing.table.DefaultTableModel;  **public class** DetailReportView **extends** JFrame **implements** ActionListener {   **private** String **year**;  **private** String **month**;  **private** Form **form**;   **private** JButton **pieChart**;  **private** JPanel **panel**;  **private** JScrollPane **scrollPane**;  **private** JButton **monthTable**;  **private** JButton **barChart**;  **private** JButton **exportFile**;   **public** DetailReportView(String year, String month) {  **this**.**month** = month;  **this**.**year** = year;  **form** = **new** Form(year, month);  init();  }   **public void** init() {   JPanel panelButton = **new** JPanel();   **monthTable** = **new** JButton();  **monthTable**.setText(**"Table"**);  panelButton.add(**monthTable**);  **monthTable**.addActionListener(**this**);   **pieChart** = **new** JButton();  **pieChart**.setText(**"Pie Chart"**);  **pieChart**.addActionListener(**this**);  panelButton.add(**pieChart**);   **barChart** = **new** JButton();  **barChart**.setText(**"Bar Chart"**);  **barChart**.addActionListener(**this**);  panelButton.add(**barChart**);   **exportFile** = **new** JButton();  **exportFile**.setText(**"ExportFile"**);  **exportFile**.addActionListener(**this**);  *// panelButton.add(exportFile);  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\** setTitle(**year** + **"/"** + **month** + **" Monthly cost"**);  **this**.getContentPane().setBackground(Color.***YELLOW***);  setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);   JTable table = **new** JTable(setTableModel());  table.setEnabled(**false**); *//* **scrollPane** = **new** JScrollPane(table);  table.setFillsViewportHeight(**true**);  table.setRowHeight(50);  table.setFont(**new** Font(**"Serif"**, Font.***BOLD***, 20));   JPanel panelBelow = **new** JPanel();  JLabel label = **new** JLabel(**"total cost $"** + **form**.getMontlyTotalCost() + **" "**);  label.setForeground(Color.***BLUE***);  label.setFont(**new** Font(**"Serif"**, Font.***BOLD***, 24));  panelBelow.add(label);  panelBelow.add(**exportFile**);   setSize(700, 600);  **panel** = **new** JPanel();  **panel**.setLayout(**new** BorderLayout());  **panel**.add(**scrollPane**, BorderLayout.***CENTER***);  **panel**.add(panelButton, BorderLayout.***NORTH***);  **panel**.add(panelBelow, BorderLayout.***SOUTH***);   getContentPane().add(**panel**);  setVisible(**true**);  }   **private** DefaultTableModel setTableModel() {  String[] columns = { **"TYPE"**, **"COST"**, **"CONTENT"** };  Object[][] data = **form**.readDataFromManager();  DefaultTableModel model = **new** DefaultTableModel(data, columns);  **return** model;  }   **public void** actionPerformed(ActionEvent e) {   **if** (e.getSource() == **exportFile**) {  **form**.exportAsExcelFile();  } **else if** (**form**.**montlyCostListSize** != 0) {   BorderLayout layout = (BorderLayout) **panel**.getLayout();  **panel**.remove(layout.getLayoutComponent(BorderLayout.***CENTER***));   **if** (e.getSource() == **monthTable**) {  **panel**.add(**scrollPane**, BorderLayout.***CENTER***);  setVisible(**true**);  } **else if** (e.getSource() == **pieChart**) {  showChart(**"pieChart"**);  } **else if** (e.getSource() == **barChart**) {  showChart(**"barChart"**);  }  }   }   **public void** showChart(String chartName) {  Chart pieChart = **new** ChartFactory().getChart(chartName, **year**, **month**);  **panel**.add(pieChart.generate(), BorderLayout.***CENTER***);  setVisible(**true**);  }   } |

|  |
| --- |
| **Form.java** |
| **package** Report;  **import** java.io.File; **import** java.io.FileNotFoundException; **import** java.io.FileOutputStream; **import** java.io.IOException; **import** java.util.ArrayList;  **import** javax.swing.JOptionPane;  **import** org.apache.poi.ss.usermodel.Cell; **import** org.apache.poi.ss.usermodel.Row; **import** org.apache.poi.xssf.usermodel.XSSFSheet; **import** org.apache.poi.xssf.usermodel.XSSFWorkbook;  **import** DB.AccountManager; **import** Model.CostRecord;  **public class** Form {   **private** String **FILE\_NAME** ; *//MyFirstExcel.xlsx";* **public int montlyCostListSize**;  AccountManager **accountManager**;    **private** String **year**;  **private** String **month**;   **public** Form(String year, String month) {  **accountManager** = **new** AccountManager(year, month, **""**);  **montlyCostListSize** = 0;  **FILE\_NAME** = **""**;  **this**.**year** = year;  **this**.**month** = month;  }   **public** String[][] readDataFromManager() {   ArrayList<CostRecord> montlyCostList = **accountManager**.getMonthlyCost();  String[][] mydata = **new** String[montlyCostList.size()][3];  **montlyCostListSize** = montlyCostList.size();   **for** (**int** i = 0; i < montlyCostList.size(); i++) {  mydata[i][0] = montlyCostList.get(i).gettype();  mydata[i][1] = montlyCostList.get(i).getcost();  mydata[i][2] = montlyCostList.get(i).getContent();  }  **return** mydata;  }   **public int** getMontlyTotalCost() {  **return accountManager**.getMontlyTotalCost();  }    **public void** createFile() {   **FILE\_NAME** = **"ReportExport/"** + **year** + **"\_"** + **month** + **"Report.xlsx"**;  File file = **new** File( **FILE\_NAME** );  **if**(!file.exists())  {  file.getParentFile().mkdirs();  **try** {  file.createNewFile();  } **catch** (IOException e) {  *//* ***TODO Auto-generated catch block*** e.printStackTrace();  }  }  }   **public void** exportAsExcelFile() {    createFile();    XSSFWorkbook workbook = **new** XSSFWorkbook();  XSSFSheet sheet = workbook.createSheet(**"Datatypes in Java"**);  Object[][] datatypes = readDataFromManager();   **int** rowNum = 0;  System.***out***.println(**"Creating excel"**);   **for** (Object[] datatype : datatypes) {  Row row = sheet.createRow(rowNum++);  **int** colNum = 0;  **for** (Object field : datatype) {  Cell cell = row.createCell(colNum++);  **if** (field **instanceof** String)   cell.setCellValue((String) field);   }  }   **try** {  FileOutputStream outputStream = **new** FileOutputStream(**FILE\_NAME**);  workbook.write(outputStream);  workbook.close();  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }   System.***out***.println(**"Done"**);  JOptionPane.*showMessageDialog*(**null**, **"DONE"**);  }   } |

|  |
| --- |
| **PieChart.java** |
| **package** Report;  **import** org.jfree.chart.ChartFactory; **import** org.jfree.chart.ChartPanel; **import** org.jfree.chart.JFreeChart; **import** org.jfree.data.general.DefaultPieDataset; **import** org.jfree.data.general.PieDataset;  **import** DB.AccountManager; **import** Model.CostRecord;  **import** java.util.ArrayList;  **import** javax.swing.JPanel;  **public class** PieChart **implements** Chart {   AccountManager **accountManager**;   **public** PieChart(String year, String month) {  **accountManager** = **new** AccountManager(year, month, **""**);  }   **public double** getContentScale(String searchType) {  ArrayList<CostRecord> monthlyCostList = **accountManager**.getMonthlyCost();  **int** totalCost = **accountManager**.getMontlyTotalCost();  **int** partCost = 0;  **for** (CostRecord costRecord : monthlyCostList) {  **if** (costRecord.gettype().equals(searchType))  partCost += Integer.*parseInt*(costRecord.getcost());  }  **return** (partCost \* 100) / totalCost;  }   **private** PieDataset createDataset() {  DefaultPieDataset dataset = **new** DefaultPieDataset();  dataset.setValue(**"Food"**, getContentScale(**"Food"**));  dataset.setValue(**"Clothing"**, getContentScale(**"Clothing"**));  dataset.setValue(**"Housing"**, getContentScale(**"Housing"**));  dataset.setValue(**"Transportation"**, getContentScale(**"Transportation"**));  dataset.setValue(**"Entertainment"**, getContentScale(**"Entertainment"**));  **return** dataset;  }   **private** JFreeChart createChart(PieDataset dataset) {  JFreeChart chart = ChartFactory.*createPieChart*(**""**, *// chart // title* dataset, *// data* **true**, *// include legend* **true**, **false**);   **return** chart;  }    **public** JPanel generate() {  JFreeChart chart = createChart(createDataset());  **return new** ChartPanel(chart);  }  } |

1. **Unit Testing**
   1. **Snapshot of Testing Result**
   2. **Unit Testing Code Listing(\*New)**

|  |
| --- |
| **AccountManagerTest.java** |
| **package** DBTest;  **import** java.sql.Connection; **import** java.sql.DriverManager; **import** java.sql.ResultSet; **import** java.sql.SQLException; **import** java.sql.Statement; **import** java.util.ArrayList;  **import** org.junit.After; **import** org.junit.Before; **import** org.junit.Test;  **import** AccountView.AccountView; **import** DB.AccountManager; **import** Model.CostRecord; **import** Model.CostRecordBuilder; **import** junit.framework.Assert;  **public class** AccountManagerTest {  AccountManager **accountManager**;  **private** String **year**;  **private** String **month**;  **private** String **day**;  AccountView **accountView**;   @Before  **public void** setUp(){  **year** = **"2018"**;  **month** = **"5"**;  **day** = **"11"**;  **accountManager** = **new** AccountManager(**year**, **month**, **day**);  **accountView** = **new** AccountView(**year**, **month**, **day**);  **accountView**.init();  }  @After  **public void** tearDown(){  Connection c = **null**;  Statement stmt = **null**;  **try** {  Class.*forName*(**"org.sqlite.JDBC"**);  c = DriverManager.*getConnection*(**"jdbc:sqlite:Calendar.db"**);  System.***out***.println(**"Opened database successfully"**);   stmt = c.createStatement();  String sql = **"DELETE FROM Account WHERE CONTENT LIKE 'Test%';"**;   stmt.executeUpdate(sql);  c.setAutoCommit(**false**);  c.commit();  stmt.close();  c.close();  } **catch** (Exception e) {  System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  System.*exit*(0);  }  System.***out***.println(**"Test data in table was deleted successfully"**);  }   @Test  **public void** addCostRecordTest(){  CostRecord costRecord = **new** CostRecordBuilder()  .year(**year**)  .month(**month**)  .day(**day**)  .content(**"Test for add"**)  .cost(**"100"**)  .type(**"Food"**)  .build();   **accountManager**.addCostRecord(costRecord);   Connection c = **null**;  Statement stmt = **null**;  **try** {  Class.*forName*(**"org.sqlite.JDBC"**);  c = DriverManager.*getConnection*(**"jdbc:sqlite:Calendar.db"**);  System.***out***.println(**"Opened database successfully"**);   stmt = c.createStatement();  String sql = **"SELECT \* FROM Account WHERE YEAR = \"2018\" "** + **"AND MONTH = \"5\" "** + **"AND DAY = \"11\" "** + **"AND CONTENT = \"Test for add\" "** + **"AND COST = \"100\" "** + **"AND TYPE = \"Food\";"**;  ResultSet rs = stmt.executeQuery(sql);  rs.getString(**"YEAR"**);*//For checking whether the data exists* } **catch** (SQLException e){  Assert.*fail*(**"Adding costRecord was failed."**);  e.printStackTrace();  } **catch** (Exception e) {  e.printStackTrace();  System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  System.*exit*(0);  } **finally** {  **try** {  **if** (stmt != **null**)  stmt.close();  **if** (c != **null**)  c.close();  } **catch** (SQLException e) {  *//* ***TODO Auto-generated catch block*** e.printStackTrace();  }  }  }   @Test  **public void** editCostRecordTest(){  */\*ADD THE TEST DATA FIRST\*/* CostRecord costRecord = **new** CostRecordBuilder()  .year(**year**)  .month(**month**)  .day(**day**)  .content(**"Test for edit"**)  .cost(**"100"**)  .type(**"Food"**)  .build();   **accountManager**.addCostRecord(costRecord);   Connection c = **null**;  Statement stmt = **null**;  String ID = **null**;  **try** {  Class.*forName*(**"org.sqlite.JDBC"**);  c = DriverManager.*getConnection*(**"jdbc:sqlite:Calendar.db"**);  System.***out***.println(**"Opened database successfully"**);   stmt = c.createStatement();  String sql = **"SELECT \* FROM Account WHERE YEAR = \"2018\" "** + **"AND MONTH = \"5\" "** + **"AND DAY = \"11\" "** + **"AND CONTENT = \"Test for edit\" "** + **"AND COST = \"100\" "** + **"AND TYPE = \"Food\";"**;  ResultSet rs = stmt.executeQuery(sql);  ID = rs.getString(**"ID"**);  } **catch** (Exception e) {  e.printStackTrace();  System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  System.*exit*(0);  } **finally** {  **try** {  **if** (stmt != **null**)  stmt.close();  **if** (c != **null**)  c.close();  } **catch** (SQLException e) {  *//* ***TODO Auto-generated catch block*** e.printStackTrace();  }  }  */\*THEN EDIT THE DATA CREATED ABOVE\*/* costRecord.settype(**"Housing"**);  **accountManager**.editCostRecord(costRecord, ID);   Connection connForEdit = **null**;  Statement stmtForEdit = **null**;  **try** {  Class.*forName*(**"org.sqlite.JDBC"**);  connForEdit = DriverManager.*getConnection*(**"jdbc:sqlite:Calendar.db"**);  System.***out***.println(**"Opened database successfully"**);   stmtForEdit = connForEdit.createStatement();  String sql = **"SELECT \* FROM Account WHERE YEAR = \"2018\" "** + **"AND MONTH = \"5\" "** + **"AND DAY = \"11\" "** + **"AND CONTENT = \"Test for edit\" "** + **"AND COST = \"100\" "** + **"AND TYPE = \"Housing\";"**;  ResultSet rs = stmtForEdit.executeQuery(sql);  String actualType = rs.getString(**"TYPE"**);  Assert.*assertEquals*(**"Housing"**, actualType);  } **catch** (Exception e) {  e.printStackTrace();  System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  System.*exit*(0);  } **finally** {  **try** {  **if** (stmtForEdit != **null**)  stmtForEdit.close();  **if** (connForEdit != **null**)  connForEdit.close();  } **catch** (SQLException e) {  *//* ***TODO Auto-generated catch block*** e.printStackTrace();  }  }  }   @Test  **public void** deleteCostRecordTest(){  */\*ADD THE TEST DATA FIRST\*/* CostRecord costRecord = **new** CostRecordBuilder()  .year(**year**)  .month(**month**)  .day(**day**)  .content(**"Test for delete"**)  .cost(**"100"**)  .type(**"Food"**)  .build();   **accountManager**.addCostRecord(costRecord);   Connection c = **null**;  Statement stmt = **null**;  String ID = **null**;  **try** {  Class.*forName*(**"org.sqlite.JDBC"**);  c = DriverManager.*getConnection*(**"jdbc:sqlite:Calendar.db"**);  System.***out***.println(**"Opened database successfully"**);   stmt = c.createStatement();  String sql = **"SELECT \* FROM Account WHERE YEAR = \"2018\" "** + **"AND MONTH = \"5\" "** + **"AND DAY = \"11\" "** + **"AND CONTENT = \"Test for delete\" "** + **"AND COST = \"100\" "** + **"AND TYPE = \"Food\";"**;  ResultSet rs = stmt.executeQuery(sql);  ID = rs.getString(**"ID"**);  } **catch** (Exception e) {  e.printStackTrace();  System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  System.*exit*(0);  } **finally** {  **try** {  **if** (stmt != **null**)  stmt.close();  **if** (c != **null**)  c.close();  } **catch** (SQLException e) {  *//* ***TODO Auto-generated catch block*** e.printStackTrace();  }  }  */\*THEN DELETE THE DATA CREATED ABOVE\*/* **accountManager**.deleteDayCostRecord(ID);   Connection connForDelete = **null**;  Statement stmtForDelete = **null**;  **try** {  Class.*forName*(**"org.sqlite.JDBC"**);  connForDelete = DriverManager.*getConnection*(**"jdbc:sqlite:Calendar.db"**);  System.***out***.println(**"Opened database successfully"**);   stmtForDelete = connForDelete.createStatement();  String sql = **"SELECT \* FROM Account WHERE YEAR = \"2018\" "** + **"AND MONTH = \"5\" "** + **"AND DAY = \"11\" "** + **"AND CONTENT = \"Test for delete\" "** + **"AND COST = \"100\" "** + **"AND TYPE = \"Food\";"**;  ResultSet rs = stmtForDelete.executeQuery(sql);  rs.getString(**"ID"**);*//For Checking whether the data exists* } **catch** (SQLException e){  Assert.*assertEquals*(**"ResultSet closed"**, e.getMessage());  } **catch** (Exception e) {  e.printStackTrace();  System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  System.*exit*(0);  } **finally** {  **try** {  **if** (stmtForDelete != **null**)  stmtForDelete.close();  **if** (connForDelete != **null**)  connForDelete.close();  } **catch** (SQLException e) {  *//* ***TODO Auto-generated catch block*** e.printStackTrace();  }  }  }   @Test  **public void** getADayCostRecordIDListTest(){  */\*ADD TEST DATA FIRST\*/* CostRecord costRecord = **new** CostRecordBuilder()  .year(**year**)  .month(**month**)  .day(**day**)  .content(**"Test for getAdayCostRecordIDList"**)  .cost(**"100"**)  .type(**"Food"**)  .build();   **accountManager**.addCostRecord(costRecord);   ArrayList<String> costRecordIDList = **accountManager**.getadayCostRecordIDList();  **int** actualListSize = costRecordIDList.size();  Assert.*assertEquals*(1, actualListSize);  }   @Test  **public void** getIDCostRecordTest(){*/\*ADD THE TEST DATA FIRST\*/* CostRecord costRecord = **new** CostRecordBuilder()  .year(**year**)  .month(**month**)  .day(**day**)  .content(**"Test for getIDCostRecord"**)  .cost(**"100"**)  .type(**"Food"**)  .build();   **accountManager**.addCostRecord(costRecord);   Connection c = **null**;  Statement stmt = **null**;  **try** {  Class.*forName*(**"org.sqlite.JDBC"**);  c = DriverManager.*getConnection*(**"jdbc:sqlite:Calendar.db"**);  System.***out***.println(**"Opened database successfully"**);   stmt = c.createStatement();  String sql = **"SELECT \* FROM Account WHERE YEAR = \"2018\" "** + **"AND MONTH = \"5\" "** + **"AND DAY = \"11\" "** + **"AND CONTENT = \"Test for getIDCostRecord\" "** + **"AND COST = \"100\" "** + **"AND TYPE = \"Food\";"**;  ResultSet rs = stmt.executeQuery(sql);  costRecord.setId(rs.getString(**"ID"**));  } **catch** (Exception e) {  e.printStackTrace();  System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  System.*exit*(0);  } **finally** {  **try** {  **if** (stmt != **null**)  stmt.close();  **if** (c != **null**)  c.close();  } **catch** (SQLException e) {  *//* ***TODO Auto-generated catch block*** e.printStackTrace();  }  }   CostRecord actualCostRecord = **accountManager**.getIdCostRecord(costRecord.getId());  String actualYear = actualCostRecord.getYear();  String actualMonth = actualCostRecord.getMonth();  String actualDay = actualCostRecord.getDay();  String actualContent = actualCostRecord.getContent();  String actualCost = actualCostRecord.getcost();  String actualType = actualCostRecord.gettype();    Assert.*assertEquals*(**"2018"**, actualYear);  Assert.*assertEquals*(**"5"**, actualMonth);  Assert.*assertEquals*(**"11"**, actualDay);  Assert.*assertEquals*(**"Test for getIDCostRecord"**, actualContent);  Assert.*assertEquals*(**"100"**, actualCost);  Assert.*assertEquals*(**"Food"**, actualType);  }    @Test  **public void** getMonthlyCostTest(){  CostRecord costRecordStub = **new** CostRecordBuilder()  .year(**year**)  .month(**month**)  .day(**day**)  .content(**"Test for getMonthlyCost"**)  .cost(**"100"**)  .type(**"Food"**)  .build();  CostRecord costRecordStub2 = **new** CostRecordBuilder()  .year(**year**)  .month(**month**)  .day(**day**)  .content(**"Test for getMonthlyCost"**)  .cost(**"200"**)  .type(**"Clothing"**)  .build();    **accountManager**.addCostRecord(costRecordStub);  **accountManager**.addCostRecord(costRecordStub2);    ArrayList<CostRecord> monthlyCostRecord =   **accountManager**.getMonthlyCost();  ArrayList<CostRecord> actualMonthlyCostRecord =   **new** ArrayList<CostRecord>();    **for** (CostRecord costRecordForTest : monthlyCostRecord) {  **if** (costRecordForTest.getContent().equals(**"Test for getMonthlyCost"**))  actualMonthlyCostRecord.add(costRecordForTest);  }    String actualContentStub = actualMonthlyCostRecord.get(0).getContent();  String actualCostStub = actualMonthlyCostRecord.get(0).getcost();  String actualTypeStub = actualMonthlyCostRecord.get(0).gettype();    String actualContentStub2 = actualMonthlyCostRecord.get(1).getContent();  String actualCostStub2 = actualMonthlyCostRecord.get(1).getcost();  String actualTypeStub2 = actualMonthlyCostRecord.get(1).gettype();    Assert.*assertEquals*(**"Test for getMonthlyCost"**, actualContentStub);  Assert.*assertEquals*(**"100"**, actualCostStub);  Assert.*assertEquals*(**"Food"**, actualTypeStub);    Assert.*assertEquals*(**"Test for getMonthlyCost"**, actualContentStub2);  Assert.*assertEquals*(**"200"**, actualCostStub2);  Assert.*assertEquals*(**"Clothing"**, actualTypeStub2);  } } |

|  |
| --- |
| **CalendarManagerTest.java** |
| **package** DBTest;  **import** java.io.BufferedReader; **import** java.io.FileNotFoundException; **import** java.io.FileReader; **import** java.io.IOException; **import** java.io.RandomAccessFile; **import** java.sql.Connection; **import** java.sql.DriverManager; **import** java.sql.ResultSet; **import** java.sql.SQLException; **import** java.sql.Statement; **import** java.util.ArrayList;  **import** org.junit.After; **import** org.junit.Before; **import** org.junit.Test;  **import** DB.CalendarManager; **import** Model.Schedule; **import** Model.ScheduleBuilder;  **import** junit.framework.Assert;  **public class** CalendarManagerTest {  CalendarManager **calendarManager**;  **private** String **year**;  **private** String **month**;  **private** String **day**;   @Before  **public void** setUp() {  **year** = **"2018"**;  **month** = **"5"**;  **day** = **"11"**;  **calendarManager** = **new** CalendarManager(**year**, **month**, **day**);  }   @After  **public void** tearDown() {  **calendarManager**.setSchedule();  ArrayList<Schedule> data = **calendarManager**.getSchedule();  **for** (Schedule schedule : data) {  **if** (schedule.getContent().contains(**"Test"**)) {  **calendarManager**.deleteDaySchedule(data.size());  System.***out***.println(**"Test data in table was deleted successfully"**);  }  }  }   @Test  **public void** addScheduleTest() {  Schedule schedule = **new** ScheduleBuilder()  .year(**year**)  .month(**month**)  .day(**day**)  .content(**"Test for add"**)  .time(**"1400-1500"**)  .isNotify(**"false"**)  .build();  **calendarManager**.addSchedule(schedule);   Connection c = **null**;  Statement stmt = **null**;  **try** {  Class.*forName*(**"org.sqlite.JDBC"**);  c = DriverManager.*getConnection*(**"jdbc:sqlite:Calendar.db"**);  System.***out***.println(**"Opened database successfully"**);   stmt = c.createStatement();  String sql = **"SELECT \* FROM Schedule WHERE YEAR = \"2018\" "** + **"AND MONTH = \"5\" "** + **"AND DAY = \"11\" "** + **"AND CONTENT = \"Test for add\" "** + **"AND TIME = \"1400-1500\" "** + **"AND NOTIFY = \"false\";"**;  ResultSet rs = stmt.executeQuery(sql);  rs.getString(**"YEAR"**);*// For checking whether the data exists* } **catch** (SQLException e) {  Assert.*fail*(**"Adding schedule was failed"**);  e.printStackTrace();  } **catch** (Exception e) {  e.printStackTrace();  System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  System.*exit*(0);  } **finally** {  **try** {  **if** (stmt != **null**)  stmt.close();  **if** (c != **null**)  c.close();  } **catch** (SQLException e) {  e.printStackTrace();  }  }  }   @Test  **public void** addScheduleWithRemindTest() {  Schedule schedule = **new** ScheduleBuilder()  .year(**year**)  .month(**month**)  .day(**day**)  .content(**"Test for add"**)  .time(**"1400-1500"**)  .isNotify(**"true"**)  .build();   **calendarManager**.addSchedule(schedule);   String filePath = **"reminderrecord.txt"**;  BufferedReader br = **null**;  **try** {  br = **new** BufferedReader(**new** FileReader(filePath));  StringBuilder sb = **new** StringBuilder();  String fileContent = br.readLine();   **while** (fileContent != **null**) {  sb.append(fileContent);  sb.append(**"\n"**);  fileContent = br.readLine();  }   String expectExistedContent = schedule.getYear() + **"/"** + ((schedule.getMonth().length() > 1) ? schedule.getMonth() : **"0"** + schedule.getMonth()) + **"/"** + schedule.getDay() + **" "** + schedule.getTime().substring(0, 2) + **":"** + schedule.getTime().substring(2, 4) + **":00"** + **" "** + schedule.getContent();   String fileContentInString = sb.toString();    System.***out***.println(**"fileContentInString = "** + fileContentInString);  System.***out***.println(**"expect = "** + expectExistedContent);   Assert.*assertTrue*(fileContentInString.contains(expectExistedContent));   } **catch** (FileNotFoundException e) {  System.***err***.println(**"Need to check whether the "** + filePath + **" exists"**);  } **catch** (IOException e) {  e.printStackTrace();  } **finally** {  RandomAccessFile raf = **null**;  **try** {  **if** (br != **null**)  br.close();  *//Restore "reminderrecord.txt" to the original state* raf = **new** RandomAccessFile(filePath, **"rw"**);  **long** length = raf.length() - 1;  **byte** b;  **do** {  length -= 1;*//Get rid of the EOF character* raf.seek(length);*//Point to the corresponding location* b = raf.readByte();  } **while** (b != 10 && length > 0);*//byte = 10 : escape character* raf.setLength(length + 1);  } **catch** (IOException e) {  e.printStackTrace();  } **finally** {  **try** {  **if** (raf != **null**)  raf.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }  }  }   @Test  **public void** editScheduleTest() {  */\* ADD THE TEST DATA FIRST \*/* Schedule schedule = **new** ScheduleBuilder()  .year(**year**)  .month(**month**)  .day(**day**)  .content(**"Test for edit"**)  .time(**"1400-1500"**)  .isNotify(**"false"**)  .build();    **calendarManager**.addSchedule(schedule);   Connection c = **null**;  Statement stmt = **null**;  String ID = **null**;  **try** {  Class.*forName*(**"org.sqlite.JDBC"**);  c = DriverManager.*getConnection*(**"jdbc:sqlite:Calendar.db"**);  System.***out***.println(**"Opened database successfully"**);   stmt = c.createStatement();  String sql = **"SELECT \* FROM Schedule WHERE YEAR = \"2018\" "** + **"AND MONTH = \"5\" "** + **"AND DAY = \"11\" "** + **"AND CONTENT = \"Test for edit\" "** + **"AND TIME = \"1400-1500\" "** + **"AND NOTIFY = \"false\";"**;  ResultSet rs = stmt.executeQuery(sql);  ID = rs.getString(**"ID"**);  } **catch** (Exception e) {  e.printStackTrace();  System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  System.*exit*(0);  } **finally** {  **try** {  **if** (stmt != **null**)  stmt.close();  **if** (c != **null**)  c.close();  } **catch** (SQLException e) {  e.printStackTrace();  }  }  */\* THEN EDIT THE DATA CREATED ABOVE \*/* schedule.setTime(**"1600-1700"**);  **calendarManager**.editSchedule(schedule, ID);   Connection connForEdit = **null**;  Statement stmtForEdit = **null**;  **try** {  Class.*forName*(**"org.sqlite.JDBC"**);  connForEdit = DriverManager.*getConnection*(**"jdbc:sqlite:Calendar.db"**);  System.***out***.println(**"Opened database successfully"**);   stmtForEdit = connForEdit.createStatement();  String sql = **"SELECT \* FROM Schedule WHERE YEAR = \"2018\" "** + **"AND MONTH = \"5\" "** + **"AND DAY = \"11\" "** + **"AND CONTENT = \"Test for edit\" "** + **"AND TIME = \"1600-1700\" "** + **"AND NOTIFY = \"false\";"**;  ResultSet rs = stmtForEdit.executeQuery(sql);  String actualTime = rs.getString(**"TIME"**);  Assert.*assertEquals*(**"1600-1700"**, actualTime);  } **catch** (Exception e) {  e.printStackTrace();  System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  System.*exit*(0);  } **finally** {  **try** {  **if** (stmtForEdit != **null**)  stmtForEdit.close();  **if** (connForEdit != **null**)  connForEdit.close();  } **catch** (SQLException e) {  e.printStackTrace();  }  }  }   @Test  **public void** editScheduleWithReminderTest() {  Schedule schedule = **new** ScheduleBuilder()  .year(**year**)  .month(**month**)  .day(**day**)  .content(**"Test for edit"**)  .time(**"1400-1500"**)  .isNotify(**"false"**)  .build();   **calendarManager**.addSchedule(schedule);    Connection c = **null**;  Statement stmt = **null**;  String ID = **null**;  **try** {  Class.*forName*(**"org.sqlite.JDBC"**);  c = DriverManager.*getConnection*(**"jdbc:sqlite:Calendar.db"**);  System.***out***.println(**"Opened database successfully"**);   stmt = c.createStatement();  String sql = **"SELECT \* FROM Schedule WHERE YEAR = \"2018\" "** + **"AND MONTH = \"5\" "** + **"AND DAY = \"11\" "** + **"AND CONTENT = \"Test for edit\" "** + **"AND TIME = \"1400-1500\" "** + **"AND NOTIFY = \"false\";"**;  ResultSet rs = stmt.executeQuery(sql);  ID = rs.getString(**"ID"**);  } **catch** (Exception e) {  e.printStackTrace();  System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  System.*exit*(0);  } **finally** {  **try** {  **if** (stmt != **null**)  stmt.close();  **if** (c != **null**)  c.close();  } **catch** (SQLException e) {  e.printStackTrace();  }  }    schedule.setIsNotify(**"true"**);  **calendarManager**.editSchedule(schedule, ID);   String filePath = **"reminderrecord.txt"**;  BufferedReader br = **null**;  **try** {  br = **new** BufferedReader(**new** FileReader(filePath));  StringBuilder sb = **new** StringBuilder();  String fileContent = br.readLine();   **while** (fileContent != **null**) {  sb.append(fileContent);  sb.append(**"\n"**);  fileContent = br.readLine();  }   String expectExistedContent = schedule.getYear() + **"/"** + ((schedule.getMonth().length() > 1) ? schedule.getMonth() : **"0"** + schedule.getMonth()) + **"/"** + schedule.getDay() + **" "** + schedule.getTime().substring(0, 2) + **":"** + schedule.getTime().substring(2, 4) + **":00"** + **" "** + schedule.getContent();   String fileContentInString = sb.toString();    System.***out***.println(**"fileContentInString = "** + fileContentInString);  System.***out***.println(**"expect = "** + expectExistedContent);   Assert.*assertTrue*(fileContentInString.contains(expectExistedContent));   } **catch** (FileNotFoundException e) {  System.***err***.println(**"Need to check whether the "** + filePath + **" exists"**);  } **catch** (IOException e) {  e.printStackTrace();  } **finally** {  RandomAccessFile raf = **null**;  **try** {  **if** (br != **null**)  br.close();  *//Restore "reminderrecord.txt" to the original state* raf = **new** RandomAccessFile(filePath, **"rw"**);  **long** length = raf.length() - 1;  **byte** b;  **do** {  length -= 1;*//Get rid of the EOF character* raf.seek(length);*//Point to the corresponding location* b = raf.readByte();  } **while** (b != 10 && length > 0);*//byte = 10 : escape character* raf.setLength(length + 1);  } **catch** (IOException e) {  e.printStackTrace();  } **finally** {  **try** {  **if** (raf != **null**)  raf.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }  }  }    @Test  **public void** deleteScheduleTest() {  */\* ADD THE TEST DATA FIRST \*/* Schedule schedule = **new** ScheduleBuilder()  .year(**year**)  .month(**month**)  .day(**day**)  .content(**"Test for delete"**)  .time(**"1400-1500"**)  .isNotify(**"false"**)  .build();    **calendarManager**.addSchedule(schedule);   */\* THEN DELETE THE DATA CREATED ABOVE \*/* **calendarManager**.setSchedule();  ArrayList<Schedule> data = **calendarManager**.getSchedule();  **calendarManager**.deleteDaySchedule(data.size());  Connection connForDelete = **null**;  Statement stmtForDelete = **null**;  **try** {  Class.*forName*(**"org.sqlite.JDBC"**);  connForDelete = DriverManager.*getConnection*(**"jdbc:sqlite:Calendar.db"**);  System.***out***.println(**"Opened database successfully"**);   stmtForDelete = connForDelete.createStatement();  String sql = **"SELECT \* FROM Schedule WHERE YEAR = \"2018\" "** + **"AND MONTH = \"5\" "** + **"AND DAY = \"11\" "** + **"AND CONTENT = \"Test for delete\" "** + **"AND TIME = \"1400-1500\" "** + **"AND NOTIFY = \"false\";"**;  ResultSet rs = stmtForDelete.executeQuery(sql);  rs.getString(**"ID"**);*// For Checking whether the data exists* } **catch** (SQLException e) {  Assert.*assertEquals*(**"ResultSet closed"**, e.getMessage());  } **catch** (Exception e) {  e.printStackTrace();  System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  System.*exit*(0);  } **finally** {  **try** {  **if** (stmtForDelete != **null**)  stmtForDelete.close();  **if** (connForDelete != **null**)  connForDelete.close();  } **catch** (SQLException e) {  e.printStackTrace();  }  }  } } |

|  |
| --- |
| **CalendarIDViewTest.java** |
| **package** CalendarViewTest;  **import static** org.junit.Assert.\*;  **import** org.junit.After; **import** org.junit.Before; **import** org.junit.Test;  **import** CalendarView.CalendarIDView; **import** CalendarView.CalendarView; **import** junit.framework.Assert;  **public class** CalendarIDViewTest {   CalendarIDView **calendarIDView**;    @Before  **public void** setUp() **throws** Exception {  CalendarView calendarView = **null**;  **calendarIDView** = **new** CalendarIDView(**"2018"**, **"5"**, **"6"**, calendarView, 5);  }   @After  **public void** tearDown() **throws** Exception {  **calendarIDView** = **null**;  }   @Test  **public void** setDayScheduleItemSizeTest(){  Integer[] result = **calendarIDView**.setDayScheduleItemSize();  *assertEquals*(1, result[0].intValue());  *assertEquals*(2, result[1].intValue());  *assertEquals*(3, result[2].intValue());  *assertEquals*(4, result[3].intValue());  *assertEquals*(5, result[4].intValue());  }    @Test(expected = IndexOutOfBoundsException.**class**)  **public void** setDayScheduleItemSizeIndexOutOfBoundsExceptionTest() {  Integer[] result = **calendarIDView**.setDayScheduleItemSize();  result[6].intValue();  } } |

|  |
| --- |
| **CalendarSettingView.java** |
| **package** CalendarViewTest;  **import static** org.junit.Assert.\*;  **import** org.junit.After; **import** org.junit.Before; **import** org.junit.Test;  **import** CalendarView.CalendarSettingView; **import** CalendarView.CalendarView;  **public class** CalendarSettingViewTest {   CalendarSettingView **calendarSettingView**;   @Before  **public void** setUp() {  CalendarView calendarView = **null**;  **calendarSettingView** = **new** CalendarSettingView(**"2018"**, **"5"**, **"6"**, calendarView);  }   @After  **public void** tearDown() **throws** Exception {  }   @Test  **public void** testCheckTimeFormat() {  *assertFalse*(**calendarSettingView**.checkTimeFormat(**"12"**));  *assertFalse*(**calendarSettingView**.checkTimeFormat(**"120000"**));  *assertFalse*(**calendarSettingView**.checkTimeFormat(**"hell"**));  *assertFalse*(**calendarSettingView**.checkTimeFormat(**""**));  *assertFalse*(**calendarSettingView**.checkTimeFormat(**"2500"**));  *assertFalse*(**calendarSettingView**.checkTimeFormat(**"2460"**));  *assertFalse*(**calendarSettingView**.checkTimeFormat(**"2410"**));  }   @Test  **public void** testCheckTimeFormatTrue() {  *assertTrue*(**calendarSettingView**.checkTimeFormat(**"1200"**));  *assertTrue*(**calendarSettingView**.checkTimeFormat(**"1233"**));  }   @Test  **public void** testStartTimeMustSmallerThanEndTime() {  *assertTrue*(**calendarSettingView**.startTimeMustSmallerThanEndTime( **"1200"**, **"1300"** ) );  *assertTrue*(**calendarSettingView**.startTimeMustSmallerThanEndTime( **"2359"**, **"2400"** ) );  *assertTrue*(**calendarSettingView**.startTimeMustSmallerThanEndTime( **"0000"**, **"1359"** ) );  *assertTrue*(**calendarSettingView**.startTimeMustSmallerThanEndTime( **"1200"**, **"1200"** ) );    *assertFalse*(**calendarSettingView**.startTimeMustSmallerThanEndTime( **"1200"**, **"1100"** ) );  *assertFalse*(**calendarSettingView**.startTimeMustSmallerThanEndTime( **"1200"**, **"1159"** ) );  *assertFalse*(**calendarSettingView**.startTimeMustSmallerThanEndTime( **"1210"**, **"1200"** ) );  *assertFalse*(**calendarSettingView**.startTimeMustSmallerThanEndTime( **"2400"**, **"2330"** ) );  }    @Test  **public void** testConfirmTheDataInputFormat() {  **boolean** timeLengthIsOne = **calendarSettingView**.confirmTheDataInputFormat(**"test"**, **"2000"**, **"1"**);  *assertFalse*(timeLengthIsOne);    **boolean** sartTimeCannotBeEnglish = **calendarSettingView**.confirmTheDataInputFormat(**"test"**, **"ss"**, **"1300"**);  *assertFalse*(sartTimeCannotBeEnglish);    **boolean** startTimeIsGreaterThanEndTime = **calendarSettingView**.confirmTheDataInputFormat(**"test"**, **"1310"**, **"1300"**);  *assertFalse*(startTimeIsGreaterThanEndTime);    **boolean** endTimeCannotBeEnglish = **calendarSettingView**.confirmTheDataInputFormat(**"test"**, **"1310"**, **"ss"**);  *assertFalse*(endTimeCannotBeEnglish);    **boolean** contentCannotBeEmpty = **calendarSettingView**.confirmTheDataInputFormat(**""**, **"1300"**, **"1400"**);  *assertFalse*(contentCannotBeEmpty);     **boolean** contentExceedLength = **calendarSettingView**.confirmTheDataInputFormat(**"hello, i am a test content and i am over 20 words."**, **"1300"**, **"1400"**);  *assertFalse*(contentExceedLength);   }    @Test  **public void** testConfirmTheDataInputFormatTrue(){  **boolean** test1 = **calendarSettingView**.confirmTheDataInputFormat(**"test~"**, **"1300"**, **"1400"**);  *assertTrue*(test1);    **boolean** test2 = **calendarSettingView**.confirmTheDataInputFormat(**"test~"**, **"2230"**, **"2300"**);  *assertTrue*(test2);    **boolean** test3 = **calendarSettingView**.confirmTheDataInputFormat(**"this is a test"**, **"1800"**, **"2400"**);  *assertTrue*(test3);  } } |

|  |
| --- |
| **CalendarViewTest.java** |
| **package** CalendarViewTest;  **import static** org.junit.Assert.\*;  **import** java.util.ArrayList;  **import** org.junit.After; **import** org.junit.Before; **import** org.junit.Test;  **import** CalendarView.CalendarView; **import** Model.Schedule; **import** Model.ScheduleBuilder; **import** junit.framework.Assert;  **public class** CalendarViewTest {   CalendarView **calendarView**;  ArrayList<Schedule> **scheduleList**;   **private** String **year**;  **private** String **month**;  **private** String **day**;   @Before  **public void** setUp() **throws** Exception {  **calendarView** = **new** CalendarView(**"2018"**, **"5"**, **"7"**);  **scheduleList** = **new** ArrayList<Schedule>();   **this**.**year** = **"2018"**;  **this**.**month** = **"5"**;  **this**.**day** = **"7"**;  }   @After  **public void** tearDown() **throws** Exception {  }   @Test  **public void** getdataTest\_One() {  Schedule schedule = **new** ScheduleBuilder().year(**year**).month(**month**).day(**day**).time(**"1200-1500"**)  .content(**"test content"**).isNotify(**"true"**).build();    **scheduleList**.add(schedule);  String[][] data = **calendarView**.getdata(**scheduleList**);   *assertEquals*(**"1"**, data[0][0]);  *assertEquals*(**"1200-1500"**, data[0][1]);  *assertEquals*(**"test content"**, data[0][2]);  *assertEquals*(**"O"**, data[0][3]);  }   @Test(expected = IndexOutOfBoundsException.**class**)  **public void** getdataErrorTest() {  String[][] data = **calendarView**.getdata(**scheduleList**);  System.***out***.println(data[0][4]);  }    @Test  **public void** getdataTest\_Two() {  Schedule schedule = **new** ScheduleBuilder().year(**year**).month(**month**).day(**day**).time(**"1200-1500"**)  .content(**"test content"**).isNotify(**"true"**).build();    Schedule schedule2 = **new** ScheduleBuilder().year(**year**).month(**month**).day(**day**).time(**"1300-1500"**)  .content(**"test content second"**).isNotify(**"false"**).build();    **scheduleList**.add(schedule);  **scheduleList**.add(schedule2);    String[][] data = **calendarView**.getdata(**scheduleList**);   *assertEquals*(**"1"**, data[0][0]);  *assertEquals*(**"1200-1500"**, data[0][1]);  *assertEquals*(**"test content"**, data[0][2]);  *assertEquals*(**"O"**, data[0][3]);    *assertEquals*(**"2"**, data[1][0]);  *assertEquals*(**"1300-1500"**, data[1][1]);  *assertEquals*(**"test content second"**, data[1][2]);  *assertEquals*(**"X"**, data[1][3]);    **try**{  System.***out***.println(data[2][0]);  *fail*(**"expected IndexOutOfBoundsException"**);  } **catch**(IndexOutOfBoundsException e) {  }   }  } |

|  |
| --- |
| **BarChartTest.java** |
| **package** ReportTest;  **import** java.sql.Connection; **import** java.sql.DriverManager; **import** java.sql.Statement; **import** java.util.ArrayList;  **import** org.junit.After; **import** org.junit.Assert; **import** org.junit.Before; **import** org.junit.Test;  **import** AccountView.AccountView; **import** DB.AccountManager; **import** Model.CostRecordBuilder; **import** Report.BarChart;  **public class** BarChartTest {  AccountManager **accountManager**;  **private** String **year**;  **private** String **month**;  **private** String **day**;  AccountView **accountView**;  BarChart **barChart**;    @Before  **public void** setUp() {  **year** = **"2018"**;  **month** = **"7"**;  **day** = **"1"**;  **accountManager** = **new** AccountManager(**year**, **month**, **day**);  **accountView** = **new** AccountView(**year**, **month**, **day**);  **barChart** = **new** BarChart(**year**, **month**);  **accountView**.init();   }    @After  **public void** tearDown() {  Connection c = **null**;  Statement stmt = **null**;  **try** {  Class.*forName*(**"org.sqlite.JDBC"**);  c = DriverManager.*getConnection*(**"jdbc:sqlite:Calendar.db"**);  System.***out***.println(**"Opened database successfully"**);    stmt = c.createStatement();  String sql = **"DELETE FROM Account WHERE CONTENT LIKE 'Test%';"**;    stmt.executeUpdate(sql);  c.setAutoCommit(**false**);  c.commit();  stmt.close();  c.close();  } **catch** (Exception e) {  System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  System.*exit*(0);  }  System.***out***.println(**"Test data in table was deleted successfully"**);  }  @Test  **public void** getContentScaleTest() {  **accountManager**.addCostRecord(**new** CostRecordBuilder()  .year(**year**)  .month(**month**)  .day(**day**)  .content(**"Test for getContentScale"**)  .cost(**"100"**)  .type(**"Food"**)  .build());  **accountManager**.addCostRecord(**new** CostRecordBuilder()  .year(**year**)  .month(**month**)  .day(**day**)  .content(**"Test for getContentScale"**)  .cost(**"200"**)  .type(**"Food"**)  .build());  **accountManager**.addCostRecord(**new** CostRecordBuilder()  .year(**year**)  .month(**month**)  .day(**day**)  .content(**"Test for getContentScale"**)  .cost(**"100"**)  .type(**"Clothing"**)  .build());  **accountManager**.addCostRecord(**new** CostRecordBuilder()  .year(**year**)  .month(**month**)  .day(**day**)  .content(**"Test for getContentScale"**)  .cost(**"1000"**)  .type(**"Entertainment"**)  .build());  **accountManager**.addCostRecord(**new** CostRecordBuilder()  .year(**year**)  .month(**month**)  .day(**day**)  .content(**"Test for getContentScale"**)  .cost(**"800"**)  .type(**"Housing"**)  .build());  **int** actualFoodScale = **barChart**.getContentTypeCost(**"Food"**);  **int** actualClothingScale = **barChart**.getContentTypeCost(**"Clothing"**);  **int** actualEntertainmentScale = **barChart**.getContentTypeCost(**"Entertainment"**);  **int** actualHousingScale = **barChart**.getContentTypeCost(**"Housing"**);  Assert.*assertEquals*(300, actualFoodScale);  Assert.*assertEquals*(100, actualClothingScale);  Assert.*assertEquals*(1000, actualEntertainmentScale);  Assert.*assertEquals*(800, actualHousingScale);   } } |

|  |
| --- |
| **FormTest.java** |
| **package** ReportTest;  **import** java.sql.Connection; **import** java.sql.DriverManager; **import** java.sql.Statement;  **import** org.junit.After; **import** org.junit.Assert; **import** org.junit.Before; **import** org.junit.Test;  **import** AccountView.AccountView; **import** DB.AccountManager; **import** Model.CostRecordBuilder; **import** Report.Form;  **public class** FormTest {  AccountManager **accountManager**;  **private** String **year**;  **private** String **month**;  **private** String **day**;  AccountView **accountView**;  Form **form**;    @Before  **public void** setUp() {  **year** = **"2018"**;  **month** = **"7"**;  **day** = **"1"**;  **accountManager** = **new** AccountManager(**year**, **month**, **day**);  **accountView** = **new** AccountView(**year**, **month**, **day**);  **form** = **new** Form(**year**, **month**);  **accountView** = **new** AccountView(**year**, **month**, **day**);  **accountView**.init();  }    @After  **public void** tearDown() {  Connection c = **null**;  Statement stmt = **null**;  **try** {  Class.*forName*(**"org.sqlite.JDBC"**);  c = DriverManager.*getConnection*(**"jdbc:sqlite:Calendar.db"**);  System.***out***.println(**"Opened database successfully"**);    stmt = c.createStatement();  String sql = **"DELETE FROM Account WHERE CONTENT LIKE 'Test%';"**;    stmt.executeUpdate(sql);  c.setAutoCommit(**false**);  c.commit();  stmt.close();  c.close();  } **catch** (Exception e) {  System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  System.*exit*(0);  }  System.***out***.println(**"Test data in table was deleted successfully"**);  }    @Test  **public void** readDataFromManagerTest() {  **accountManager**.addCostRecord(**new** CostRecordBuilder()  .year(**year**)  .month(**month**)  .day(**day**)  .content(**"Test for readDataFromManager"**)  .cost(**"100"**)  .type(**"Food"**)  .build());  **accountManager**.addCostRecord(**new** CostRecordBuilder()  .year(**year**)  .month(**month**)  .day(**day**)  .content(**"Test for readDataFromManager"**)  .cost(**"100"**)  .type(**"Clothing"**)  .build());    String[][] actualData = **form**.readDataFromManager();  String actualType1 = actualData[0][0];  String actualCost1 = actualData[0][1];  String actualContent1 = actualData[0][2];  String actualType2 = actualData[1][0];  String actualCost2 = actualData[1][1];  String actualContent2 = actualData[1][2];    Assert.*assertEquals*(**"Food"**, actualType1);  Assert.*assertEquals*(**"100"**, actualCost1);  Assert.*assertEquals*(**"Test for readDataFromManager"**, actualContent1);  Assert.*assertEquals*(**"Clothing"**, actualType2);  Assert.*assertEquals*(**"100"**, actualCost2);  Assert.*assertEquals*(**"Test for readDataFromManager"**, actualContent2);  } } |

|  |
| --- |
| **PieChartTest.java** |
| **package** ReportTest;  **import** java.sql.Connection; **import** java.sql.DriverManager; **import** java.sql.Statement; **import** java.util.ArrayList;  **import** org.junit.After; **import** org.junit.Assert; **import** org.junit.Before; **import** org.junit.Test;  **import** AccountView.AccountView; **import** DB.AccountManager; **import** Model.CostRecordBuilder; **import** Report.PieChart;  **public class** PieChartTest {  AccountManager **accountManager**;  **private** String **year**;  **private** String **month**;  **private** String **day**;  AccountView **accountView**;  PieChart **pieChart**;    @Before  **public void** setUp() {  **year** = **"2018"**;  **month** = **"7"**;  **day** = **"1"**;  **accountManager** = **new** AccountManager(**year**, **month**, **day**);  **accountView** = **new** AccountView(**year**, **month**, **day**);  **pieChart** = **new** PieChart(**year**, **month**);  **accountView**.init();   }    @After  **public void** tearDown() {  Connection c = **null**;  Statement stmt = **null**;  **try** {  Class.*forName*(**"org.sqlite.JDBC"**);  c = DriverManager.*getConnection*(**"jdbc:sqlite:Calendar.db"**);  System.***out***.println(**"Opened database successfully"**);   stmt = c.createStatement();  String sql = **"DELETE FROM Account WHERE CONTENT LIKE 'Test%';"**;   stmt.executeUpdate(sql);  c.setAutoCommit(**false**);  c.commit();  stmt.close();  c.close();  } **catch** (Exception e) {  System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  System.*exit*(0);  }  System.***out***.println(**"Test data in table was deleted successfully"**);  }  @Test  **public void** getContentScaleTest() {  **accountManager**.addCostRecord(**new** CostRecordBuilder()  .year(**year**)  .month(**month**)  .day(**day**)  .content(**"Test for getContentScale"**)  .cost(**"100"**)  .type(**"Food"**)  .build());  **accountManager**.addCostRecord(**new** CostRecordBuilder()  .year(**year**)  .month(**month**)  .day(**day**)  .content(**"Test for getContentScale"**)  .cost(**"200"**)  .type(**"Food"**)  .build());  **accountManager**.addCostRecord(**new** CostRecordBuilder()  .year(**year**)  .month(**month**)  .day(**day**)  .content(**"Test for getContentScale"**)  .cost(**"100"**)  .type(**"Clothing"**)  .build());  **accountManager**.addCostRecord(**new** CostRecordBuilder()  .year(**year**)  .month(**month**)  .day(**day**)  .content(**"Test for getContentScale"**)  .cost(**"1000"**)  .type(**"Entertainment"**)  .build());  **accountManager**.addCostRecord(**new** CostRecordBuilder()  .year(**year**)  .month(**month**)  .day(**day**)  .content(**"Test for getContentScale"**)  .cost(**"800"**)  .type(**"Housing"**)  .build());  Double actualFoodScale = **pieChart**.getContentScale(**"Food"**);  Double actualClothingScale = **pieChart**.getContentScale(**"Clothing"**);  Double actualEntertainmentScale = **pieChart**.getContentScale(**"Entertainment"**);  Double actualHousingScale = **pieChart**.getContentScale(**"Housing"**);  Assert.*assertEquals*(13, actualFoodScale, 0.00001);  Assert.*assertEquals*(4, actualClothingScale, 0.00001);  Assert.*assertEquals*(45, actualEntertainmentScale, 0.00001);  Assert.*assertEquals*(36, actualHousingScale, 0.00001);   } } |

1. **Measurement**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **10659038 李柏霖** | | **106598048 郭士銓** | | **106598053 陳韻文** | |
| HW#1 | | | | | |
| 日期/時間 | 事件 | 日期/時間 | 事件 | 日期/時間 | 事件 |
| 2018/3/6 15:00~16:30  16:50~17:40 | 討論專案題目、  Problem Statement | 2018/3/6 15:00~16:30  16:50~17:40 | 討論專案題目、  Problem Statement | 2018/3/6 15:00~16:30  16:50~17:40 | 討論專案題目、  Problem Statement |
| 2018/3/8 16:40~17:30 | 撰寫Problem Statement | 2018/3/8 16:40~17:30 | 撰寫Problem Statement | 2018/3/8 16:40~17:30 | 撰寫Problem Statement |
| Total Time | 3hrs 10mins | Total Time | 3hrs 10mins | Total Time | 3hrs 10mins |
| HW#2 | | | | | |
| 2018/3/15  17:03~18:52 | 撰寫System features | 2018/3/15  17:03~18:52 | 撰寫System features | 2018/3/16  10:00~12:00  14:00~15:00  16:00~18:00 | 撰寫System Context Diagram、Use Case |
| 2018/3/16  10:00~12:10  14:00~15:00  16:00~17:59 | 撰寫System Context Diagram、Use Case FEA-01~FEA-04 | 2018/3/16  10:00~12:10  14:00~15:00  16:00~17:59 | 撰寫System Context Diagram、Use Case FEA-01~FEA-04 | 2018/3/19  10:15~10:55 | 撰寫Non-functional Requirements and Constraints、Glossary |
| 2018/3/19  10:15~10:55 | 撰寫Non-functional Requirements and Constraints、Glossary | 2018/3/19  10:15~10:55 | 撰寫Non-functional Requirements and Constraints、Glossary |  |  |
| 2018/3/20  11:05~11:40  17:20~17:49 | Problem  Statement 修改、Software Environments修改 | 2018/3/20  11:05~11:40  17:20~17:49 | Problem Statement 修改、Software Environments修改 |  |  |
| Total Time | 7hrs 37mins | Total Time | 7hrs 37mins | Total Time | 5hrs 40min |
| HW#3 | | | | | |
| 2018/3/29  17:00~18:06 | 定義concepts | 2018/3/29  17:00~18:06 | 定義concepts | 2018/3/29  17:00~18:06 | 定義concepts |
| 2018/3/30  10:00~12:00  14:30~15:07  16:00~18:00 | 定義concepts、設計Domain Model | 2018/3/30  10:00~12:00  14:30~15:07  16:00~18:00 | 定義concepts、設計Domain Model | 2018/3/30  10:00~12:00  14:30~15:07  16:00~18:00 | 定義concepts、設計Domain Model |
| 2018/4/2  10:20~10:40  17:30~18:16 | 設計Domain Model、Glossary修改 | 2018/4/2  10:20~10:40  17:30~18:16 | 設計Domain Model、Glossary修改 | 2018/4/2  10:20~10:40  17:30~18:16 | 設計Domain Model、Glossary修改 |
| Total | 6hrs 49mins | Total | 6hrs 49mins | Total | 6hrs 49mins |
| HW#4 | | | | | |
| 2018/4/13  10:00~12:10  17:20~17:39 | 撰寫Logical Architecture、設計Class Diagram | 2018/4/13  10:00~12:10  17:20~17:39 | 撰寫Logical Architecture、設計Class Diagram | 2018/4/13  10:00~12:00 | 撰寫Logical Architecture、設計Class Diagram |
| 2018/4/16  10:00~12:02  15:48~18:28 | 設計Class Diagram | 2018/4/16  10:00~12:02  15:48~18:28 | 設計Class Diagram | 2018/4/16  15:30-18:00 | 設計Class Diagram |
| 2018/4/17  10:35~11:57  15:00~18:01 | 設計Class Diagram | 2018/4/17  10:35~11:57  15:00~18:01 | 設計Class Diagram | 2018/4/17  10:30~12:00  16:30-17:30 | 設計Class Diagram |
| 2018/4/18  14:00~15:12 | 設計Class Diagram | 2018/4/18  14:00~15:12 | 設計Class Diagram | 2018/4/19  14:00~15:00  17:00~19:00 | 設計Class Diagram |
| 2018/4/19  14:00~15:12  16:50~19:20 | 設計Class Diagram | 2018/4/19  14:00~15:12  16:50~19:20 | 設計Class Diagram | 2018/4/22  20:00-22:00 | 設計System Sequence Diagram、Sequence Diagram |
| 2018/4/23  10:00~14:00  15:00~18:00  20:00~22:00 | 設計System Sequence Diagram、Sequence Diagram | 2018/4/22  11:20~14:30  15:30~19:00  20:30~23:00 | 設計System Sequence Diagram、Sequence Diagram | 2018/4/23  10:00-12:00  14:00-18:00 | 設計System Sequence Diagram、Sequence Diagram、System Context Diagram修改、 |
| 2018/4/24  11:00~12:00  14:00~18:00  20:00~22:00 | 設計System Sequence Diagram、Sequence Diagram、System Context | 2018/4/23  10:00~12:22  14:00~18:07  21:00~22:08 | 設計System Sequence Diagram、Sequence Diagram、System Context Diagram修改、新增目錄、頁碼 | 2018/4/24  10:00-10:30  11:30-12:00 | System Feature修改、Use Case修改、Domain Model修改、撰寫Contract |
| 2018/5/3  14:00~15:00  17:00~17:30 | 設計Sequence Diagram | 2018/4/24  10:00~10:40  11:20~12:10 | System Feature修改、Use Case修改、Domain Model修改、撰寫Contract | 2018/5/2  22:00-23:30 | sequence diagram add pattern on it |
|  |  | 2018/4/25  09:40~10:30  1400:15:12 | Measurement修改 | 2018/5/3  14:00-15:00  17:00-17:30 | sequence diagram add pattern on it  and write timelog |
|  |  | 2018/5/2  14:00~15:06  18:35~19:29 | 設計Sequence Diagram |  |  |
|  |  | 2018/5/3  14:20~15:10  16:40~18:22 | 設計Sequence Diagram、整理文件內容 |  |  |
| Total | 28hrs 10mins | Total | 40hrs 51mins | Total | 25hrs 10mins |
| HW#5 | | | | | |
| 2018/04/28  13:00-18:00 | 完成AccountManager和Accountview | 2018/5/5  14:00~18:00  20:30~24:00 | 撰寫測試 | 2018/4/21  10:00-22:00 | Write code |
| 2018/04/30  09:00-12:00 | 完成  Accountsettingview | 2018/5/6  11:00~14:00  15:00~18:00  20:30~24:00 | 撰寫測試 | 2018/4/29  10:00-23:00 | write code |
| 2018/05/03  10:00-12:00 | 完成  AccountIDview | 2018/5/7  10:00~12:04  14:00~15:20  16:00~17:50 | code review、撰寫實作 | 2018/4/28  10:00-23:00 | Write code |
| 2018/05/04  10:00-12:00  15:00-18:00 | 完成Functionchosenview和修改bug | 2018/5/8  15:00~15:48  18:40~21:18 | 撰寫實作(改變accountManagerTest中Test case data的initial方式)、撰寫作業5 Snapshot of testing result, unit testing code listing sequence diagram refinement | 2018/5/4  16:00-19:00 | Write document |
| 2018/05/05  13:00-18:00 | 完成例外處理和修改view的畫面 | 2018/5/10  14:00~15:10  17:00~17:39  21:30~23:29 | 撰寫作業五class diagram、system sequence diagram and sequence diagram refinement | 2018/5/5  10:00-23:00 | Write code |
| 2018/05/06  14:00-18:00 | 完成例外處理和修改view的畫面 | 2018/5/11  09:00~12:03  12:30~15:10  16:10~18:35 | sequence diagram refinement、document refinement | 2018/5/6  10:00-23:00 | Write code and unit test |
| 2018/05/08  19:00-21:00 | 修改class diagram |  |  | 2018/5/7  14:00-18:00 | Write project |
| 2018/05/09  14:00-15:00 | 完成有關複數新增頁面的例外處理 |  |  | 2018/5/8  18:00-20:00 | Draw class digram |
| 2018/05/10  19:00-23:30 | 修改squence diagram |  |  | 2018/5/10  18:00-23:00 | Draw sequence Digram |
| 2018/05/11  09:30-15:00 | 修改ssd 和 sd |  |  | 2018/5/11  10:00-15:00 | Draw sequence Digram |
| Total | 37hrs | Total | 37hrs 36mins | Total | 64hrs |
| HW#6 | | | | | |
| 2018/05/19  09:30~17:00 | 完成WindowsReminder的class | 2018/5/28  14:00~17:00 | System Context Diagram refinement、  Use Case refinement (extension part)、  Domain Model refinement | 2018/5/28  14:00~17:00 | System Context Diagram refinement、  Use Case refinement (extension part)、  Domain Model refinement |
| 2018/05/20  09:30~17:00 | 完成WindowsReminder外部的batch |  |  |  |  |
| 2018/05/26  09:30~17:00 | 完成WindowsReminder外部的script |  |  |  |  |
| 2018/05/27  09:30~17:00 | 完成WindowsReminder剩餘的delete功能 |  |  |  |  |
| Total | 30hrs | Total | 3hrs | Total | 3hrs |
| HW#7 | | | | | |
| 2018/6/23  15:30~20:40 | Contract refinement、整理文件內容 | 2018/6/23  15:30~20:40 | Contract refinement、整理文件內容 | 2018/6/9  13:00~23:00 | Write generate chart feature |
| 2018/6/24  11:00~13:00  15:00~20:00 | 整理文件內容、撰寫unit test | 2018/6/24  11:00~13:00  15:00~20:00 | 整理文件內容、撰寫unit test | 2018/6/24  22:00~23:00 | Write export file feature |
| 2018/6/25  10:00~12:05  14:20~18:25 | 撰寫unit test、  Summary of System Features refinement、  Use Case Diagram refinement、  Use Case refinement、  System Sequence Diagram refinement、  Contract refinement、  Sequence Diagram refinement、  Unit Testing Code Listing refinement | 2018/6/25  10:00~12:05  14:20~18:25 | 撰寫unit test、  Summary of System Features refinement、  Use Case Diagram refinement、  Use Case refinement、  System Sequence Diagram refinement、  Contract refinement、  Sequence Diagram refinement、  Unit Testing Code Listing refinement | 2018/6/25  10:00~12:00  14:00~17:00 | Write homework7 |
| 2018/6/26  10:00~12:04  14:45~19:03 | 撰寫unit test、  製作ppt、  解決行事曆部分程式的bug | 2018/6/26  10:00~12:04  14:45~19:03 | 撰寫unit test、  製作ppt、  解決行事曆部分程式的bug | 2018/6/26  10:00~12:00  15:00~16:30 | Prepare demo ppt and refactor the code |
| 2018/6/27  10:00~12:02  14:00~16:00 | 撰寫unit test、  Line of code of classes refinement、  Source code listing refinement、  Comparison with design and implementation class refinement、  Summary of Implementation and design refinement、  製作ppt | 2018/6/27  10:00~12:02  14:00~16:00 | 撰寫unit test、  Line of code of classes refinement、  Source code listing refinement、  Comparison with design and implementation class refinement、  Summary of Implementation and design refinement、  製作ppt | 2018/6/27  10:00~12:00  14:00~16:00 | Write homework and prepare ppt |
| Total | 28hrs 44mins | Total | 28hrs 44mins | Total | 23.5hrs |