FunSheets Software

Team 2 - Iteration 2

4 March 2013

Table 1: Team

Name	ID Number
Kevin Cameron	9801448
Addison Rodomista	1967568
Dragos Dinulescu	6304826
Adrian Max McCrea	9801448
Ghazal Zamani	1971158
Karim Kaidbey	9654726
Carmelo Fragapane	6298265
Long Wang	9547967
Simone Ovilma	9112510
Nicholas Constantinidis	6330746
Asmaa Alshaibi	9738231

1 Introduction

This document is the project design document. As the name suggest this document will basically refer to the design that went behind the creating of this program. The goal of this document is to familiarise the reader with the thought process and planning that went into deciding how the program should be made. The document will introduce the overall architectural design of the FunSheets program as well as a more detailed look into the design of the subsystems, followed by some dynamic design scenario of the program.

2 Architectural Design

This section gives a high-level description of the deliverable 2 system in terms of its modules and their respective purpose and exact interfaces. You will also find the updated and detailed version of the architecture design that was presented in deliverable 1 for our FunSheet spreadsheet application.

2.1 Architecture Diagram

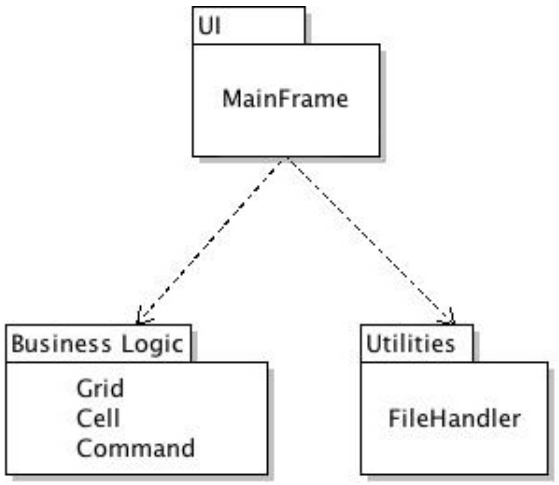


Figure 1: High-Level Structure Package Diagram

The rationale behind this design was to create a separation between the model and the view. By separating the model and view we are decoupling these objects so that changes to one can affect any number of others without requiring the changed object to know details of the others.

This iteration's design is drastically different from that in deliverable 1. In the deliverable 1 design the team focused solely on a functioning simple application that got the job done. In this deliverable we designed it with functionality, reusability, and cleanliness

in mind.

In this deliverable the system consists of three separate subsystems. These subsystems are the GUI or view, the spreadsheet logic or model, and then finally the utilities. The GUI represents what the user sees and how they interact with the system. The model is the behind the scenes operations of the system. This means handling the inputted formulas passed by the view, performing some validation on the formula, formulating the proper result, and then giving the result to the view to be displayed. The utilities includes the necessary miscellaneous functions required by the system. In this case the only current function is the loading and saving of the spreadsheets.

2.2 Subsystem Interfaces Specifications

The following subsections describe the interfaces between the subsystems. This includes the function call, the service of the function, the parameters that are passed in order to have a service fulfilled, and finally the valid ranged of values accepted by the functions.

GUI Subsystem Interface

The GUI Subsystem never gets called open for any services, it does however use the other subsystem interfaces.

Spreadsheet Logic Subsystem Interface

Function Call	Grid()
Service	Initializes the grid object.

Function Call	getSelectedRow()
Service	Returns the spreadsheet's currently selected row.
Return Description	The spreadsheet's currently selected row.

Function Call	getSelectedColumn()
Service	Returns the spreadsheet's currently selected column.
Return Description	The spreadsheet's currently selected column.

Function Call	insertValue(double value, int row, int column)
Service	Inserts the given value into the given row and column.
Parameter Description	The value to be inserted and the integer row and column that will be inserted into.
Parameter Constraints	Both the row and column should be between 0 and 10 inclusive.

Function Call	insertValue(String value, int row, int column)
Service	Inserts the given value into the given row and column.
Parameter Description	The string value to be inserted, and the integer row and column that will be inserted into.
Parameter Constraints	Both the row and column should be between 0 and 10 inclusive.

Function Call	evaluateCell(Cell cell)
Service	Takes the given cell and analyzes its contents to see if it is a primitive or a string. If it is a primitive it simply sets the cell's value to the primitive. If it is a string it evaluates the formula and sets the cell's value to the corresponding result.
Parameter Description	The cell whose contents will be evaluated.
Return Description	The double result of the evaluation.

Function Call	getRowCount()
Service	Gets the current number of rows.
Return Description	The current number of rows.

Utilities Subsystem Interface

Function Call	FileHandler(Component component)
Service	Initializes the FileHandler object.
Parameter Description	The component that the FileHandler will operate on.

Function Call	saveFile(Grid grid)
Service	Saves the given spreadsheet to the user's chosen destination.
Parameter Description	The grid in which represents the spreadsheet and its contents.

Function Call	loadFile(Grid grid)
Service	Loads the given spreadsheet to the user's chosen destination.
Parameter Description	The grid in which represents the spreadsheet and its contents.

Function Call	setSaved(Boolean saved)
Service	Signifies whether the current open spreadsheet is saved or not.
Parameter Description	True if the currently open spreadsheet is saved, false otherwise.

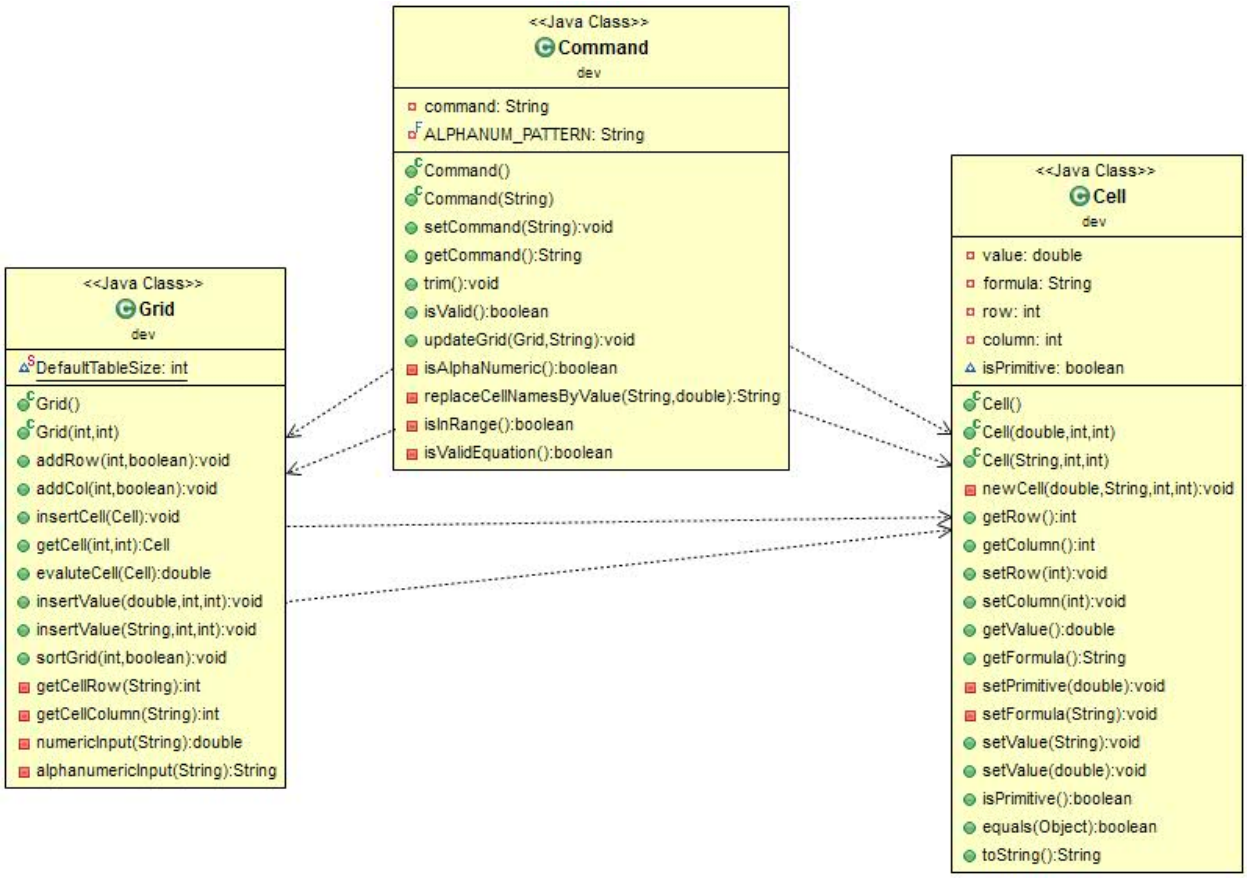
Function Call	checkSaved()
Service	Returns a boolean representing whether the current open spreadsheet is saved or not.
Return Description	True if the currently open spreadsheet is saved, false otherwise.

3 Detailed Design

3.1 Subsystem <Spreadsheet Logic>

Detailed Design Diagram

The rational behind this design was to make the subsystem this way was to simplify writing the code as well as improving it's overall comprehensiveness as many people would be using it an need to understand it. To achieve this it was split into three classes Cell, Grid and Command. Cell is the base unit that makes up Grid. Command is used to read formulas and apply them from a Cell.



Units Description

Cell class is a class that stores: a value(value), a formula(formula), a row(row), a column(column), and a primitive identifier(isPrimitive). Most of the items just listed are self-explaining but elaboration may be needed on isPrimitive; it is basically an identifier to indicate weather the cell is simply holding a value (true) or it is following a formula (false).

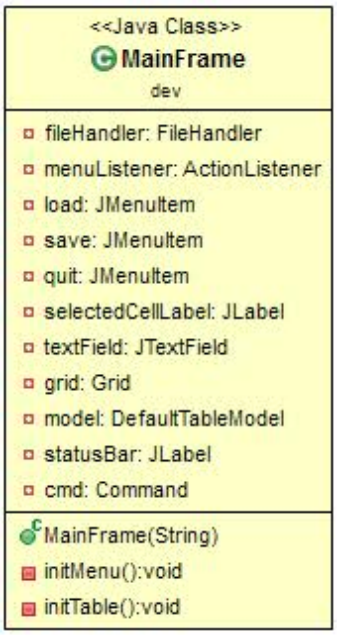
The Grid class is fairly simple it simply creates a two-dimensional array of Cell that is sized at DefaultTableSize by DefaultTableSize.

Command class is used to take formulas from non-primitive Cells and apply them to obtain a value.

3.2 Subsystem <Graphic User Interface>

Detailed Design Diagram

The rational behind designing the subsystem this way was to keep the whole graphic user interface in one class so it could be easily understood and manipulated by everyone working on it in the future.



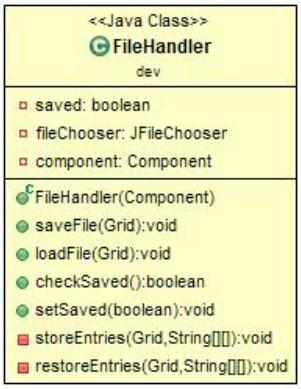
Units Description

Basically this creates a window with a menu bar that contains commands to save load and quit. Under the menu it has an input line for the a 10 by 10 grid below it. At the bottom of the window there is a status bar that is used to communicate with a user.

3.3 Subsystem <Utilities>

Detailed Design Diagram

This Subsystem was designed by keeping the function simple to program for the developer. This is achieve though the use of already existing classes from Java swing, most notably JFileChooser.



Units Description

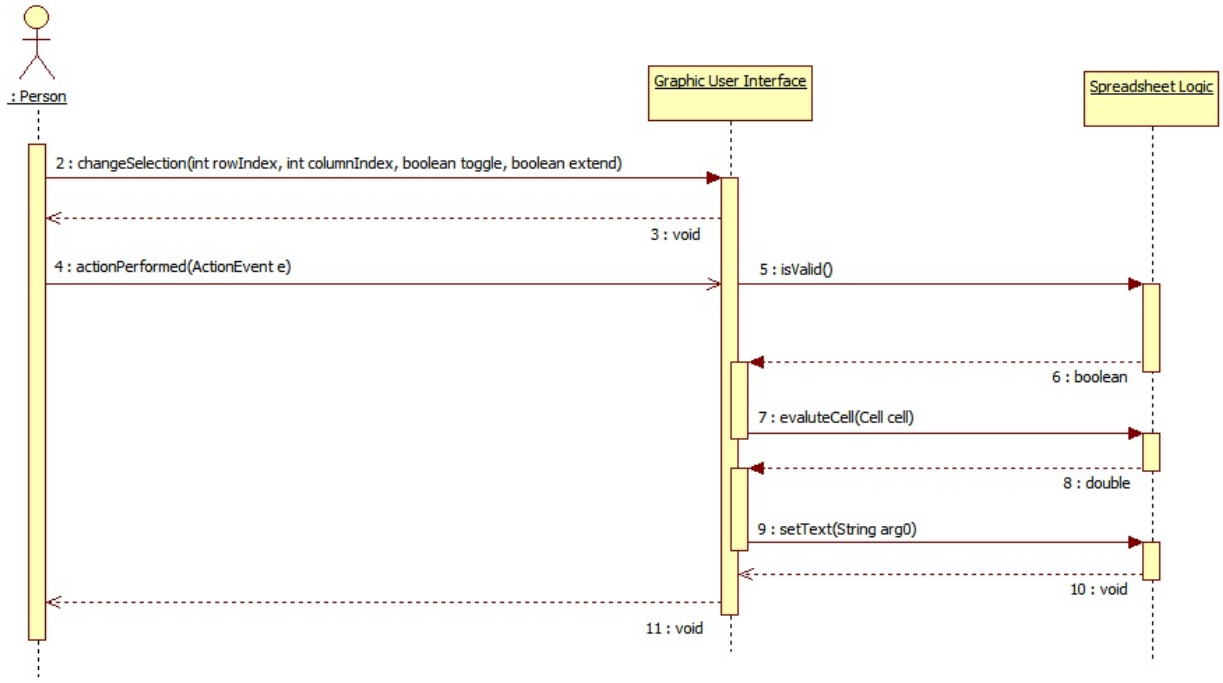
This subsection allows for a user to browse through their file system to select a location to save the currently opened grid or load an existing grid. Files are saved in .cvs format, the advantage of this is that it is a recognized format and can be opened with other programs such as excel or even a standard text editor but the disadvantage is that the currently working grid will not save the formula of cells but rather only their values.

4 Dynamic Design Scenarios

These two UML sequence diagrams show important execution scenarios of the system. These scenarios demonstrate how the various subsystems and units are interacting to achieve a system-level service.

UML Sequence Diagram 1

UML diagram showing a scenario where user selects a cell, then enters a command which is then validated and the cell contents get updated.



UML Sequence Diagram 2

UML diagram showing a scenario where user selects the "Load" menu option, then selects a file to be loaded, which is then processed and the cell contents get updated.