Raymond Shum
CST338
December 18, 2020
Final Project – Assig 8

SimpleERP Application

Overview

SimpleERP is a console application built using the Model View Controller design pattern. It contains three modules that allow a single user to manage their staff roster, update and remove an employee's sales information and to view a comprehensive sales report. The application is composed of 7 classes: Main, Model, View, Controller, Employee, SalesRecord and Sales.

Menus

Main Menu

```
Main Menu
Please select your option:
1) Manage Employees
2) Manage Employee Sales
3) View Sales Report
Enter your selection ("Q" to quit):
```

The Main Menu is the first menu that the user will see. It allows the user to reach each of the three modules.

Manage Employees Menu

```
================
Manage Employees
================

First Name          Last Name           Employee ID

Larry               Chiem               100
Raymond             Shum                101
Ian                 Rowe                102
Nicholas            Stankovich          103

(B) Back to Main Menu (F) Sort by First Name (L)Sort by Last Name (E)Sort by ID
(A) Add Employee (R) Remove Employee
Enter your selection ("Q" to quit):
```

The Manage Employee Menu allows the user to add and remove employees from their staff list as well as sort the list for readability. Adding and removing the employees will also add and remove their associated sales from the Model.

Manage Employee Sales Menu

```
====================
Mange Employee Sales
====================

First Name          Last Name           Employee ID

Larry               Chiem               100
Raymond             Shum                101
Ian                 Rowe                102
Nicholas            Stankovich          103

(B) Back to Main Menu (F) Sort by First Name(L)Sort by Last Name (E)Sort by ID
(V) View Employee Sales Record
Enter your selection ("Q" to quit):
```

The Manage Employee Sales Menu allows a user to select an employee and view their sales history.

Employee Sales History Menu

```
==================================
Raymond Shum's Sales History
==================================
Sale ID             Date of Sale        Sales Amount

1010                2019-03-05          $90
1011                2020-05-17          $2300
1012                2020-10-09          $150

(B) Back to View Employees (T) Sort by Higest Total Sales (S) Sort by Most Recent Sales
(O) Sort by Oldest Sales (L) Sort by Lowest Total Sales (A) Add Sale (R) Remove Sale
Enter your selection ("Q" to quit):
```

A submenu for Manage Employee Sales, the Employee Sales History Menu allows a user to add and remove sales entries from an employee's sales history. It also allows them to sort the entries for readability.

```
==================
Master Sales Record
==================
Sale ID             Date of Sale        Sales Amount

1000                2019-01-20          $1500
1001                2020-05-01          $100
1002                2020-07-23          $3000
1010                2019-03-05          $90
1011                2020-05-17          $2300
1012                2020-10-09          $150
1020                2020-04-02          $500
1021                2020-05-07          $5000
1022                2020-07-01          $2500
1030                2019-10-04          $3500
1031                2020-09-27          $2500
1032                2020-12-15          $1500
============================================================
Total Sales:                            $22640
(B) Back to View Employees (T) Sort by Higest Total Sales (S) Sort by Most Recent Sales
(O) Sort by Oldest Sales (L) Sort by Lowest Total Sales (I) Sort by Sale ID
Enter your selection ("Q" to quit):
```

Master Sales Record Menu

The Master Sales record Menu allows a user to view a comprehensive report on the sales totals for the organization. Sales entries are displayed with Sale ID as their identifier, rather than employee ID.

Output

The application will perform validation and confirm and selection choices for all user input. Non-exhaustive examples below.

Invalid Menu Selection

```
Main Menu
Please select your option:
1) Manage Employees
2) Manage Employee Sales
3) View Sales Report
Enter your selection ("Q" to quit): test
Invalid entry. Please select one of the displayed options.
```

Invalid Option Selection

```
================
Manage Employees
================

First Name          Last Name           Employee ID

Larry               Chiem               100
Raymond             Shum                101
Ian                 Rowe                102
Nicholas            Stankovich          103

(B) Back to Main Menu (F) Sort by First Name (L)Sort by Last Name (E)Sort by ID
(A) Add Employee (R) Remove Employee
Enter your selection ("Q" to quit): g
Invalid entry. Please select one of the displayed options.
```

Adding/Removing Employees

```
Are you sure you want to add a new employee? (Y/N)
b
Invalid entry. Please select one of the displayed options.
Are you sure you want to add a new employee? (Y/N)
t
Invalid entry. Please select one of the displayed options.
Are you sure you want to add a new employee? (Y/N)
y
Please enter the employee's first name:
Dave
Please enter the employee's last name:
123a
First/Last names should not contain digits.
Please enter the employee's last name:
Simpson
You have entered: Dave Simpson. Is this correct? (Y/N)
f
Invalid entry. Please select one of the displayed options.
You have entered: Dave Simpson. Is this correct? (Y/N)
y
```

Adding/Removing Sales

```
Please enter the date of sale in the form of YYYY-MM-DD
2020-18-20
You have entered an invalid date. Please try again.
Please enter the date of sale in the form of YYYY-MM-DD
2020-05-12
You have entered 2020-05-12. Is this correct? (Y/N)
y
Please enter the sale amount as a whole number:
test
Sale amount must be a whole number and only contain digits. Please try again.
Please enter the sale amount as a whole number:
100
You have entered 100. Is this correct? (Y/N)
y
```

Quitting the Application

```
Are you sure you want to quit? (Y/N)
g
Invalid entry. Please select one of the displayed options.
Are you sure you want to quit? (Y/N)
y
You have confirmed that you wish to quit. Goodbye!
```

Implementation

Overview

The application is built using the Model View Controller design pattern. The main will call the Model, View and Controller objects. Controller will interact directly with Model and View. View and Model are separated by Controller. Model will interact with the Sale, SalesRecord and Employee classes. Specifically, Model holds a SalesRecord object and an Employee[] object and Employee holds a Sale[] object.

All classes that are part of the model: Model, Sale, SalesRecord and Employee should have accessors, mutators, copy constructors and toString methods as well.

For all entries, please reference the UML for a list of methods/variables and the Javadoc for documentation.

Main

The Main exists simply to call the Model, View and Controller constructors.

Controller

The Controller takes user input and uses its logic to update Model through its accessors and mutators. It calls methods from View to provide a representation of the current state of Model.

View

The View is composed of methods to display the menu, options and errors to the user. It takes formatted Strings, passed by Controller and incorporates them as the body for these menus. It does not hold any instance variables and does not interact with Model. It is reliant on Controller to pass processed String objects to display the state of Model.

Model

The Model holds the data of the application, as well as accessors and mutators used by Controller for manipulation. Model holds the representation of the organization: its staff list and Master Sales Record. In order to maintain its staff and sales arrays, Model will interact with 3 objects: Employee, SalesRecord and Sales.

Sale

The Sale class represents an individual employee's total revenue generated in a single day. It holds a LocalDate variable of the date that the sale was made. It also holds salesAmount, an integer value (whole numbers only) that represents the total value of the revenue generated on that day. It also holds the Sale ID of the object, which is a unique identifier used to identify individual sales.

Sale also holds a static method for sorting incoming Sale[].

SalesRecord

The SalesRecord class holds an array of Sale objects as well as the current number of Sale objects held in the array. The class is responsible for maintaining an accurate representation of the array with its methods. It must be self-contained, meaning that external validation and filtering of parameters can be performed but is unnecessary.

Employee

The Employee class represents and employee of the organization that is using SimpleERP. The class holds a SalesRecord object, which represents the employee's complete sales history. It also holds identifying information, such as the employee's name and ID.

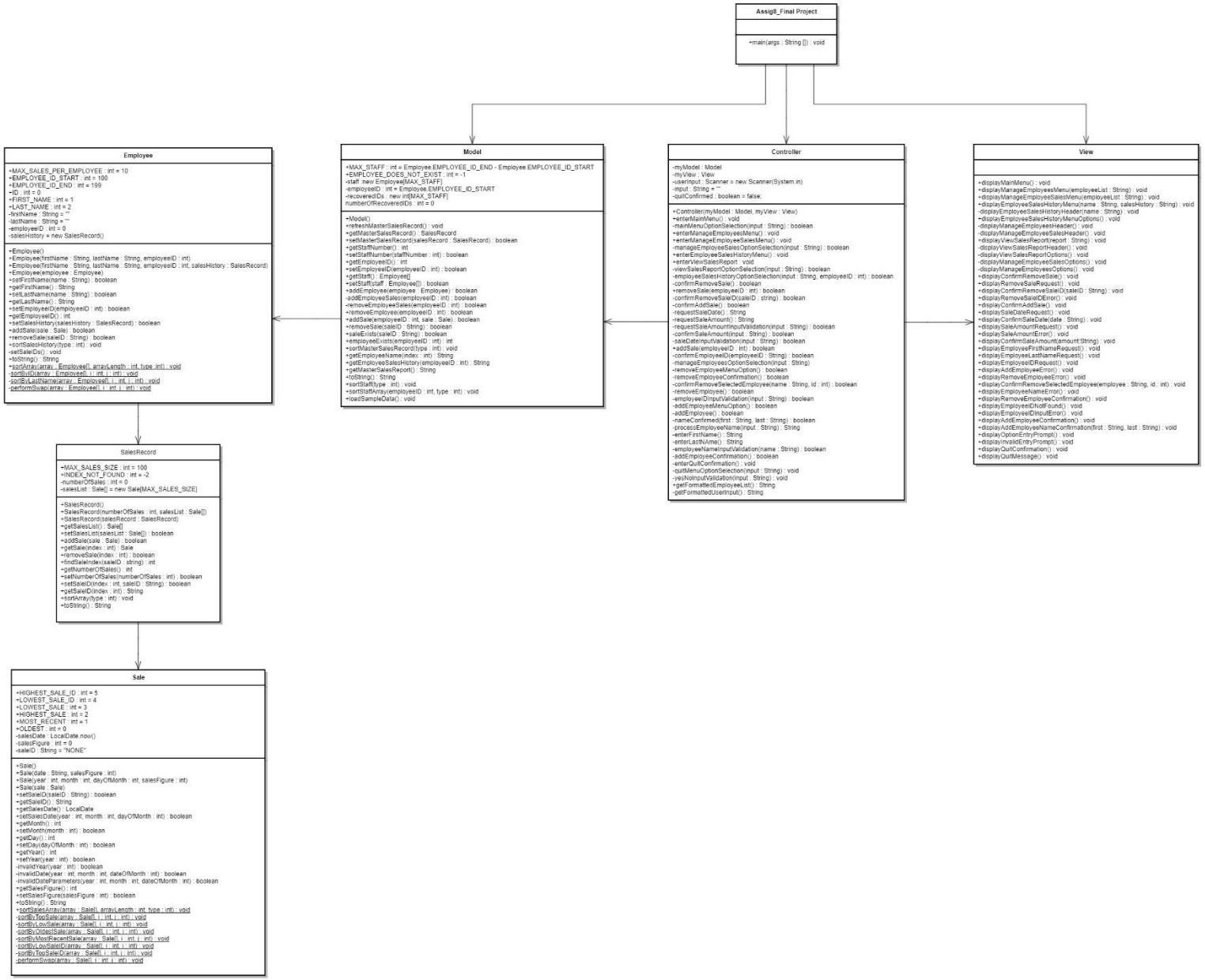
Employee also holds a static method for sorting incoming Employee[].

UML

Has been attached for convenience and will also be present in the project upload. Please refer to the following page for the UML diagram.

Javadoc

The Javadoc will be present in the upload for reference. It can also be generated by running the command on the source code, also included in the project upload.


Assig8_Final Project
+main(args : String []) : void
Employee
+MAX_SALES_PER_EMPLOYEE : int = 10
+EMPLOYEE_ID_START : int = 100
+EMPLOYEE_ID_END : int = 199
+ID : int = 0
+FIRST_NAME : int = 1
+LAST_NAME : int = 2
-firstName : String = ""
-lastName : String = ""
-employeeID : int = 0
-salesHistory = new SalesRecord()
+Employee()
+Employee(firstName : String, lastName : String, employeeID : int)
+Employee(firstName : String, lastName : String, employeeID : int, salesHistory : SalesRecord)
+Employee(employee : Employee)
+setFirstName(name : String) : boolean
+getFirstName() : String
+setLastName(name : String) : boolean
+getLastName() : String
+setEmployeeID(employeeID : int) : boolean
+getEmployeeID() : int
+setSalesHistory(salesHistory : SalesRecord) : boolean
+addSale(sale : Sale) : boolean
+removeSale(saleID : String) : boolean
+sortSalesHistory(type : int) : void
-setSaleIDs() : void
+toString() : String
+sortArray(array : Employee[], arrayLength : int, type :int) : void
-sortByID(array : Employee[], i : int, j : int) : void
-sortByLastName(array : Employee[], i : int, j : int) : void
-performSwap(array : Employee[], i : int, j : int) : void
Model
+MAX_STAFF : int = Employee.EMPLOYEE_ID_END - Employee.EMPLOYEE_ID_START
+EMPLOYEE_DOES_NOT_EXIST : int = -1
-staff :new Employee[MAX_STAFF]
-employeeID : int = Employee.EMPLOYEE_ID_START
-recoveredIDs : new int[MAX_STAFF]
numberOfRecoveredIDs : int = 0
+Model()
+refreshMasterSalesRecord() : void
+getMasterSalesRecord() : SalesRecord
+setMasterSalesRecord(salesRecord : SalesRecord) : boolean
+getStaffNumber() : int
+setStaffNumber(staffNumber : int) : boolean
+getEmployeeID() : int
+setEmployeeID(employeeID : int) : boolean
+getStaff() : Employee[]
+setStaff(staff : Employee[]) : boolean
+addEmployee(employee : Employee) : boolean
-addEmployeeSales(employeeID : int) : boolean
-removeEmployeeSales(employeeID : int) : boolean
+removeEmployee(employeeID : int) : boolean
+addSale(employeeID : int, sale : Sale) : boolean
+removeSale(saleID : String) : boolean
+saleExists(saleID : String) : boolean
+employeeExists(employeeID : int) : int
+sortMasterSalesRecord(type : int) : void
+getEmployeeName(index : int) : String
+getEmployeesSalesHistory(employeeID : int) : String
+getMasterSalesReport() : String
+toString() : String
+sortStaff(type : int) : void
+sortStaffArray(employeeID : int, type : int) : void
+loadSampleData() : void
Controller
-myModel : Model
-myView : View
-userInput : Scanner = new Scanner(System.in)
-input : String = ""
-quitConfirmed : boolean = false;
+Controller(myModel : Model, myView : View)
+enterMainMenu() : void
-mainMenuOptionSelection(input : String) : boolean
+enterManageEmployeesMenu() : void
+enterManageEmployeeSalesMenu() : void
-manageEmployeeSalesOptionSelection(input : String) : boolean
+enterEmployeeSalesHistoryMenu() : void
+enterViewSalesReport : void
-viewSalesReportOptionSelection(input : String) : boolean
-employeeSalesHistoryOptionSelection(input : String, employeeID : int) : boolean
-confirmRemoveSale() : boolean
+removeSale(employeeID : int) : boolean
-confirmRemoveSaleID(saleID : string) : boolean
-confirmAddSale() : boolean
-requestSaleDate() : String
-requestSaleAmount() : String
-requestSaleAmountInputValidation(input : String) : boolean
-confirmSaleAmount(input : String) : boolean
-saleDateInputValidation(input : String) : boolean
+addSale(employeeID : int) : boolean
-confirmEmployeeID(employeeID : String) : boolean
-manageEmployeesOptionSelection(input :String)
-removeEmployeeMenuOption() : boolean
-removeEmployeeConfirmation() : boolean
-confirmRemoveSelectedEmployee(name :String, id : int) : boolean
-removeEmployee() : boolean
-employeeIDInputValidation(input : String) : boolean
-addEmployeeMenuOption() : boolean
-addEmployee() : boolean
-nameConfirmed(first : String, last : String) : boolean
-processEmployeeName(input : String) : String
-enterFirstName() : String
-enterLastNAme() : String
-employeeNameInputValidation(name : String) : boolean
-addEmployeeConfirmation() : boolean
-enterQuitConfirmation() : void
-quitMenuOptionSelection(input : String) : void
-yesNoInputValidation(input : String) : void
+getFormattedEmployeeList() : String
-getFormattedUserInput() : String
View
+displayMainMenu() : void
+displayManageEmployeesMenu(employeeList : String) : void
+displayManageEmployeeSalesMenu(employeeList : String) : void
+displayEmployeeSalesHistoryMenu(name : String, salesHistory : String) : void
-displayEmployeeSalesHistoryHeader(name : String) : void
+displayEmployeeSalesHistoryMenuOptions() : void
-displayManageEmployeesHeader() : void
-displayManageEmployeeSalesHeader() : void
+displayViewSalesReport(report : String) : void
-displayViewSalesReportHeader() : void
-displayViewSalesReportOptions() : void
-displayManageEmployeeSalesOptions() : void
-displayManageEmployeesOptions() : void
+displayConfirmRemoveSale() : void
+displayRemoveSaleRequest() : void
+displayConfirmRemoveSaleID(saleID : String) : void
+displayRemoveSaleIDError() : void
+displayConfirmAddSale() : void
+displaySaleDateRequest() : void
+displayConfirmSaleDate(date : String) : void
+displaySaleAmountRequest() : void
+displaySaleAmountError() : void
+displayConfirmSaleAmount(amount:String) : void
+displayEmployeeFirstNameRequest() : void
+displayEmployeeLastNameRequest() : void
+displayEmployeeIDRequest() : void
+displayAddEmployeeError() : void
+displayRemoveEmployeeError() : void
+displayConfirmRemoveSelectedEmployee(employee : String, id : int) : void
+displayEmployeeNameError() : void
+displayRemoveEmployeeConfirmation() : void
+displayEmployeeIDNotFound() : void
+displayEmployeeIDInputError() : void
+displayAddEmployeeConfirmation() : void
+displayAddEmployeeNameConfirmation(first : String, last : String) : void
+displayOptionEntryPrompt() : void
+displayInvalidEntryPrompt() : void
+displayQuitConfirmation() : void
+displayQuitMessage() : void
SalesRecord
+MAX_SALES_SIZE : int = 100
+INDEX_NOT_FOUND : int = -2.
-numberOfSales : int = 0
-salesList : Sale[] = new Sale[MAX_SALES_SIZE]
+SalesRecord()
+SalesRecord(numberOfSales : int, salesList : Sale[])
+SalesRecord(salesRecord : SalesRecord)
+getSalesList() : Sale[]
+setSalesList(salesList : Sale[]) : boolean
+addSale(sale : Sale) : boolean
+getSale(index : int) : Sale
+removeSale(index : int) : boolean
+findSaleIndex(saleID : string) : int
+getNumberOfSales() : int
+setNumberOfSales(numberOfSales : int) : boolean
+setSaleID(index : int, saleID : String) : boolean
+getSaleID(index : int) : String
+sortArray(type : int) : void
+toString() : String
Sale
+HIGHEST_SALE_ID : int = 5
+LOWEST_SALE_ID : int = 4
+LOWEST_SALE : int = 3
+HIGHEST_SALE : int = 2
+MOST_RECENT : int = 1
+OLDEST : int = 0
-salesDate : LocalDate.now()
-salesFigure : int = 0
-saleID : String = "NONE"
+Sale()
+Sale(date : String, salesFigure : int)
+Sale(year : int, month : int, dayOfMonth : int, salesFigure : int)
+Sale(sale : Sale)
+setSaleID(saleID : String) : boolean
+getSaleID() : String
+getSalesDate() : LocalDate
+setSalesDate(year : int, month : int, dayOfMonth : int) : boolean
+getMonth() : int
+setMonth(month : int) : boolean
+getDay() : int
+setDay(dayOfMonth : int) : boolean
+getYear() : int
+setYear(year : int) : boolean
-invalidYear(year : int) : boolean
-invalidDate(year : int, month : int, dateOfMonth : int) : boolean
-invalidDateParameters(year : int, month : int, dateOfMonth : int) : boolean
+getSalesFigure() : int
+setSalesFigure(salesFigure : int) : boolean
+toString() : String
+sortSalesArray(array : Sale[], arrayLength : int, type : int) : void
-sortByTopSale(array : Sale[], i : int, j : int) : void
-sortByLowSale(array : Sale[], i : int, j : int) : void
-sortByOldestSale(array : Sale[], i : int, j : int) : void
+sortByMostRecentSale(array : Sale[], i : int, j : int) : void
-sortByLowSaleID(array : Sale[], i : int, j : int) : void
-sortByTopSaleID(array : Sale[], i : int, j : int) : void
-performSwap(array : Sale[], i : int, j : int) : void

Testing

Overview

Methods for all classes should be unit tested. Methods should be decomposed to help do so. There should be an attempt to adhere to this rule of thumb: a method should do one thing well.

For all classes with arrays, accessors and mutators should ensure that there are no privacy leaks. Methods involving arrays should never throw NullPointerException or ArrayIndexOutOfBounds exceptions. Testing should be done to ensure this is the case.

For classes that take user input, input must be validated and filtered before passing as a parameter to any classes in the model. This should be done even if the model classes are designed to filter for bad values. As such, input validation should be tested for all methods that make use of it.

For any classes with mutators and accessors, testing should be done to ensure that the proper values are being set or returned.

For the View class, testing should be done to ensure that formatting is correct on displayed messages.

For the Controller class, testing should be done to ensure that the application logic is function correctly. For example, option selections should take the user to the correct menu. Properly formatted input should pass validation and improperly formatted input should be rejected. There should be no infinite loops that the user is unable to exit.

Known Issues

Design

These are issues with the application design that are currently unresolved but would be addressed if this project had another phase for implementation.

1. Application only allows for one concurrent user. To allow for more than one user, array operations need to be placed in a thread that performs synchronized calls.
2. SaleID is not truly unique. It is currently the concatenation of EmployeeID and the index of the element on the saleHistory array. This is fine for day to day operations, but in a real organization with need to reference sales over the course of months and years, a truly unique identifier should be generated.
3. EmployeeID functions like an IP address block. EmployeeIDs are granted from a block of integer values, recovered when an employee is removed and redistributed when a new employee is onboarded. In a real organization, old IDs may be maintained for a certain period for the purpose of record keeping.

Bugs

These are bugs that are currently unaddressed but would be addressed if this project had another phase for implementation.

1. User input returns multiple words for a single line – Unaddressed because input validation occurs for each returned word and user is prompted to confirm if choices are correct.
2. It is possible to select the option to add/remove employees/sales from an empty/full array – unaddressed because the classes prevent the user from actually adding/removing these objects to a full/empty array. However, the user should not have the option to initiate these method calls.