[Introduction 2](#_Toc443459072)

[Overview 2](#_Toc443459073)

[Cell Range 3](#_Toc443459074)

[Example Demonstrating Program Features 3](#_Toc443459075)

[Program Behavior 4](#_Toc443459076)

[Commands 4](#_Toc443459077)

[Quit 5](#_Toc443459078)

[Display Cell Content 5](#_Toc443459079)

[Set Cell 5](#_Toc443459080)

[Text Cell Type 6](#_Toc443459081)

[RealCell Type 6](#_Toc443459082)

[Clear Cell 9](#_Toc443459083)

[Clear Spreadsheet 9](#_Toc443459084)

[Display Spreadsheet 9](#_Toc443459085)

[Hint 9](file:///C:\Users\chrisdar\Documents\APCS\2015-16\Projects\Project2-TextExcel\Unit%205%20Text%20Excel\Text%20Excel%20A%20Student%20Guide.docx#_Toc443459086)

[Example 9](#_Toc443459087)

[Getting Started 12](#_Toc443459088)

[Checkpoints 13](#_Toc443459089)

[Checkpoint 1: Main command loop, Spreadsheet, SpreadsheetLocation 13](#_Toc443459090)

[Checkpoint 2: Assign and inspect TextCells, clear cells, and print sheet 14](#_Toc443459091)

[Hints 15](file:///C:\Users\chrisdar\Documents\APCS\2015-16\Projects\Project2-TextExcel\Unit%205%20Text%20Excel\Text%20Excel%20A%20Student%20Guide.docx#_Toc443459092)

[Final Submission: ValueCell, PercentCell and FormulaCell 16](#_Toc443459093)

[Hint 16](file:///C:\Users\chrisdar\Documents\APCS\2015-16\Projects\Project2-TextExcel\Unit%205%20Text%20Excel\Text%20Excel%20A%20Student%20Guide.docx#_Toc443459094)

[Extra Credit 16](#_Toc443459095)

[1. Command error handling – 5 points 16](#_Toc443459096)

[2. Command history – 5 points 17](#_Toc443459097)

[3. File Saving and Loading – 5 points 18](#_Toc443459098)

[Save Spreadsheet 18](#_Toc443459099)

[Load Spreadsheet 18](#_Toc443459100)

[Hints 19](file:///C:\Users\chrisdar\Documents\APCS\2015-16\Projects\Project2-TextExcel\Unit%205%20Text%20Excel\Text%20Excel%20A%20Student%20Guide.docx#_Toc443459101)

[File Reading and Writing 19](#_Toc443459102)

[Rubric 20](#_Toc443459103)

# Introduction

VisiCalc[[1]](#footnote-1) was the first “killer app” that made the personal computer an essential business tool. It was a spreadsheet program that showed the number crunching prowess and automation capabilities of the personal computer. It moved the computer out of the hobbyist’s home and into corporate America. The objective of this assignment is to create a lightweight version of the spreadsheet program on the console output window of Eclipse.

# Overview

A spreadsheet is a two dimensional rectangular grid of cells indexed by column letters and row numbers. A letter followed by a number identifies a cell within the spreadsheet (e.g., "D13" or "F9"). The cell designated by "C2" refers to the cell in the third column (labeled "C") that is in the 2nd row as illustrated here:

|A |B |C |D

1 | | | |

2 | | |(cell C2) |

3 | | | |

4 | | | |

Your program will repeatedly accept commands from the user (to change values stored in cells, perform operations on cells, etc.), until the user types "quit". At this point your program will end. Your program will also have a command to print the spreadsheet’s content to the console. This will show the impact of any changes caused by the user. The commands that are supported by your program are described in detail later on in this document.

This project is broken down into two parts, each described by its own document:

* Part A – That’s this document!
* Part B – Builds on top of Part A.

Each of these parts is composed of one or more Checkpoints. Each Checkpoint is a “deliverable” and has a due date associated with it.

We have defined the set of classes that you will use for this project. They are summarized in the diagram that follows. Note: only some of these classes are needed at the beginning of the project. We recommend that you wait to include a class when it is needed to satisfy a Checkpoint deliverable.



## Cell Range

Some of the commands supported by your program apply to a cell range. A cell range is a contiguous group of cells in a rectangular region of the spreadsheet. A1-B3, A1-A7 and A1-B1 are examples of valid cell ranges. The format of a cell range is:

<upper-left>-<lower-right>[[2]](#footnote-2),

where

* <upper-left> is the top, left cell of the rectangular region
* <lower-right> is the bottom, right cell of the rectangular region

Note that it is valid for a cell range to be composed only of cells within one row (A1-A7) or one column (A1-B1).

## Example Demonstrating Program Features

Your spreadsheet will be able to store three major types of cells:

* Empty cell – All of the cells in a new spreadsheet begin as empty cells
* Text cell – A cell containing string content. The content is entered by a user with surrounding double quotes. When printing the content of the spreadsheet, a text cell’s content is printed without the double quotes.
* Real cell – A cell containing decimal values. The Formula cell and the Percent cell are kinds of real cell that will also be included in the project…but more on that later.

An example (partial) spreadsheet is shown below with these non-empty cells:

* C2: the text value "hello"
* C3: the real value 2.0
* D3: the real value 4.0
* A1: the percent value 3%
* F3: the average of cell C3 and D3, represented as ( avg C3-D3 )
* Cell range B5-E7 is highlighted to illustrate a cell range.

|A |B |C |D |E |F

1 |3% | | | | |

2 | | |hello | | |

3 | | |2.0 |4.0 | |3.0

4 | | | | | |

5 | | | | | |

6 | | | | | |

7 | | | | | |

# Program Behavior

Your spreadsheet will not have a GUI (Graphical User Interface) like Microsoft Excel, but will instead be more similar to VisiCalc. All input and output will be done through the console window of Eclipse. To ensure that our output will fit well within the console window, our spreadsheet will be limited to the size 12 columns by 20 rows.

To facilitate the design and testing of your project, you will begin from a starter project that includes interfaces for you to implement and test classes that your code must pass. The code you submit for each checkpoint and the final project must work with **unmodified copies of these supporting files**. More details on those interfaces and tests will appear later in this document.

When your program starts, an empty spreadsheet will be created and displayed on the console (see the *Example* section later in this document). After this, your program will enter a loop where it waits for a command, acts based on the command, and prints either the result of the command or the updated spreadsheet depending on the specific command. The program exits when the user enters the command quit.

Each row of the spreadsheet must be numbered and each column lettered. There are **12 columns (A-L) by 20 rows**. Horizontally, each cell has a fixed length and is separated by a |. **The width of each cell is 10 characters**, not including the | on each side. If the value displayed for the cell doesn’t fit within this space, it is truncated in the display. The internal representation of the value must **not** be truncated. Truncation is only for display of the cell within the spreadsheet. When displaying the content of a single cell the entire value must be displayed. **If you do not understand this distinction, then ask!**

# Commands

The following table lists the commands that your application must support. Commands are instructions entered by the user to perform actions to manage the spreadsheet. Note that commands, cell references and cell range are case-insensitive. For example, AVG, aVg, and avg all mean the same thing. And cell references A5 and a5 refer to the same cell.

However, both case and whitespace are important in text values:

* "Hello" is different from "hello"
* "hi fi" is different from "hi fi"

The following table shows the commands that your program must support. Details on each command are described in the sections below. Note: in the examples, every element of a command is separated by a single whitespace. This is true for all commands and will be very helpful to you in your project.

|  |  |  |
| --- | --- | --- |
| Command | Example | Action |
| quit | quit | Quits the program |
| <cell> | B3 | Displays the content of the cell **as entered by the user** when the cell’s value was set. |
| <cell> = <*value*> | F7 = "semper fi"  B2 = ( avg B5-D6 )  E9 = 5  C2 = 4.5  D1 = ( 2 \* 7 / 3 )  D2 = ( C2 + 1 )  F5 = 6.2837% | Set the content of the specified cell using the provided value. The type of the cell required is inferred from the value. In the examples shown, F7 is a TextCell; B2, D1, and D2 are FormulaCell; E9 and C2 are ValueCell; F5 is a PercentCell. |
| clear <cell> | clear H13 | Makes the specified cell an EmptyCell |
| clear | clear | Clears all cells in the spreadsheet by setting each cell to an EmptyCell |
| print | print | The content of the entire spreadsheet is displayed |

## Quit

Syntax: quit

When this command is entered, the program exits.

## Display Cell Content

Syntax: <cell>

Example: B3

This command contains only a reference to a cell. When this command executes, your program displays the content for the cell **exactly as it was entered by the user**. There is one exception to this rule: when the command references an EmptyCell, an empty string surrounded by double quotes ("") is displayed.

Note: This command displays content that is generally *different* from what is displayed when the spreadsheet is displayed. See the Display Spreadsheet section below.

## Set Cell

Syntax: <cell> = <value>

Example: B3 = 5

Sets the content of the referenced cell with the value supplied. This is a complex activity that requires special attention. First, your program infers the type of cell from the value supplied. Your code will choose among the types TextCell, ValueCell, PercentCell and FormulaCell. Once instantiated, this cell instance will be initialized using the <value> supplied by the user.

The following subsections describe more detail on each cell type.

### Text Cell Type

Your program creates a TextCell instance when the assignment command specifies double-quotes around its value. For example,

A1 = "Four SCORe and seven years ago"

C1 = "4.0"[[3]](#footnote-3)

When the user **displays** a TextCell instance’s content (see the Display Cell Content section), your program prints the value exactly as it was entered by the user (including the double-quotes). For example, the command A1 causes your program to print:

"Four SCORe and seven years ago"

When the **full spreadsheet prints**, the value displayed is always truncated to fit inside the cell width.

|A |B |C |D |E |F |G |H |I |J |K |L |

1 |Four SCORe| |4.0 | | | | | | | | | |

### RealCell Type

Real cells are cells that hold real numbers (Java’s double) and participate in calculations. You will create three classes to implement the behaviors for numbers: ValueCell, PercentCell, and FormulaCell. These are subclasses of an abstract class, RealCell, that captures the behavior common numeric-based cells. The following subsections describe more detail about the subclasses of RealCell.

#### ValueCell Type

Your program creates a ValueCell instance when the assignment command specifies a simple double value. For example,

A1 = 8.42259265958979

C1 = 4

When the user **displays** a ValueCell instance’s content, your program prints the value exactly as it was entered by the user. For example, the command A1 causes your program to print:

8.42259265958979

When the **full spreadsheet prints**, the value displayed is truncated (if necessary) to fit inside the cell width. The value is **not rounded**. Note that C1 displays as "4.0" even though it was set as the value 4. This is because the double type displays this way automatically.

|A |B |C |D |E |F |G |H |I |J |K |L |

1 |8.42259265| |4.0 | | | | | | | | | |

#### PercentCell Type

Your program creates a PercentCell instance when the assignment command specifies a decimal value followed by a %. For example,

A1 = 8.92259265958979%

When the user **displays** a PercentCell instance’s content, your program prints the value exactly as it was entered by the user. For example, the command A1 causes your program to print:

8.92259265958979%

When the **full spreadsheet prints**, the value displayed is always truncated to the nearest whole % and includes the "%" character. The value is **not rounded**. For example, the A1 value from above prints out like this:

|A |B |C |D |E |F |G |H |I |J |K |L |

1 |8% | | | | | | | | | | | |

#### FormulaCell Type

Your program creates a FormulaCell instance when the assignment command specifies an expression contained in parentheses (notice that this is distinct from a TextCell because the value is not enclosed in double-quotes). Some examples:

A1 = ( 1 + B5 + 3 )

C1 = ( 4 )

Formulas are arithmetic expressions involving real constants, cell references, and the operators +, -, \*, and /. Order of operations must be strictly left-to-right (no operator precedence) for full credit.

For extra credit you may follow standard operator precedence rules (see the Extra Credit section for more information).

Note that a formula can contain one or more cell references inside of it (the example above has a reference to B5). When a referenced cell’s value changes, the change affects the result displayed in the formula’s cell.

You will notice that formula expressions allow the user to specify what are known as **circular references**. A circular reference is created when one cell’s formula refers to cells whose formula refers back to the original cell. Some simple examples of circular references:

1. A1 = ( B1 ); B1 = ( A1 )
2. A1 = ( B1 ); B1 = ( C1 ); C1 = ( A1 )

You are **not** required to deal with circular references in your code. They will not be tested.

##### Method Formula

In addition to an arithmetic expression, a formula may be a **method** formula, as in

B1 = ( AVG A2-A5 )

A method formula contains only one method (either SUM or AVG) and cannot contain arithmetic expressions.

##### Displaying FormulaCells

When the user **displays** a FormulaCell instance’s content, your program prints the value exactly as it was entered by the user. For example, the command A1 (see above examples) causes your program to print:

( 1 + B5 + 3 )

The command B1 causes your program to print:

( AVG A2-A5 )

In Part A of this project, when the **full spreadsheet prints**, formula cells will not calculate the formula value to display. You may print any value you like for the FormulaCell as long as it fits within the cell width and is school appropriate.

In Part B of this project, when the **full spreadsheet prints**, the formula cell will be evaluated with the value printed in the spreadsheet and truncated to fit inside the cell width. Consider the example above with

A1 = ( 1 + B5 + 3 ). If B5 has the value 0.0, then A1 will display 4.0 in the spreadsheet. If B5 is later changed to 1.5, then A1 will display 5.5:

|A |B |C |D |E |F |G |H |I |J |K |L |

1 |5.5 | | | | | | | | | | | |

##### Parsing Formulas

Some example formulas are shown in the table below.

|  |  |
| --- | --- |
| Input Example | Action |
| B3 = ( 4 \* 6 + 3 ) | Assigns formula to cell B3, which displays 27.0 when evaluated. |
| A9 = ( B3 \* 6 + 3 ) | Assigns formula to cell A9, which when evaluated retrieves the value of cell B3, multiplies the retrieved value by 6 and adds 3. |
| A1 = 5% A2 = ( 50 + A1 ) | A2’s getDoubleValue() will need to call A1’s  getDoubleValue() which will return 0.05. Therefore, A2’s getDoubleValue() returns 50.05 (because it is 50 + 0.05). |
| L14 = ( SUM B6-C12 ) | This formula calculates the sum of the values in cell range B6-C12. |
| C12 = ( AVG A1-A5 ) | This formula calculates the average of the values in cell range A1-A5. |
| D1 = ( C12 \* A9 ) | This formula calculates the product of two cell references. In turn, these cell references reference other cells (for example, C12 calculates the average of the values in cell range A1-A5). |

To parse[[4]](#footnote-4) a formula entered by the user, note that:

* The value must always start with a left parenthesis followed by a space
* All operators and operands in the expression are separated from each other with a space
* The cell range for a method formula does not have any spaces
* The formula ends with a space and then a right parenthesis

Although the formula is surrounded by parentheses, we will not allow parentheses inside the formula, and such forms will not be tested.

## Clear Cell

Syntax: clear <cell>

Example: clear A1

When this command is entered, the reference cell is replaced with an EmptyCell instance.

## Clear Spreadsheet

Syntax: clear

When this command is entered, all of the cells in the spreadsheet are replaced with EmptyCell instances.

## Display Spreadsheet

Syntax: print

When this command executes the entire spreadsheet is printed.

### Hint

We strongly recommend that your code calculates the value to display for a cell at the time that the print command executes. An alternate (and more problematic) solution would precalculate a cell’s value and save it for later use. Such a strategy will complicate your code when formula cell values change because it has a ripple effect across formulas that reference cells with formulas.

# Example

This section shows the progression of processing commands starting from the initial launch of your program. Lines entered by the user are shown below in **bold red**.

On start, your program prints an empty spreadsheet:

|A |B |C |D |E |F |G |H |I |J |K |L |

1 | | | | | | | | | | | | |

2 | | | | | | | | | | | | |

3 | | | | | | | | | | | | |

4 | | | | | | | | | | | | |

5 | | | | | | | | | | | | |

6 | | | | | | | | | | | | |

7 | | | | | | | | | | | | |

8 | | | | | | | | | | | | |

9 | | | | | | | | | | | | |

10 | | | | | | | | | | | | |

11 | | | | | | | | | | | | |

12 | | | | | | | | | | | | |

13 | | | | | | | | | | | | |

14 | | | | | | | | | | | | |

15 | | | | | | | | | | | | |

16 | | | | | | | | | | | | |

17 | | | | | | | | | | | | |

18 | | | | | | | | | | | | |

19 | | | | | | | | | | | | |

20 | | | | | | | | | | | | |

**A1 = "Hello"**

**print**

|A |B |C |D |E |F |G |H |I |J |K |L |

1 |Hello | | | | | | | | | | | |

2 | | | | | | | | | | | | |

3 | | | | | | | | | | | | |

4 | | | | | | | | | | | | |

5 | | | | | | | | | | | | |

6 | | | | | | | | | | | | |

7 | | | | | | | | | | | | |

8 | | | | | | | | | | | | |

9 | | | | | | | | | | | | |

10 | | | | | | | | | | | | |

11 | | | | | | | | | | | | |

12 | | | | | | | | | | | | |

13 | | | | | | | | | | | | |

14 | | | | | | | | | | | | |

15 | | | | | | | | | | | | |

16 | | | | | | | | | | | | |

17 | | | | | | | | | | | | |

18 | | | | | | | | | | | | |

19 | | | | | | | | | | | | |

20 | | | | | | | | | | | | |

**A1**

"Hello"

**B3 = 17**

**print**

|A |B |C |D |E |F |G |H |I |J |K |L |

1 |Hello | | | | | | | | | | | |

2 | | | | | | | | | | | | |

3 | |17.0 | | | | | | | | | | |

4 | | | | | | | | | | | | |

5 | | | | | | | | | | | | |

6 | | | | | | | | | | | | |

7 | | | | | | | | | | | | |

8 | | | | | | | | | | | | |

9 | | | | | | | | | | | | |

10 | | | | | | | | | | | | |

11 | | | | | | | | | | | | |

12 | | | | | | | | | | | | |

13 | | | | | | | | | | | | |

14 | | | | | | | | | | | | |

15 | | | | | | | | | | | | |

16 | | | | | | | | | | | | |

17 | | | | | | | | | | | | |

18 | | | | | | | | | | | | |

19 | | | | | | | | | | | | |

20 | | | | | | | | | | | | |

**B3**

17

**A3 = "this is a really long string"**

**print**

|A |B |C |D |E |F |G |H |I |J |K |L |

1 |Hello | | | | | | | | | | | |

2 | | | | | | | | | | | | |

3 |this is a |17.0 | | | | | | | | | | |

4 | | | | | | | | | | | | |

5 | | | | | | | | | | | | |

6 | | | | | | | | | | | | |

7 | | | | | | | | | | | | |

8 | | | | | | | | | | | | |

9 | | | | | | | | | | | | |

10 | | | | | | | | | | | | |

11 | | | | | | | | | | | | |

12 | | | | | | | | | | | | |

13 | | | | | | | | | | | | |

14 | | | | | | | | | | | | |

15 | | | | | | | | | | | | |

16 | | | | | | | | | | | | |

17 | | | | | | | | | | | | |

18 | | | | | | | | | | | | |

19 | | | | | | | | | | | | |

20 | | | | | | | | | | | | |

**A3**

"this is a really long string"

**C3 = 12**

**C4 = 9**

**B4 = 103**

**print**

|A |B |C |D |E |F |G |H |I |J |K |L |

1 |Hello | | | | | | | | | | | |

2 | | | | | | | | | | | | |

3 |this is a |17.0 |12.0 | | | | | | | | | |

4 | |103.0 |9.0 | | | | | | | | | |

5 | | | | | | | | | | | | |

6 | | | | | | | | | | | | |

7 | | | | | | | | | | | | |

8 | | | | | | | | | | | | |

9 | | | | | | | | | | | | |

10 | | | | | | | | | | | | |

11 | | | | | | | | | | | | |

12 | | | | | | | | | | | | |

13 | | | | | | | | | | | | |

14 | | | | | | | | | | | | |

15 | | | | | | | | | | | | |

16 | | | | | | | | | | | | |

17 | | | | | | | | | | | | |

18 | | | | | | | | | | | | |

19 | | | | | | | | | | | | |

20 | | | | | | | | | | | | |

**D4 = ( aVg b3-C4 )**

**D4**

( aVg b3-C4 )

**print**

|A |B |C |D |E |F |G |H |I |J |K |L |

1 |Hello | | | | | | | | | | | |

2 | | | | | | | | | | | | |

3 |this is a |17.0 |12.0 | | | | | | | | | |

4 | |103.0 |9.0 |35.25 | | | | | | | | |

5 | | | | | | | | | | | | |

6 | | | | | | | | | | | | |

7 | | | | | | | | | | | | |

8 | | | | | | | | | | | | |

9 | | | | | | | | | | | | |

10 | | | | | | | | | | | | |

11 | | | | | | | | | | | | |

12 | | | | | | | | | | | | |

13 | | | | | | | | | | | | |

14 | | | | | | | | | | | | |

15 | | | | | | | | | | | | |

16 | | | | | | | | | | | | |

17 | | | | | | | | | | | | |

18 | | | | | | | | | | | | |

19 | | | | | | | | | | | | |

20 | | | | | | | | | | | | |

**D3 = 0**

**F5 = ( sum B3-d4 )**

**F5**

( Sum B3-d4 )

**print**

|A |B |C |D |E |F |G |H |I |J |K |L |

1 |Hello | | | | | | | | | | | |

2 | | | | | | | | | | | | |

3 |this is a |17.0 |12.0 |0.0 | | | | | | | | |

4 | |103.0 |9.0 |35.25 | | | | | | | | |

5 | | | | | |176.25 | | | | | | |

6 | | | | | | | | | | | | |

7 | | | | | | | | | | | | |

8 | | | | | | | | | | | | |

9 | | | | | | | | | | | | |

10 | | | | | | | | | | | | |

11 | | | | | | | | | | | | |

12 | | | | | | | | | | | | |

13 | | | | | | | | | | | | |

14 | | | | | | | | | | | | |

15 | | | | | | | | | | | | |

16 | | | | | | | | | | | | |

17 | | | | | | | | | | | | |

18 | | | | | | | | | | | | |

19 | | | | | | | | | | | | |

20 | | | | | | | | | | | | |

**clear A3**

**print**

|A |B |C |D |E |F |G |H |I |J |K |L |

1 |Hello | | | | | | | | | | | |

2 | | | | | | | | | | | | |

3 | |17.0 |12.0 |0.0 | | | | | | | | |

4 | |103.0 |9.0 |35.25 | | | | | | | | |

5 | | | | | |176.25 | | | | | | |

6 | | | | | | | | | | | | |

7 | | | | | | | | | | | | |

8 | | | | | | | | | | | | |

9 | | | | | | | | | | | | |

10 | | | | | | | | | | | | |

11 | | | | | | | | | | | | |

12 | | | | | | | | | | | | |

13 | | | | | | | | | | | | |

14 | | | | | | | | | | | | |

15 | | | | | | | | | | | | |

16 | | | | | | | | | | | | |

17 | | | | | | | | | | | | |

18 | | | | | | | | | | | | |

19 | | | | | | | | | | | | |

20 | | | | | | | | | | | | |

**C3 = 1000**

**print**

|A |B |C |D |E |F |G |H |I |J |K |L |

1 |Hello | | | | | | | | | | | |

2 | | | | | | | | | | | | |

3 | |17.0 |1000.0 |0.0 | | | | | | | | |

4 | |103.0 |9.0 |282.25 | | | | | | | | |

5 | | | | | |1411.25 | | | | | | |

6 | | | | | | | | | | | | |

7 | | | | | | | | | | | | |

8 | | | | | | | | | | | | |

9 | | | | | | | | | | | | |

10 | | | | | | | | | | | | |

11 | | | | | | | | | | | | |

12 | | | | | | | | | | | | |

13 | | | | | | | | | | | | |

14 | | | | | | | | | | | | |

15 | | | | | | | | | | | | |

16 | | | | | | | | | | | | |

17 | | | | | | | | | | | | |

18 | | | | | | | | | | | | |

19 | | | | | | | | | | | | |

20 | | | | | | | | | | | | |

**clEAR**

**prInt**

|A |B |C |D |E |F |G |H |I |J |K |L |

1 | | | | | | | | | | | | |

2 | | | | | | | | | | | | |

3 | | | | | | | | | | | | |

4 | | | | | | | | | | | | |

5 | | | | | | | | | | | | |

6 | | | | | | | | | | | | |

7 | | | | | | | | | | | | |

8 | | | | | | | | | | | | |

9 | | | | | | | | | | | | |

10 | | | | | | | | | | | | |

11 | | | | | | | | | | | | |

12 | | | | | | | | | | | | |

13 | | | | | | | | | | | | |

14 | | | | | | | | | | | | |

15 | | | | | | | | | | | | |

16 | | | | | | | | | | | | |

17 | | | | | | | | | | | | |

18 | | | | | | | | | | | | |

19 | | | | | | | | | | | | |

20 | | | | | | | | | | | | |

**QuiT**

# Getting Started

You will create your TextExcel project using a procedure similar to what you used for your Calculator project. We will provide you with a starter-project in class that you will import into Eclipse. This project contains the interfaces you must implement, and the test cases that must pass. The interfaces include:

* Grid – to be implemented by your Spreadsheet class. This defines the behavior of a spreadsheet as a set of cells that can be accessed by specifying row and column.
* Cell – to be implemented by all of your cell classes (EmptyCell, TextCell, RealCell, ValueCell, PercentCell, FormulaCell). In some cases, you will use inheritance between your cell classes. When this is the case, only the superclass specifies the Cell interface. If you are confused by this, ask us! The Cell interface defines the basic capabilities of a cell in the spreadsheet.
* Location – to be implemented by your SpreadsheetLocation class. This specifies how your program identifes a specific cell within the spreadsheet: by row and column.

Open these files in Eclipse, look them over, and read the comments. They give a partial recipe on how you will design your classes, and also state what our tests require. ***Do not change these interfaces.***

The tests are stored in TestsALL.java.

If you have any questions about how your program should behave, look at TestsALL.java first, or try running the tests and see what Eclipse prints as the “expected” output if they fail. If your checkpoints or final submissions are incompatible with the tests, they will not be graded and you will not get points for that work. Ensure the appropriate tests run **and pass** before you submit.

# Checkpoints

Note that the required code for checkpoints is cumulative. When you submit each checkpoint, the submitted program must pass the details in the current checkpoint as well as all objectives from previous checkpoints. When a test that previously passed is broken because of new work, this is called a *regression*. You will lose points for regressions.

To facilitate testing when your project is graded, include all of your classes and the code you imported into your project (including the interfaces and tests) as part of each checkpoint submission.

## Checkpoint 1: Main command loop, Spreadsheet, SpreadsheetLocation

In this checkpoint, the code you submit must include four Java source files that you write:

* **Spreadsheet.java:** A class that implements the provided Grid interface
  + You must correctly implement the getRows() and getCols() methods of the Grid interface.
  + None of the other Grid methods will be tested in this checkpoint, so you may include any dummy implementations you like as long as they compile and don’t crash the program.
  + Your Spreadsheet constructor must initialize a 2D array of cells with all elements containing EmptyCell objects.
* **SpreadsheetLocation.java**: A class that fully implements the Location interface, and contains a constructor taking a single String parameter (e.g., "D20"). Remember, all commands are case-insensitive so d20 must also work correctly.
* **TextExcel.java:** A class with a main method that constructs a Spreadsheet object instance and has the command loop to
  + Read commands;
  + Call the spreadsheet’s processCommand method to process each line of user input;
  + Print the String returned from processCommand;
  + Repeats command processing until "quit" is entered by the user.

You do not need a correct implementation of any command other than quit, as long as the program compiles and does not crash.

* **EmptyCell.java:** A class that implements the provided Cell interface, and represents an empty cell.

*Testing: All of the tests provided in A\_Checkpoint1 must pass. You must also ensure that your program works as specified when run interactively (test it by running it yourself).*

## Checkpoint 2: Assign and inspect TextCells, clear cells, and print sheet

For this checkpoint, you must do the following (see the hints shown below for advice):

* Create a TextCell class that implements the Cell interface. A TextCell stores string values.
* Implement the getGridText() method on your Spreadsheet class, to return a String containing the *entire sheet grid* in the form described in the spec.
* Implement the processCommand() method on your Spreadsheet class. This method returns a String to the caller. The caller displays the returned content to the console. Implement enough of the processCommand() method to handle the following four commands (remember the quit command must continue to work:
  + **Cell display** (e.g., the command A1). This returns the value at that cell; see above for examples.
  + **Assignment to string values** (e.g., A1 = "Hello"). Assign the cell location the correct (TextCell) type with an object instance that has the supplied String value. The processCommand() method returns a String. In this case, return an empty string since there is no content to display.
  + **Clearing the entire sheet** (clear). Clear all cells of the Spreadsheet instance, replacing all cells with EmptyCell instances. Return an empty string.
  + **Clearing a particular cell** (e.g., clear A1). Clear the specified cell location by replacing that cell with an EmptyCell object instance. Return an empty string.
  + **Print the entire spreadsheet** (print). Return a String containing the entire content of the spreadsheet. The caller displays this content.
* Implement the getCell() method on the Spreadsheet class. This method accepts a Location, and returns the Cell object instance at that location. For this checkpoint, the Cell instance returned must be either an EmptyCell or a TextCell, depending on whether that cell was assigned a string value or is empty. For future checkpoints, other types of Cells will be returned.

*Testing: All of the tests provided in A\_Checkpoint1 and A\_Checkpoint2 must pass. You must also ensure that your program works as specified when run interactively.*

### Hints

* To parse commands, you may find the split() method useful (see the String class). When you call split() on a string, and give it a delimiter string, such as " " (i.e., a String with a space), it will return an array of Strings split up by the delimiter you passed in. For example:

String str = "Apple Banana Orange";

String[] arr = s.split(" ");

will set arr to an array consisting of three strings: "Apple", "Banana" and "Orange". There is also a method overload that takes a second parameter to set a limit on the number of strings returned. For example, s.split(" ", 2); returns an array consisting of two strings: "Apple" and "Banana Orange".

* Your spreadsheet output from getGridText() must match *exactly* what the test expects, including spaces. An easy way to see what is expected is to run this code in your main() method *temporarily*

TestsALL.Helper th = **new** TestsALL.Helper();

System.*out*.println(th.getText());

and then output your spreadsheet’s text immediately after so you can see how they line up.

* + *Note! Be sure to remove the above (and any other code you add to help yourself debug) before you submit to us! You must write your own grid formatting code.*
* When parsing cell identifiers (such as B3) you will need to convert the column letter (B) into a column index of the array of cells in your Spreadsheet class. The following code is a useful way to do this. If you expect the column letter to appear at index i in the String str, then the column index is:

int col = Character.*toUpperCase*(str.charAt(i)) - 'A';

This code sets col to 0 for the letter A, 1 for B, etc. This works because the char data type stores characters as numbers, and the capital letters have sequential values when viewed in alphabetical order.

* For cell display commands (such as A1), your processCommand() method should use the fullCellText() method on the Cell interface.
* The getGridText() method should use the abbreviatedCellText() method on the Cell interface.
* Throughout this project we will add more and more functionality to the processCommand() method, and the implementation can grow very long. Break functionality up into helper methods as appropriate for readability.

## Final Submission: ValueCell, PercentCell and FormulaCell

For this checkpoint, you must do the following:

* Implement a RealCell super class that is **abstract** and implements the Cell interface. In addition, create three subclasses that extend RealCell: ValueCell, PercentCell, and FormulaCell.   
    
  The RealCell super class stores the string representing what the user entered. This is the string your code prints when displaying the content of one cell (see the RealCell Type section). The RealCell class needs an abstract getDoubleValue() method that returns the calculated value of the cell. Subclasses of RealCell implement this method because each subclass determines the value in a different way.

For this checkpoint,

* ValueCell and PercentCell must implement methods getDoubleValue() and abbreviatedCellText() correctly.
* FormulaCell can return anything that is 10 characters long. **This code does not need to parse formulas (that will come in Part B)**.
* The fullCellText() method must return the content as entered by the user for all supported cell types.
* Continue to add functionality to the processCommand() function, so that it handles
  + Percent assignment (e.g. A1 = 5.2%)
  + Real value assignment (e.g. A1 = 5.2, or A1 = ( A2 + A3 \* 4 ), or A1 = ( sum A1-D4 )).

*Testing: All of the tests provided in A\_Checkpoint1, A\_Checkpoint2 and A\_Checkpoint3 must pass. You must also ensure that your program works as specified when run interactively.*

### Hint

Although PercentCell must extend RealCell, there are two ways you could do that:

1. PercentCell directly extends RealCell
2. PercentCell extends ValueCell (which in turn extends RealCell).

Choose the way that you are most comfortable with.

# Extra Credit

There are three opportunities for extra credit in Part A. You can attempt any or all of them. ***Complete the non-extra credit first! Save a copy of all of your code without any extra credit work. Then you have something to fall back on if you ruin the required portion of you code!***

## 1. Command error handling – 5 points

If the user enters an invalid command, an error (e.g., "ERROR: Invalid command.") should be displayed. No other action should be taken, and the program should wait for the next command. Letting the program crash does not qualify as error handling. ☺

There are several cases you must handle:

* Entering a command that does not exist.
* Entering a command that does exist, but not following the required format.
* Entering a cell reference that is outside of the spreadsheet range.

The FormulaCell will be tested, but parsing of formula content will not (that will be extra credit in Part B).

## 2. Command history – 5 points

Maintain a history of the commands the user entered. If the user types history enable n, where n is a positive number, your program keeps track of the most recent n commands the user entered. When the user types history display, your program displays the most recent n commands the user entered, one per line, in descending order (most recent first, followed by next most recent, etc). If the user entered fewer than n commands, history display shows all of the commands that the user entered so far.

If the user types history clear m, where m is a positive number, it should clear the oldest m commands from the history. (If there are fewer than m commands in the history, it should clear all of them.)

Finally, if the user types history disable, the program should clear the entire history and no longer keep track of commands the user entered (unless the user enters the history enable n command again). If command history is not being tracked, the command history display returns a message such as "command history is not enabled". The command history should not record history commands themselves (history display, etc).

For example, if the user first enters the following commands

history enable 4  
A1 = 1  
A2 = 2  
A3 = 3  
B1 = 4  
clear B1  
B2 = 5  
history display

Your program displays this history:

B2 = 5  
clear B1  
B1 = 4  
A3 = 3

Then, if the user enters history clear 2 and then types history display again, your program displays this history:

B2 = 5  
clear B1

## 3. File Saving and Loading – 5 points

Sometimes we want to pause work on a spreadsheet and return to work on it later. At times like this, we want to save a copy of the content. Of course, this saved copy is of little use unless we also have a way to read (also called load) a saved copy into our program. By implementing both of these operations, we can restore our program from our last save point. This is very handy if our computer restarts or someone closes our application before we complete our work with the spreadsheet.

|  |  |  |
| --- | --- | --- |
| Command | Example | Action |
| save <filename> | save data.csv | Saves the content of the spreadsheet to the specified file |
| open <filename> | open data.csv | Overwrites the current spreadsheet with the content of the specified file. **The entire table is displayed after data import.** |

### Save Spreadsheet

When this command executes the entire spreadsheet is saved to the specified file <filename>.

You will use the information below (see the File Reading and Writing section) to create a new text file for saving. You will write the content of your spreadsheet in “comma-separated value” format, also known as “CSV”. The process of saving data like this is sometimes referred to as *serialization* (you are serializing your spreadsheet to a file).

Each line of the text file contains the content of one cell of your spreadsheet, and will be of the format:

CellReference,CellType,FullCellText

For example, following is the content of a valid csv file:

A20,ValueCell,2.2  
B20,PercentCell,2.4%  
C20,FormulaCell,( A20 + B20 )  
D20,TextCell,"howdy"

**You must adhere to the above format exactly.**

Notice the following:

* This content does not contain any EmptyCell content. Your code will assume that any cell not referenced in the CSV file is an EmptyCell. This keeps the file size small for spreadsheets that are sparsely populated.
* The format of the cell’s data in the file is identical to what the Cell instance returns from its fullCellText() method.
* There are no spaces between the commas.

### Load Spreadsheet

When this command executes the entire spreadsheet is replaced with the content of the specified file **and** the spreadsheet’s content is displayed. You will use the information below (see the File Reading and Writing section) to open an existing file for reading. The process of loading data like this is sometimes referred to as *deserialization* (you are deserializing your spreadsheet from a file).

The format of the file content is identical to the above format used to save the file. Write code to use a Scanner instance to read each line from the file, and create the appropriate Cell object instance (with the correct type and data), and store that Cell instance into the Spreadsheet.

### Hints

* Implement save first. Then save a spreadsheet to a file and open the file in notepad. Make sure that the format is correct (see the Saving section above). Save a spreadsheet with all of the types of cells when you do this.
* If a filename is specified without a path (e.g., save myData.csv), it will be saved (or opened) in your Project directory inside your Eclipse workspace. At least, that’s how it works in Windows; Mac users…you’re on your own.
* To specify a full path for the filename include drive letter and backslashes. For example, save c:\stuff\myData.csv.

### File Reading and Writing

To write to a file, use java.io.PrintWriter. Here is some sample code to show how to use it[[5]](#footnote-5):

String fileName = "c:\\stuff\\myData.csv";

PrintWriter file;

try

{

file = new PrintWriter(fileName);

}

catch (FileNotFoundException e)

{

return "Could not find file '" + fileName + "'";

}

file.write("This is some text to write to the file");

file.close();

To read from a file, use java.io.File. Here is some sample code to show how to use it:

String fileName = "c:\\stuff\\myData.csv";

File myFile = new File(fileName);

Scanner fileReader;

try

{

fileReader = new Scanner(myFile);

}

catch(FileNotFoundException e)

{

return "Could not find file '" + fileName + "'";

}

// From here, use the Scanner instance just like you

// have in other assignments.

// ...

fileReader.close();

# Rubric

The following table shows how much each part of the project is worth.

|  |  |
| --- | --- |
| REQUIREMENT | POINTS |
| Main Loop | 10 |
| Processing Commands | 35 |
| Other Spreadsheet Methods | 20 |
| Spreadsheet Location | 10 |
| Cell Types | 35 |
| Checkpoint 1 | 11 |
| Checkpoint 2 | 12 |
| Checkpoint 3 | 12 |
| TOTAL | **145 points** |
| *Extra credit: Command Error Handling* | 5 |
| *Extra credit: Command History* | 5 |

1. Available: http://www.bricklin.com/history/vcexecutable.htm [↑](#footnote-ref-1)
2. The notation <x> indicates a placeholder of information. It is not literal text. [↑](#footnote-ref-2)
3. This is a tricky example. Even though the content is a number, it is treated like a string because the number is surrounded by "". [↑](#footnote-ref-3)
4. parse - to resolve into component parts. This is the process of extracting pieces such as operators and operands from the formula. [↑](#footnote-ref-4)
5. We have not discussed the try/catch block. Use this to neatly handle the case where the file requested does not exist. [↑](#footnote-ref-5)