Project Description

Spreadsheet systems allow several sheets to be stored in a single spreadsheet workbook. For example, in the Excel system when you create a new workbook it has, by default, three sheets.

The sheets can be named or renamed but the name must be at least one character long. The default sheets are initially named Sheet1, Sheet2 and Sheet3 but these can be changed by the user at any.

Sheet names can include the characters A to Z, upper and lower case; the digits 0 to 9 and space. Sheet names must be unique (i.e. duplicates are NOT allowed). Sheet names should be treated as case-insensitive. For example, **INCOME** and **income** and **Income** should be treated as the same name.

Additional sheets can be inserted into the workbook and a workbook can have up to 256 sheets. When a sheet is inserted into the workbook it is given the default name SheetN (where is N is the next value in the monotonically increasing integer sequence which starts at 4).

Sheets can also be deleted from the workbook but there must always be at least one sheet in the workbook.

We want to develop a Java class (or classes, if necessary) to allow the sheet names to be managed as a list of names. Each entry in the list will record a single sheet name. The following operations should be provided

- **add** – adds a new entry into the sheet name list, unless it already contains 256 entries.

  ```
  public boolean add()
  ```

  The name of the new entry is `SheetN`, where N is the next available number in the sequence. The new name is added to the end of the name list (i.e. new sheets are added at the end of the spreadsheet workbook).

  Note N is not necessarily the same as the number of sheets in the list. For example, into a new workbook (i.e. with three sheets, Sheet1, Sheet2 and Sheet3) we could add a new sheet (i.e. Sheet4) and then delete the original three. The next add operation would use the name Sheet5 even though there would only be two sheets in the workbook. *EndNote*

  If the list contained less than 256 sheets and a new sheet was successfully added then the method returns true. Otherwise it returns false.

- **remove** – removes a sheet name from the list.

  ```
  public int remove(String sheetName)
  ```

  The `remove` operation is passed the name of the sheet to be removed and searches the list for the name. If the name is not found the remove operation does nothing (i.e. the list is unaffected). If the name is found it should be removed from the list, unless it is the only name in the list. The spreadsheet workbook must always have at least one sheet so the name list must always have at least one name entry.

  If the `remove` operation is successful the method returns the index position of the sheet removed. Otherwise it returns -1.

- **remove** – an overloaded version of remove that uses an index position instead of a name.

  public String remove(int index)

  The `remove` operation is passed the `index` position of a sheet to be removed. If the index value is out of range (i.e. not in the range 1 >= index <= list length) the remove operation does nothing (i.e. the list is unaffected). If the `index` is valid then the sheet name at the index position should be removed from the list, unless it is the only name in the list. The spreadsheet workbook must always have at least one sheet so the name list must always have at least one name entry.

  If the `remove` operation is successful the method returns the name of the sheet removed. Otherwise it returns a null/empty string.

- **move** – allows a sheet name to be moved from one position in the list to another.

  public int move(String from, String to, boolean before)

  `move` is passed two sheet names. The sheet should only be moved if BOTH sheet names are in the list and they are not the same (i.e. it is not the same sheet name twice). If any one or both of the names are not found then the move operation does nothing (i.e. the list is unaffected).

  The `from` name is the sheet to be moved. The `to` name is the sheet before or after which the from sheet should be moved to. If the third parameter, `before`, is true the sheet should be moved to before the `to` sheet, otherwise it should be moved to after the `to` sheet.

  If the `move` operation is successful the method returns the index of the position the sheet was moved **to**. Otherwise it returns -1.

- **move** – an overloaded version of the move using indices instead of names

  public String move(int from, int to, boolean before)

  `move` is passed two sheet indices. The `move` should only be performed if BOTH sheet indices exist (i.e. they are in the range 1 >= index <= list length) and they are not equal (i.e. it is not the same sheet index twice). If any one or both of the indices do not exist the move operation should be ignored (i.e. the list is unaffected).

  The `from` index is the index position of the sheet to be moved. The `to` index is the index position of the sheet before or after which the first sheet should be moved to. If the third parameter, `before`, is true the sheet should be moved to before the `to` sheet, otherwise it should be moved to after the `to` sheet.

  If the `move` operation is successful the method returns the name of the `from` sheet (i.e. the sheet that was moved). Otherwise it returns a null/empty string.

- **moveToEnd** – allows a sheet to be moved from its current position to the end of the list

  public String moveToEnd(int from)

  `moveToEnd` is passed the index position of a sheet. The `moveToEnd` should only be performed if the sheet index exists (i.e. is in the range 1 >= index <= list length). If it does not exist the `moveToEnd` operation does nothing (i.e. the list is unaffected).

The `from` index is the position of the sheet to be moved. The sheet at this position should be moved to the end of the list (i.e. it should become the last entry in the list or, if you prefer, it should be moved to after the current last entry in the list).

If the `moveToEnd` operation is successful the method returns the name the sheet that was moved. Otherwise it returns a null/empty string.

- **moveToEnd** – an overloaded version of the moveToEnd using an index instead of a name

```
public int moveToEnd(String from)
```

`moveToEnd` is passed the name of a sheet. The `moveToEnd` should only be performed if the sheet name exists (i.e. it is one of the names in the list). If it does not exist the `moveToEnd` operation does nothing (i.e. the list is unaffected).

The `from` string contains the name of the sheet to be moved. The sheet should be moved to the end of the list (i.e. it should become the last entry in the list or, if you prefer, it should be moved to after the current last entry in the list).

If the `moveToEnd` operation is successful the method returns the index position of the sheet that was moved. Otherwise it returns -1.

- **rename** – changes the name of an existing sheet

```
public int rename(String currentName, String newName)
```

`rename` is passed two sheet names. Renaming should only be performed if the `currentName` sheet name is in the list and the `newName` sheet name isn't. Otherwise rename does nothing (i.e. the list is unaffected).

If the `currentName` is successfully changed to the `newName` then the method returns the index position of the sheet renamed. Otherwise it returns -1.

- **index** – returns the index position of a name in the list

```
public int index(String sheetName)
```

`index` is passed the name of a sheet. It searches the list for the sheet name. If it is found index returns the position of the name in the list. If it is not found index returns -1.

- **sheetName** – returns the name of the sheet at the specified index position.

```
public String sheetName(int index)
```

`sheetName` is passed an index position. If the index position exists (i.e. is in the range 1 >= index <= list length) the method returns the name of the sheet at that position. Otherwise it returns a null/empty string.

- **display** – displays the names in the list on the screen.

```
public void Display()
```

- **length** - returns an integer value representing the number of items in the list.

```
public int length()
```

Forming Teams

You have the option of undertaking the project alone or as part of a team. If you wish to form a team please email me by **4pm Thursday 3 November 2016 (GMT)** with the names and ID numbers of the team members. Teams are restricted to a maximum of four members. All team members are awarded the same score for the project. **Management of the team is entirely a matter for the team members**.

Submission Requirements

Your submitted solution should include the source files (.java) for any classes developed by you, as well as a 'driver' program (.java) that demonstrates that the class(es) support all of the operations required by the specification.

In **EACH AND EVERY** one of the .java files you submit you should include prominent comments at the beginning of the code that contain the ID NUMBER(s) and NAME(s) of (all of) the author(s).

NOTE: **ONLY .java files and any text files required should be submitted. No exe's, tar's or any other file type should be submitted. It is YOUR responsibility to ensure that the files submitted are the correct versions. It should NOT be necessary to install additional software or use any specific development environment to test your submission.**

The author(s) of a submission may be invited to attend a meeting to explain, justify or describe aspects of the solution.

Submissions must be received on or before

4pm Wednesday 23 November 2016 (GMT)

This project accounts for 25% of the overall marks available for the module (or 30% if you have reclaimed your Mid-Term marks).