Tarea 2 Enunciado – Hoja de Cálculo

Descripción del problema

Las hojas de cálculo (p. ej., Excel, LibreOffice Calc o Google Sheets) son de las herramientas más utilizadas para llevar la contabilidad de muchos negocios pequeños (y algunos no tan pequeños). Debido a problemas que surgieron a causa los métodos de redonde, la empresa Acme S.A. contrató a su empresa para desarrollar una pequeña extensión al software de hoja de cálculo que utilizan.

La extensión solicitada debe extender las funcionalidades de la hoja de cálculo y permitir el uso de fracciones y operaciones con fracciones. Según los encargados de la contabilidad de la empresa, las fracciones almacenarían mejor los datos de sus bitácoras financieras que su equivalente en punto flotante.

Como parte del proyecto, su empresa debe presentar un avance de la extensión con las operaciones básicas solicitadas antes de proceder con la integración de las interfaces gráficas. El avance que debe de pasar todos los casos de prueba suministrados por el equipo financiero de la empresa de Acme S.A.

Las operaciones solicitadas para la extensión son las siguientes:

1. Sumar y multiplicar filas y columnas desde una celda inicial hasta una celda final.
2. Obtener el valor máximo y mínimo de una fila o columna desde una celda inicial hasta una celda final.
3. Obtener el promedio y la mediana de una fila o columna desde una celda inicial hasta una celda final.
4. Crear subconjuntos formados por celdas de la hoja de cálculo.
5. Sumar y multiplicar subconjuntos.

El avance deberá leer los datos de prueba desde la entrada estándar como se muestran en la siguiente tabla.

Tabla 1: Entrada estándar.

```
4 8

01/02, 02/02, 03/02, 04/02, 05/02, 06/02, 07/02, 00/01
01/05, 02/05, 03/05, 04/05, 05/05, 06/05, 07/05, 00/01
01/07, 02/07, 03/07, 04/07, 05/07, 06/07, 07/07, 00/01
00/01, 00/01, 00/01, 00/01, 00/01, 00/01, 00/01, 00/01

>=CEL(H1)
>=SUMA(B1:G1)
>=CEL(H2)
>=PROMEDIO(A2:G2)
>=CEL(H3)
>=MEDIANA(A3:G3)
>=CEL(B4)
>=MULT(B1:B3)
>=CEL(G4)
>=CONJUNTO(conj1,A1,B2,C3,C4)
>=SUMA(conj1)
>=IMPRIMIR()
```

Además de los datos que debe cargar en su hoja de cálculo, su programa debe leer una serie de instrucciones. Las instrucciones indican la posición donde se debe guardar el resultado y la operación que se debe realizar.

La salida esperada debe mostrarse en la salida estándar. Esta debe mostrar la hoja de cálculo actualizada después de haber efectuado las instrucciones solicitadas.

Tabla 2: Salida estándar.

```
   |     A      B      C      D      E      F      G      H
---+------ ------ ------ ------ ------ ------ ------ ------
  1|  1/2    1/1    3/2    2/1    5/2    3/1    7/2   27/2
  2|  1/5    2/5    3/5    4/5    1/1    6/5    7/5    4/5
  3|  1/7    2/7    3/7    4/7    5/7    6/7    1/1    4/7
  4|  0/1   4/35    0/1    0/1    0/1    0/1    0/1  93/70
```

La siguiente tabla muestra la lista de comandos válidos.

Tabla 3: Lista de comandos.

Comando	**Descripción**
`=CEL(celda)`	Se posiciona en una celda.
`=CONJUNTO(nombre, celda1,…,celda_n)`	Crea un conjunto con un nombre específico y agrega las celdas que se le indican en el argumento.
`=SUMA(celda1:celda2)`	Suma una fila o columna desde la celda1 hasta la celda2
`=SUMA(nombreConjunto)`	Suma los valores de un conjunto con un nombre específico.
`=MULT(celda1:celda2)`	Multiplica una fila o columna desde la celda1 hasta la celda2.
`=MULT(nombreConjunto)`	Multiplica los valores de un conjunto con un nombre específico.
`=PROMEDIO(celda1:celda2)`	Obtiene el promedio de una fila o columna desde la celda1 hasta la celda2.
`=MEDIANA(celda1:celda2)`	Obtiene la mediana de una fila o columna desde la celda1 hasta la celda2.
`=MIN(celda1:celda2)`	Obtiene la fracción con el valor mínimo de una fila o columna desde la celda1 hasta la celda2.
`=MAX(celda1:celda2)`	Obtiene la fracción con el valor máximo de una fila o columna desde la celda1 hasta la celda2.
`=IMPRIMIR()`	Imprime toda la hoja de cálculo.
`=IMPRIMIR(nombreConjunto)`	Imprime un conjunto específico. Cada fracción va separada por un espacio.

Control de versiones

El proyecto será realizado en parejas en un repositorio de control de versiones. Ambos participantes deben trabajar todas las fases del proceso (análisis, diseño, programación, documentación, pruebas) y deben dejar evidencia en el historial de *commits*. La nota de un participante será proporcional a la cantidad y calidad de los *commits* que realizó.

Haga buen uso del repositorio de control de versiones. Cada *commit* debe tener un cambio y un mensaje significativo, y "no romper el build". No haga "megacommits" que implementan muchas funcionalidades distintas y acumuladas en varios días de desarrollo.

Por ninguna razón deben agregar archivos binarios a control de cambios o que hayan sido generados a través de un proceso automatizado, por ejemplo, los archivos de la carpeta **build**, **ejecutables**, **bibliotecas**, **documentación** generada por Javadoc. En su lugar, deben tener un archivo **.gitignore** adecuado. Deben agregar al profesor del curso como colaborador a su repositorio de control de versiones.

Las carpetas y archivos que se deben agregar al control de versiones son los siguientes:

Tabla 04. Carpetas y archivos el proyecto.

Recurso	Descripción	Versionado
.gradle/	Archivos de Gradle generados automáticamente.	NO
.idea/	Archivos de IntelliJ generados automáticamente.	NO
design/	Contiene el pseudocódigo de sus procedimientos.	SI
doc/	Contiene su documentación generada por Javadoc.	NO
gradle/	Carpeta con archivos Gradle necesarios para construir el proyecto con la versión de Gradle con la que fue creado.	SI
src/	Archivos con su código (archivos fuente).	SI
tests/	Archivos con los casos de prueba facilitados por el profesor.	SI
build.gradle	Contiene configuraciones para construir el proyecto.	SI
gradlew	Contiene configuraciones para construir el proyecto.	SI
gradle.bat	Contiene configuraciones para construir el proyecto.	SI
settings.graddle	Contiene configuraciones para construir el proyecto.	SI
.gitignore	Indica cuales archivos y carpetas no deben ser incluidos al control de versiones.	SI

Análisis

En un archivo README.md en notación Markdown plasme el resultado de la fase de análisis. Si no conoce el formato Markdown, asegúrese de seguir un manual o buscar un tutorial, ya que la buena calidad del código también será evaluada. Por ejemplo, un párrafo en estas notaciones se forma al separar por dos cambios de línea y no por uno. Su documento de análisis debe incluir:

1. Una descripción del problema a resolver. Puede adaptar partes de este enunciado, pero no copiarlas exactamente igual, ya que el enunciado va dirigido al estudiantado, mientras que el README va dirigido a cualquier persona que visite el proyecto y quiera entender el problema que la solución resuelve.
2. Incluir una guía de usuario que indique como compilar el archivo .jar y como ejecutar los casos de prueba.
3. Un análisis detallado del diseño de clases. Debe incluir un diagrama de clases UML donde se especifique los atributos y métodos de cada clase, su visibilidad, y relación don otras clases.

4. Una sección de créditos, donde indica su nombre e información de contacto (correo institucional). Si utiliza recursos de terceros, como código fuente (por ejemplo, si utilizó código de StackOverflow), dé el respectivo crédito.

Diseño

Su proyecto debe traer una carpeta llamada **design** donde debe incluir el pseudocódigo de su proyecto. Durante la etapa de diseño debe enforcarse en la solución general, no es necesario enforcarse en los detalles del lenguaje Java. Utilice las 8 instrucciones básicas del paradigma procedimental (ver en Mediación Virtual) más la instrucción RETURN.

Su diseño debe de utilizar el paradigma de Programación Orientada Objetos. Por esta razón, su solución debe tener un archivo por cada clase. Su proyecto debe tener como mínimo cuatro clases: Controlador, HojaDeCalculo, ListaDeFraciones, y Fraccion. Los nombres anteriores son sugerencias y pueden cambiar.

- La clase controladora es la que se encarga de manejar la lógica del programa. Esta clase posee el método main y run. También puede encargarse de la lectura de datos e impresión de los datos.
- La clase que representa la hoja de cálculo debe de poseer un atributo que debe almacenar los valores y métodos para manipular los atributos. Esta clase también necesita un atributo que almacene los conjuntos (listas de fracciones) y utilice un identificador para poder acceder a esa lista. Se recomienda usar la clase HashMap. Ejemplos de uso pueden ser encontrados en el siguiente enlace: https://www.w3schools.com/java/java_hashmap.asp
- La clase lista fracciones debe de almacenar nodos con fracciones y debe poseer métodos para poder manipular los datos (p. ej., agregar nodo, eliminar nodo, ordenar lista, toString, etc.).
- La clase fracción es similar a la que se vio en clase, pero debe de agregar los métodos necesarios para poder realizar todas las operaciones básicas con fracciones (p. ej., restar, mayor, menor, toString, etc.).

La imagen del **diagrama de clases UML** debe de guardarse en esta carpeta.

Documentación

Su código debe trae documentación interna siguiendo la convención de **Javadoc** vista en clase. Las clases, atributos, procedimientos públicos y privados deben venir documentados. El siguiente ejemplo corto muestra cómo se espera la documentación de un procedimiento:

Tabla 05. Ejemplo de documentación Javadoc.

```
/**
 * El método {@code calcularMediana} calcula la mediana de un arreglo de
 * números reales.
 *
 * @param numElementos Es un número de tipo {@code int} que indica el tamaño
 *                     del arreglo.
 * @param arrayElementos Arreglo de {@code doubles} al que hay que sacar la
 *                       mediana.
 * @return Se retorna un {@code double} con el resultado de la mediana
 *  estadística de arreglo.
 */
public double calcularMediana(final int numElementos,
```

```
                final double[] arrayElementos) { … }
```

Además, debe incluir documentación dentro del cuerpo de los métodos para explicar que hacen las instrucciones más significativas de su procedimiento.

Estilo y Pruebas

Su código debe de apegarse al estándar de **Google** y **SUN** (¡ambas!). Además, debe utilizar nombres significativos para los procedimientos y las variables.

Asegúrese de verificar el funcionamiento de su programa con los casos de prueba (caja negra) que se adjuntan. Tiene que generar al menos una prueba de tamaño y dificultad significativa.

Evaluación

1. [10%] Buen uso de los commits, estructura de archivos y directorios, ignorar archivos autogenerados.
2. [15%] Análisis (README.md).
3. [15%] Diseño de la solución (pseudocódigo, diagramas de clase, e inclusión en el README.md).
4. [10%] Documentación interna: Javadoc y documentación dentro de los procedimientos.
5. [40%] Correcta implementación del programa, nombres significativos, y pasar el CheckStyle (Sun y Google).
6. [10%] Pasar todos los casos de prueba (caja negra).

Cualquier acto de plagio será sancionado severamente.