Part 1

- **DataFrame**: interface used to implement classes such as DataFrameFile and DataFrameDirectory. It contains the following methods:
    - **at**: returns a single value given a row (int) and a column (string).
    - **lat**: returns a single value given a row (int) and a column (string).
    - **columns**: returns the number of columns.
    - **size**: returns the number of rows.
    - **sort**: returns a list of sorted values given a column label (string) and a Comparator of Object.
    - **query**: returns a list of elements where a label fulfills a certain condition given a Predicate of the data structure the LinkedList contains.
    - **getColumnValues**: returns a list of elements of a certain column.
    - **accept**: method to use alongside a Visitor object in order to run Visitor operations.
    - **iterator**: simple DataFrame iterator which iterates through the DataFrame main data structure.
- **DataFrameFile**: DataFrame implementation. Contains all implemented methods previously mentioned and also the DataFrame data structure. It also serves as superclass for classes like CSVDataFrame, JSONDataFrame and TXTDataFrame.
    - **Data structure**: the current LinkedList is rows-oriented which means every List element corresponds to a row but the previous data structure was columns oriented:

      ```
      HashMap<String, LinkedList<Object>>
      ```

      In this case every Map key corresponds to a column label and every List<Object> is one column. This solution was pretty good for column-based operations but we decided to switch over to the current structure when we realized how terrible it is for row-based operations like query. At first we also thought getting columns using the current structure would be a bit tedious but turned out to be pretty easy using Java streams when we learned how to use them.
    - **Why Object?** The reason why we use Object is because the file columns can contain strings or numbers so a generic structure serves well for this purpose. Instead of using multiple number types (int, float,

double…) we just use double since Gson (provided JSON parse library) parses numbers as double regardless of whether they are int or not. Another reason why we just use double is the visitor design pattern (will be explained shortly).

- **CSVDataFrame**: DataFrameFile subclass generated by **CSVDataFrameFactory**.
- **JSONDataFrame**: DataFrameFile subclass generated by **JSONDataFrameFactory**.
- **TXTDataFrame**: DataFrameFile subclass generated by **JSONDataFrameFactory**.
- **DataFrameFactory**: abstract class used to implement methods which are used to parse files. It also serves as superclass for classes like JSONDataFrameFactory, TXTDataFrameFactory and CSVDataFrameFactory.
    - **createDataFrame**: creates a DataFrame object.
    - **removeChars**: this method removes unnecessary characters from a string or array (depends on the paramameter).

Part 2

- **DataFrameDirectory**: DataFrame implementation which contains multiple DataFrame (DataFrameFile/DataFrameDirectory). This class is the only reason why all methods from DataFrame return a list instead of a single value because every implemented method iterates through the whole DataFrame list running the same operation over and over (the same applies to the Iterator), so we have a single list where every element corresponds to a DataFrameFile.

Part 4

- **MapReduce**: class which implements map and reduce methods. It includes a list of Dataframe used to run the operations.
    - **MapColumn**: maps the list given a column label and returns the values.
    - **ReduceMaxMinSum**: maps a list of object and applies reduce given a binary operator to all elements.
    - **ReduceAvg**: maps a list of object and applies average to all elements.
    - **MapMaxRow**: maps the rows with the highest maximum value on a certain column.

Part 5

- **Visitor**: class used to define/implement both visit methods. It also serves as superclass for classes like AverageVisitor, MaximumVisitor, MinimumVisitor and SumVisitor.
    - **visit (DataFrameFile)**: this method is just defined because it will be implemented in the Visitor subclasses (one for every desired operation).
    - **visit (DataFrameDirectory)**: like the methods in DataFrameDirectory this method iterates through the whole directory running accept on every DataFrame which executs visit.
- **SumVisitor**: class used to implement the visit method.
    - **visit**: maps through the given column and applies the sum operation to the elements.
- **MinimumVisitor**: class used to implement the visit method.
    - **visit**: maps through the given column and applies the minimum operation to the elements.
- **MaximumVisitor**: class used to implement the visit method.
    - **visit**: maps through the given column and applies the maximum operation to the elements.
- **AverageVisitor**: class used to implement the visit method.
    - **visit**: maps through the given column and applies the average operation to the elements.

Part 6

- **Observer**: serves as superclass for LogObserver and QueryObserver and contains a list of string used as logger for the executed operations. It implements a getter and a method to add more registers to the list.
- **LogObserver**: observer subclass which tracks all operations applied to a DataFrame.
- **QueryObserver**: observer subclass which tracks query operations applied to a DataFrame.
- **Subject**: used as an observer manager class since it contains a list of Observer. It implements three methods
    - **subscribe**: add an observer to the list.
    - **unsubscribe**: remove an observer from the list.

    - **notify**: updates one or more observers depending on the method name. For example, if the program executes the method query on a DataFrame both observers will be updated.

- **DataFrameInterceptor**: dynamic proxy that acts as a middleman between the target and the method ran on that target. It contains a subject to run notify after a method has been executed.