Monthly Budget Analysis
Matthew Lochman
CIS 106-ONL1 Spring 2018

Project Summary:
The project will focus on the creation of a program that will allow the used to enter an itemized list of their monthly credits and debits from an account. The program will have a feature list including

- Separate tables to view credits and debits. Tables allow sorting by up to two columns simultaneously. (implemented)
- Functional menu bar. (implemented)
- Adding credits and debits dialogs accessible through the Edit menu or right-click context menu. (implemented)
- Removing selected credits and debits (singular or multiple) from Edit menu item or right-click context menu. (implemented)
- Ability to add, edit, or remove options for credit/debit categories and types. (implemented)
- Checkbox trees with years/months for data set. Checking and unchecking particular months set the visibility of the data entries in the tables. (implemented)
- Totals and net amounts displayed in status bar. (implemented)
- Summary statistics for credits and debits display in side pane. (implemented)
- Standard file writing/saving functionality (new, open, save, save as), including warning prompts to save when unsaved data is going to be overwritten (new/open) or lost when closing program. All data as well as changes to possible categories and types are saved. (implemented)
- Undo/Redo features. (not yet implemented)
- Generate report for visible data, which includes summary statistics, pie charts, bar graphs, etc. Report will generate overall as well as monthly analysis when directed. (implemented)
- Printing or generating PDF of the generated report. (not yet implemented)

Project Background:
This project will provide a detailed summary of the user's monthly expenses. As such, it will allow the user to make informed decisions on money matters. In particular, its goal is to highlight information such as typical amounts of discretionary funds available and expenses paid for necessary services such as utilities. It should also provide the user with information necessary to project possible saving (in the form of unused monthly income) as well as give comparisons to previous months to highlight significant differences or abnormal spending patterns within a particular category.

IPOS Requirements:

The user will need to provide various data to the program. Itemized credits and debits should include monetary amounts, the month, day, and year the transaction occurred, and a name or brief description of the transaction. Additionally, the user will provide categories and types for each transactions. For example, possible categories for a debit could include utilities or groceries and possible types for a credit would be direct deposit, cash deposit, and transfer. Data will be stored as a comma delimited text file using an application specific .trd file extension. Previously saved data files are opened by selecting files through a standard open file dialog. Data is displayed in tables based on the visibility of individual months set by the user in a checkbox tree. The user can sort data in the display tables by up to two columns concurrently. The user can also manage the possible categories and types for the transactions through a corresponding dialog accessible in the Edit menu. Based on the checkbox tree selection, output to the user includes total of credits, debits, and net amount, as well as standard summary statistics for the credits and debits. Optional goal 1: Reports to include graphs (pie graphs, bar graphs, histograms) to aid analysis. Optional goal 2: Allow reports to be printed or generated as PDF files.

Class Hierarchy Requirements:

The *MainDisplayController* class manages the main scene for the program. It has numerous variables to store relevant data as well as those related to the different components of the stage whose properties are loaded through an FXML file. It also has various methods, which address various functions. It has Accessor and Mutator methods for its fields so its child windows can access certain private fields. There are a collection of onAction methods used by the menu components linked in the FXML file. Two helper methods, which fix column sizes and create new child windows, respectively. It has a collection of mutator methods to add and remove transaction data and update the displayed statistics data, as well as methods to add and remove nodes from the Checkbox Tree. Finally there is an initialize method to set up everything when the program first opens.

MainDisplayController
- mainController : MainDisplayController - primaryStage : Stage - currentFile : File - needSaved: boolean - creditData : ObservableList<Credit> - debitData : ObservableList<Debit> - creditVisible : FilteredList<Credit> - debitVisibile : FilteredList<Debit> - creditDisplayed : ObservableList<Credit> - debitDisplayed : ObservableList<Debit>


- creditOutliers : ObservableList<Double>
- debitOutliers : ObservableList<Double>
- creditDateMap : ObsevableMap<String, ObservableList<String>>
- debitDateMap : ObservableMap<String, ObservableList<String>>
- treeRoot : TreeItem<String>
- creditRoot : TreeItem<String>
- debitRoot : TreeItem<String>
- creditTable : TableView<Credit>
- creditDescriptionColumn : TableColumn<Credit, String>
- creditDateColumn : TableColumn<Credit, String>
- creditAmountColumn : TableColumn<Credit, String>
- creditCategoryColumn : TableColumn<Credit, String>
- creditTypeColumn : TableColumn<Credit, String>
- debitTable : TableView<Debit>
- debitDescriptionColumn : TableColumn<Debit, String>
- debitDateColumn : TableColumn<Debit, String>
- debitAmountColumn : TableColumn<Debit, String>
- debitCategoryColumn : TableColumn<Debit, String>
- debitTypeColumn : TableColumn<Debit, String>
- dateTree : TreeView<String>
- totalCreditLabel : Label
- totalDebitLabel : Label
- totalNetLabel : Label
- summaryStatTextArea : TextArea
+ setStage(Stage stage) : void
+ setController(MainDisplayController controller) : void
+ setCurrentFile(File file) : void
+ setNeedsSaved(boolean b) : void
+ getTotalCreditLabel () : Label
+ getTotalDebitLabel() : Label
+ getTotalNetLabel() : Label
+ getCreditTextArea() : TextArea
+ getDebitTextArea() : TextArea
+ getController() : MainDisplayController
+ getPrimaryStage() : Stage
+ getCurrentFile() : File
+ getNeedsSaved() : boolean
+ getCreditData() : ObservableList<Credit>
+ getDebitData() : ObservableList<Debit>
+ getCreditRoot() : TreeItem<String>
+ getDebitRoot() : TreeItem<String>
+ getCreditOutliers() : ObservableList<Double>
+ getDebitOutliers() : ObservableList<Double>

+ closeMenuAction(ActionEvent event) : void - newMenuAction(ActionEvent event) : void - openFileMenuAction(ActionEvent event) : void - saveAsFileMenuAction(ActionEvent event) : void - saveFileMenuAction(ActionEvent event) : void - manageCategoryMenuAction(ActionEvent event) : void - manageTypeMenuAction(ActionEvent event) : void - newCreditMenuAction(ActionEvent event) : void - newDebitMenuAction(ActionEvent event) : void - deleteCreditMenuAction(ActionEvent event) : void - deleteDebitFileMenuAction(ActionEvent event) : void + aboutMenuAction(ActionEvent event) : void - generateReportMenuAction(ActionEvent event) : void - setDefaultColumnSize() : void - makeNewDialog(String title, Scene dialogScene) : void + addTransactionData(Credit credit) : void + addTransactionData(Debit debit) : void + removeTransactionData(Credit credit) : void + removeTransactionData(Debit debit) : void - generateCreditSummary() : void - generateDebitSummary() : void + purgeData() : void - addMonthNode(int yearLocation, TreeItem<String> treeRoot, String year, String month) : void - addYearNode(TreeItem<String> treeRoot, String year, String month) : void - removeYearNode(TreeItem<String> treeRoot, String year) : void - removeMonthNode(TreeItem<String> treeRoot, String year, String month) : void - Initialize() : void

The *Transaction* Class is the super-class for the two main transaction types, Credit and Debit. It has fields for the visibility, description, month, day, year, and amount. It comes with two constructors, and accessor methods for each of its fields, as well as formatted string versions of the full date, year, month, and amount. It also has methods for producing a comma delimited file-write string and a standard toString method. Finally, it has mutator methods to set all of its fields.

Transaction
- MONTHS : final static String[] - visibility : SimpleBooleanProperty - description : String - month : int - day : int

- year : int - amount : double
+ Transaction() + Transaction(boolean vis, String desc, int mon, int d, int y, double a) + getDescription() : String + getMonth() : int + getDay() : int + getYear() : int + getDate() : String + getYearString() : String + getMonthString() : String + getAmount() : double + getAmountString() : String + getVisibility() : boolean + getVisibilityProperty() : SimpleBooleanProperty + getWriteString() : String + toString() : String + setDescription(String desc) : void + setMonth(int mon) : void + setDay(int d) : void + setYear(int y) : void + setAmount(double a) : void + setVisibility(boolean b) : void

The *Credit* class is a subclass of the *Transaction* class. It adds new fields for category and type, as well as housing the static lists of possible categories and types. It comes with two contructors. It has static accessor and mutator methods for the category and type lists as well as regular accessors and mutators for the other fields. Additionally, it has three methods that return Predicates for filtering ObservableLists of Credits and one extractor method for ObservableLists to fire change events on the visibility property.

Credit
- category : String - type : String - creditCategories : static ObservableList<String> - creditTypes : static ObservableList<String>
+ Credit() + Credit(boolean vis, String desc, int mon, int d, int y, double a, String cat, String t) + getCreditCategorySize() : static int + getCreditCategory(int i) : static String + getCreditTypeSize() : static int + getCreditType(int i) : static String

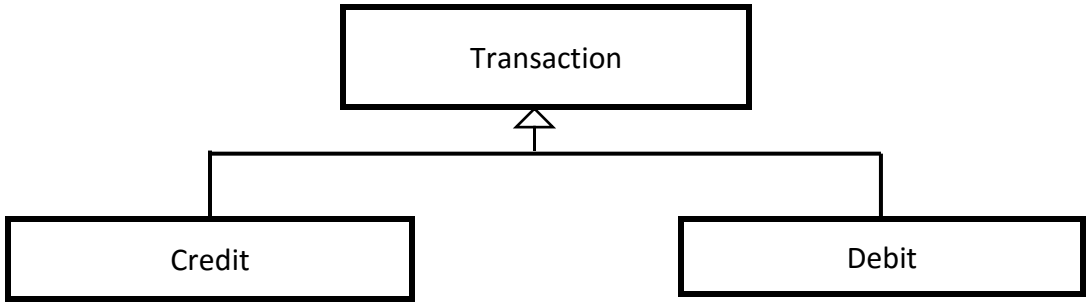
+ getAllCreditCategories() : static ObservableList<String> + getAllCreditMethods() : static ObservableList<String> + addCreditCategory(String s) : static void + removeCreditCategory(String s) : static void + clearCreditCategories() : static void + setDefaultCreditCategories() : static void + addCreditType(String s) : static void + removeCreditType (String s) : static void + clearCreditType () : static void + setDefaultCreditType () : static void + creditVisibilityFilter() : static Predicate<Credit> + creditYearPredicateGenerator(String year) : static Predicate<Credit> + creditMonthPredicateGenerator(String month) : static Predicate<Credit> + creditExtractor(Credit c) : static Callback<Credit, Observable[]> + getCategory() : String + getMethod() : String + getWriteString() : String + toString() : String + setCategory(String cat) : void + setType(String t) : void

The *Debit* class is also a subclass of *Transaction*. It has the exact same fields and methods as *Credit* (but with debit instead).

Debit
- category : String - type : String - debitCategories : static ObservableList<String> - debitTypes : static ObservableList<String>
+ Debit() + Debit(boolean vis, String desc, int mon, int d, int y, double a, String cat, String t) + getDebitCategorySize() : static int + getDebitCategory(int i) : static String + getDebitTypeSize() : static int + getDebitType(int i) : static String + getAllDebitCategories() : static ObservableList<String> + getAllDebitMethods() : static ObservableList<String> + addDebitCategory(String s) : static void + removeDebitCategory(String s) : static void + clearDebitCategories() : static void + setDefaultDebitCategories() : static void + addDebitType(String s) : static void

+ removeDebitType (String s) : static void + clearDebitType () : static void + setDefaultDebitType () : static void + debitVisibilityFilter() : static Predicate<Debit> + debitYearPredicateGenerator(String year) : static Predicate<Debit> + debitMonthPredicateGenerator(String month) : static Predicate<Debit> + debitExtractor(Debit c) : static Callback<Debit, Observable[]> + getCategory() : String + getMethod() : String + getWriteString() : String + toString() : String + setCategory(String cat) : void + setType(String t) : void

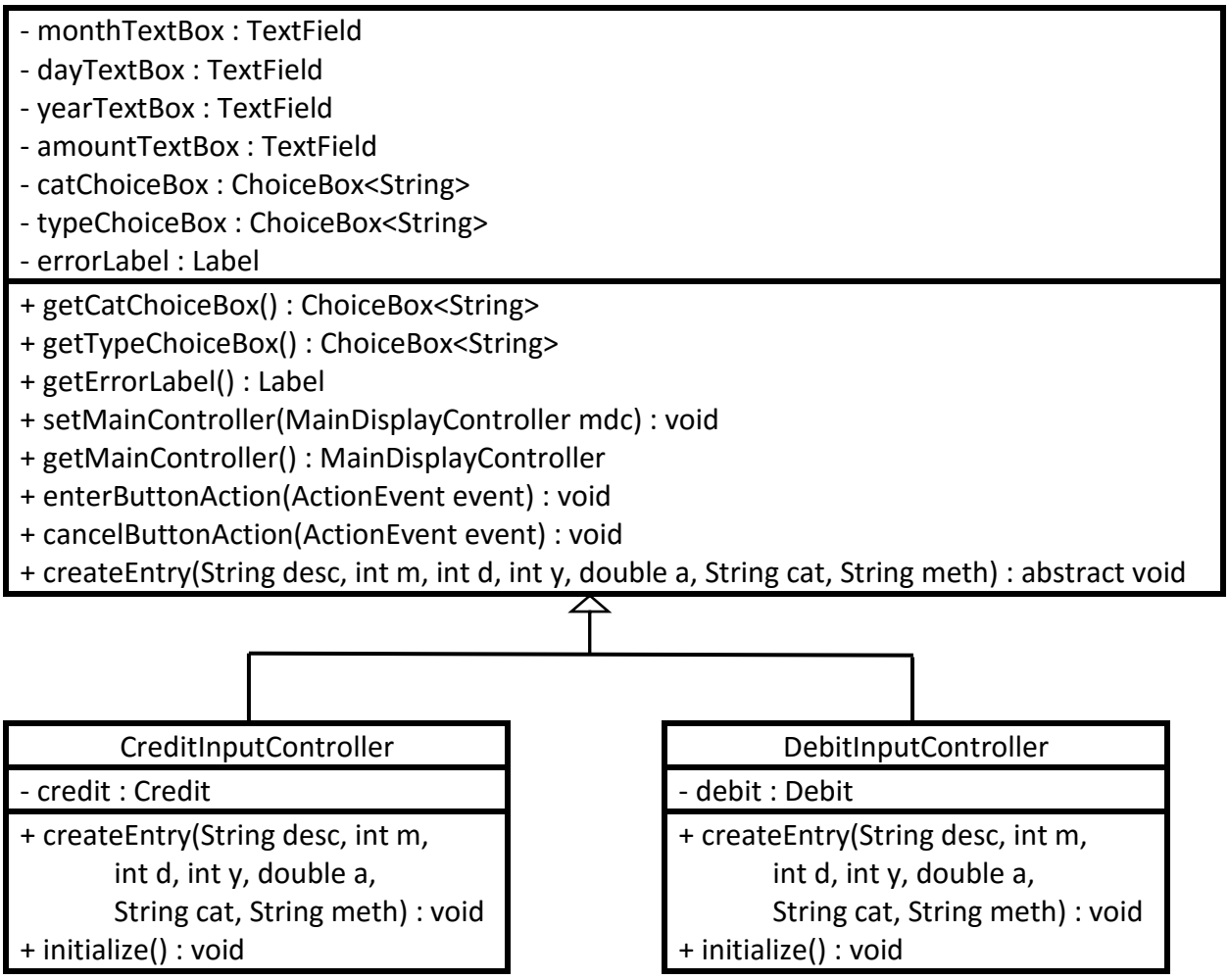
Since the classes themselves take up too much space to show inheritance, we include an abbreviated diagram here:



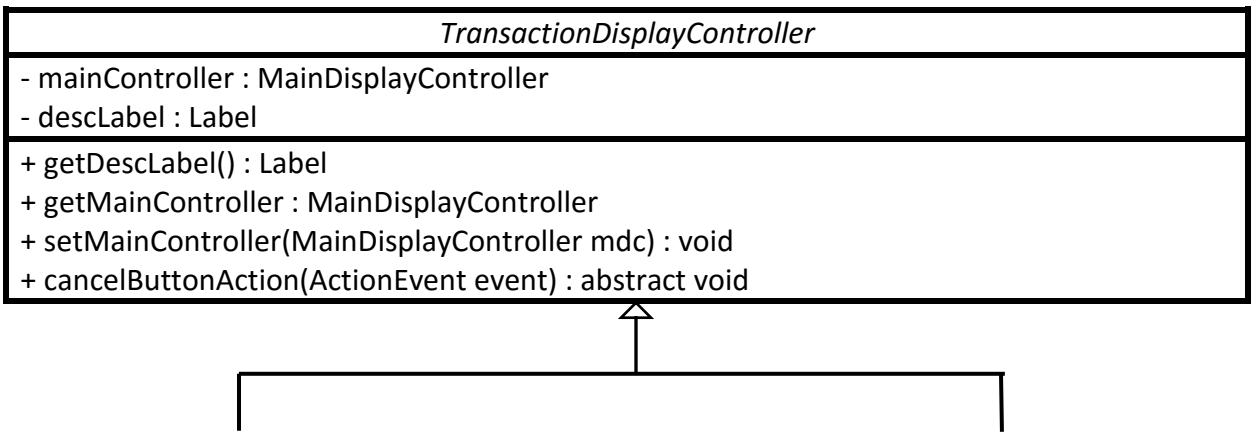
The *TransactionInputController* class is an abstract class that builds the groundwork for the data input scene. It has a field to hold a reference to the main display controller as well as for the scene components from its FXML file. It has accessor methods so its subclasses can access some of the scene components. It also has onAction methods for the buttons. In particular, the enter button's onAction method handles input validation. Finally, it has an abstract method createEntry, to be overridden by its subclasses.

The *CreditInputController* class is a subclass of *TransactionInputController* that adds Credit specific functionality. The *DebitInputController* class is a subclass of *TransactionInputController* that adds Debit specific functionality.

TransactionInputController
- mainController : MainDisplayController - descTextBox : TextField



Similar to the input controller, the *TransactionDisplayController* is an abstract class for displaying the confirmation scene after the user inputs a transaction. It has some basic fields for the window layout and one abstract method.
The *CreditDisplayController* and *DebitDisplayControllers* extend the *TransactionDisplayController*. They have one object field and two methods.



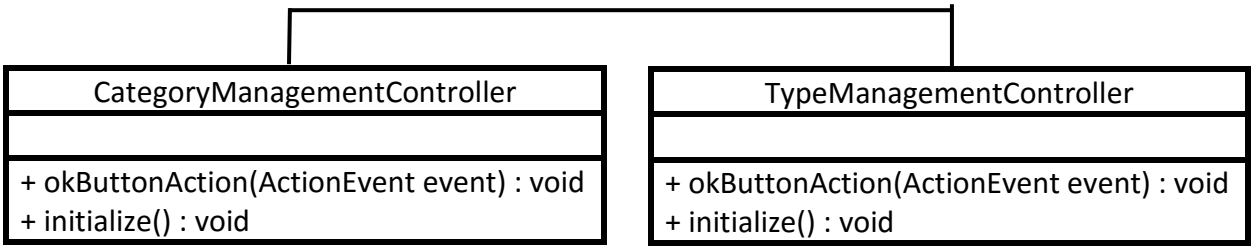
CreditDisplayController
- credit : Credit
+ cancelButtonAction (ActionEvent event) : void + setCredit(Credit c) : void

DebitDisplayController
- debit : Debit
+ cancelButtonAction (ActionEvent event) : void + setDebit(Debit c) : void

The *ManagementController* abstract class sets the stage for managing the transaction categories and types. It has fields for the credit and debit items being managed, a String to refer to the item type (credit or type), and the ListViews for the items. It's methods include accessors for these fields and a setter method for the item type field. It also has action methods for the eight buttons in the scene.

The *CategoryManagementController* and *TypeManagementController* are extensions of the *ManagementController* that allow the user to manage the credit and debit categories and types, respectively. Each overrides the okButtonAction method and have individual initialize methods to populate the ListViews.

ManagementController
- itemType : String - creditItems : ObservableList<String> - debitItems : ObservableList<String> - creditListView : ListView<String> - debitListView : ListView<String>
+ getCreditItems() : ObservableList<String> + getDebitItems() : ObservableList<String> + getCreditListView() : ListView<String> + getDebitListView() : ListView<String> + getItemType() : String + setItemType(String s) : void - addDialog () : String - editDialog() : String + addCreditButtonAction (ActionEvent event) : void + editCreditButtonAction (ActionEvent event) : void + removeCreditButtonAction (ActionEvent event) : void + addDebitButtonAction (ActionEvent event) : void + editDebitButtonAction (ActionEvent event) : void + removeDebitButtonAction (ActionEvent event) : void + okButtonAction (ActionEvent event) : abstract void + cancelButtonAction (ActionEvent event) : void



The *SplashScreenController* is a simple class that manages the splash screen scene, which displays when the program first runs or when the about menu option is chosen.

SplashScreenController
- okButton : Button
+ okButtonAction(ActionEvent event) : void

The *DataIO* class houses the methods related to file reading and writing for the program. It has one field to hold the mainController reference and a contructor that sets this value. Its methods are used to alert the user to save unsaved changes, create a file save/open dialog, write the data in a comma delimited file, and read data from such a file.

DataIO
- mainController : MainDisplayController
+ DataIO(MainDisplayController mdc) + checkDataSaveState() : String + createFileDialog(String s) : boolean - readDataFile(File file) : void - writeDataFile(File file) : void

The *StatCalculator* class calculates summary statistics for a List of doubles as well as identifying data outliers. It has fields for these statistics and two constructors. Its methods include accessor methods for the fields as well as calculation methods for the statistics.

StatCalculator
- sum : double - aMean : double - sampStDev : double - popStDev : double - min : double - quartile1 : double - median : double

- quartile3 : double - max : double - IQR : double - outliers : ObservableList<Double> - dataList : ObservableList<Double> - n : int
+ StatCalculator() + StatCalculator(ObservableList<Double> data) + getSum() : double + getAMean() : double + getSampStDev() : double + getPopStDev() : double + getMin() : double + getQuartile1() : double + getMedian() : double + getQuartile3() : double + getMax() : double + getIQR() : double + getOutliers() : ObservableList<Double> + toString() : String + toMoneyString : String - calcSum() : void - calcMean() : void - calcStDev() : void - calcMedian(List<Double> list) : double - calc5NumSum() : void - calcOutliers() : void

The *ReportDisplayController* class gives various summary tools in a tabbed view for the user. These include the summary statistics and a list of all credits and debits that are considered outliers, a set of pie charts that give credits and debit transactions grouped by category, and a set of stacked bar charts that give each month broken down by category. It has fields for the different components of the scene and various methods to generate all of the above mentioned features.

ReportDisplayController
- mainController : MainDisplayController - primaryStage : Stage - creditDisplayed : ObservableList<Credit> - debitDisplayed : ObservableList<Debit> - creditTextArea : TextArea - debitTextArea : TextArea

- creditable : TableView<Credit> - creditDescriptionColumn : TableColumn<Credit, String> - creditDateColumn : TableColumn<Credit, String> - creditAmountColumn : TableColumn<Credit, String> - creditCategoryColumn : TableColumn<Credit, String> - creditTypeColumn : TableColumn<Credit, String> - debitTable : TableView<Debit> - debitDescriptionColumn : TableColumn<Debit, String> - debitDateColumn : TableColumn<Debit, String> - debitAmountColumn : TableColumn<Debit, String> - debitCategoryColumn : TableColumn<Debit, String> - debitTypeColumn : TableColumn<Debit, String> - totalCreditLabel : Label - totalDebitLabel : Label - totalNetLabel : Label - creditPieChart : PieChart - debitPieChart : PieChart - creditStackedBarChart : StackedBarChart<String, Double> - debitStackedBarChart : StackedBarChart<String, Double>
+ setMainController(MainDisplayController mdc) : void + setPrimaryStage(Stage stage) : void - closeMenuAction(ActionEvent event) : void - generateContent() : void - generateCreditPieChart() : void - generateDebitPieChart() : void - generateCrediStackedBarChart() : void - generateDebitStackedBarChart() : void - setDefaultColumnSize() : void - initialize() : void

Conclusion:

This program has a variety of uses for the user as highlighted in the background section above. Much of this should be achievable from a basic programming perspective (though producing dynamic pie charts might be impossible). Although many of these features are typically built in to any online banking system, the implementation of these features should prove to be a sufficient challenge both in scope and complexity. Also, some of these are not always included, or if they are they are challenging to find or compile in one location. One of the main reasons behind this project idea is to provide the information that I want in one place.

Glossary

Credit	A transaction that increases the amount of funds in an account.
Debit	A transaction that decreases the amount of funds in an account.
Comma Delimited	Refers to the text data being separated by commas in the file.
Context Menu	The menu that opens when you right-click something with the mouse.
Summary Statistics	Statistics including the mean, standard deviation, sum, minimum, first quartile, median, third quartile, and maximum.
Mean	The sum of a collection of numbers divided by the number of numbers in the collection.
Standard Deviation	A measure that is used to quantify the amount of variation or dispersion of a set of data values.
First Quartile	The middle number between the smallest number and the median of the data set.
Median	The value separating the higher half of a data sample, a population, or a probability distribution, from the lower half.
Third Quartile	The middle value between the median and the highest value of the data set.

References

Accounting Coach, Bank's debits and credits, https://www.accountingcoach.com/debits-and-credits/explanation/4

Wikipedia, Comma-separated values, https://en.wikipedia.org/wiki/Comma-separated_values

Wikipedia, Arithmetic mean, https://en.wikipedia.org/wiki/Arithmetic_mean

Wikipedia, Standard Deviation, https://en.wikipedia.org/wiki/Standard_deviation

Wikipedia, Median, https://en.wikipedia.org/wiki/Median

Wikipedia, Quartiles, https://en.wikipedia.org/wiki/Quartile