University of Ottawa
Faculty of Engineering
Department of Chemical and Biological Engineering

CHG4343 – Computer-Aided Design in Chemical Engineering

Distillation Column Design and Economic Analysis

Report

Group 7:
Alenko, Tatum 5553340
Kari, Sandra 3237223
Kay, Jordon 6312787
Leung, Janice 6389179
Song, Yun 6485635

Executive Summary

The report is aimed at detailing the design of a distillation simulator written in the JAVA programming language with a strong object-oriented frame of mind. The objective of this application is to allow users to determine the theoretical amount of trays within a distillation column that would be required given a series of problem parameters and constraints. From the theoretical number of trays, the total cost and profit of the distillation column was determined and compared to other ventures to determine the most profitable one. This simulator contains a variety of object oriented programming aspects such as polymorphism, inheritance, interface, cloning, exception handling File I/O, etc. In the design of the simulator, the McCabe-Thiele method, in an equivalent way to a graphical method, was used to determine the theoretical number of trays that are necessary to separate two components into their required mole fraction. With the use of object-oriented programming, the design of this simulator mainly revolved around the object concept of a column and streams, which were broken down into separate classes with deep levels of inheritance representing the multiple and modularized components that make up process operations such as distillation It was determined that of the three ventures provided, Venture 1 was assessed to be the most profitable with a 1-year payback period of $30,542,547.12 and a tray requirement of 6. Moreover, the validations conducted of the designed simulator confirm that the calculated results are accurate and satisfy the design requirements sought after the client, 2PS.

Table of Contents

List of Tables

List of Figures

Nomenclature

R	Reflux ratio
x_F	Component fraction in feed stream
x_D	Component fraction in distillate
x_B	Component fraction in bottoms
ΔH^{vap}	Latent heat (kJ/kmol)
C_P	Specific heat (kJ/kmolK)
T_B	Boiling point (K)
T_F	Feed temperature (K)
q	Feed quality

1 Introduction

Java's most significant advantage is the ability to move easily from one system to another. When Java is compiled, it is compiled into a platform independent byte code, which is interpreted by Java Virtual Machine (JVM) when the code is being run. Thus, the same cade can be run in different system with Java. Moreover, Java uses object orientated programming language, which allows modular maintainable application.

Object Orientated Programming (OOP) refers to the programming technique that combines data structures with functions to create reusable objects. The advantage of object-orientated programming over the procedural programming is the easiness of modifying the program. For instance, it enables the programmers to create modules that do not need to be changed when another object with similar features is created. In OOP, objects are the important basic run-time entities. The OOP requires careful application of abstractions and division of problems into manageable pieces, which follows the mechanism of problem solving. When an OOP is used for problem solving, the problem is analyzed in terms of objects and rely on the communication between them, as such when a program is executed, objects interact with each other by sending messages.

The two fundamental concepts in an OOP are class and objects. Objects are instances of a class. They can be variables, functions, and data structures. In an object-orientated world, objects can be interpreted as objects in the real-word, which would always have states and behaviors. Some examples in a chemical engineering problem are reactor, distillation column, linear function. States can be referred as the characteristics of the object such as name, materials of construction, and wall thickness. Behaviors are presented through methods, such as calculation methods. Classes are the blueprint of individual objects. Objects of the same type are in one class. Methods are included in the class as well. The most fundamental concepts in OOP are abstraction, encapsulation, polymorphism, and inheritance, respectively.

Abstraction is the concept of dealing with ideas rather than the real event. It is achieved in Java by implementing interfaces and abstract classes. When abstraction is implemented in the code, the details of the process is hidden from the other objects and only the essential features are shown. For members that have features changing frequently, they are separated into subclasses logically according to their functions and properties. Thus, abstractions help the design of system becomes flexible and easy to maintain. This concept applied on abstract classes and abstract

methods. In an abstract class, some basic properties and functions are included. Abstract classes may or may not contain abstract methods.

Encapsulation is related to various use of modifiers i.e. private, protected and public. The concept of encapsulation can be understood as the process of wrapping code and data together into a single unit. The private members that are encapsulated can only be accessed through the getter and setter methods. Thus, a class can have control over the content in the fields. It also allows changing one part of the code without affecting the other part.

Inheritance is the way Java used to define a new class, and another class that share the same properties and methods of it. This is referred to as the parent-child (class and sub-class) relationship. The use of inheritance generally enhances the ability of reuse code as well as making the design process much simpler. However, it is important that Java does not support multiple inheritance by extending multiple classes. This can be achieved by implementation of polymorphism.

In OOP, polymorphism is used to represent a "is-a" relationship, thus allows multiple inheritance. The important concept of applying polymorphism is the use of overloading methods. In this project, all these concepts mentioned above would be implemented in various design perspective.

Note: contained in with this work is a detailed report concerning the design of the simulator developed by Group 7. Also submitted along with this report is a zip file containing the following:

- **User Guide.pdf** – pdf file containing instructions on how to set up, startup and run the simulator, make sure to read this before attempting to run any code;
- **Simulator _7.zip** – a zip file needed to run the code through the Eclipse IDE
- **Simulator_7.jar** – a jar file (Java achieve) to run the code through an OS shell
- **NumericalSolution_7 (MATLAB)** – a folder containing the MATLAB files that were used for the final simulator validation

2 Background

The report is aimed at detailing the design of a distillation simulator written in the JAVA programming language with a strong object-oriented frame of mind. The objective of this application is to allow users to determine the theoretical amount of trays within a distillation column that would be required given a series of problem parameters and constraints. From the theoretical number of trays, the total cost and profit of the distillation column will be determined and can be compared to other ventures to determine the most profitable one. This simulator contains a variety of object oriented programming aspects such as polymorphism, inheritance, interface, cloning, exception handling File I/O, etc. In the design of the simulator, the McCabe-Thiele method was used to determine the theoretical number of trays that are necessary to separate two components into their required mole fraction. This method uses a graphical approach to determine the number of trays by stepping between the operating lines (stripping and enriching) and the vapour-liquid equilibrium curve (Seider, Henley, & Roper, 2011). However, a traditional hand graphical method cannot be used explicitly to determine the number of trays with computer code; therefore, numerical methods will be used to produce the equilibrium curve, which will allow for the number of trays to be determined, in an equivalent way to a graphical method.

The enriching line is determined from the reflux ratio and the desired mole fraction of the distillate. The equation to calculate the enriching line is:

$$y = \left(\frac{R}{R+1}\right)x + \left(\frac{x_D}{R+1}\right) \tag{2.1}$$

Next the q line is calculated to determine the intersection point of the stripping and enriching line. The value of q, the feed quality, represents the mole fraction of liquid in the feed. First, q can be calculated using the following equation:

$$q = \frac{\Delta H^{vap} + C_P(T_B - T_F)}{\Delta H^{vap}} \tag{2.2}$$

Once the value of q is determined, the q line can be determined as follows:

$$y = \left(\frac{q}{q-1}\right)x - \left(\frac{x_F}{q-1}\right) \tag{2.3}$$

After the enriching and q line have been calculated, the equation of the stripping line can be determined. By equating Eq. 2.1 and 2.3 the intersection point of all three lines can be found. The mole fraction in the bottoms will determine the second point of the stripping line, which lies on the 45° line. The slope of the stripping line can then be calculated:

$$\text{slope} = \frac{\Delta y}{\Delta x} \qquad 2.4$$

Using the standard equation of a line, the y-intercept can be evaluated. These lines represent the operating lines of the system. To determine the number of trays using the McCabe-Thiele method, the vapour-liquid equilibrium curve needs to be found. The vapour-liquid equilibrium of a system is determined experimentally and is presented in as data point. To be able to step between the operating lines and the equilibrium curve, thus these data point need to be represented through functional form.

In the early stages of design, two types of algorithms to achieve this end were considered: polynomial regression and cubic spline interpolation. Although the former tends to provide smooth curvature, they do not ensure truly proper characterization of the equilibrium data simply because the solutions do not all pass through these points such as with spline interpolants. However, splines can sometimes provide too much curvature, which may be undesirable. Shape-preserving piecewise cubic Hermite interpolation (PCHIP) remedies this problem by reducing the second order derivative equality constraint at the node points thereby preserving the monotonicity and shape of the data (Fritsch & Carlson, 1980). This algorithm was chosen as the interpolation method of choice and its merits will be explained in the sections that follow.

Once the interpolant function is developed, the number of trays can be found. Starting at the bottoms of the stripping line, which represents the mole fraction of bottoms leaving the tower, the number of trays is determined by the number of steps taken between the equilibrium curve and operating lines. The stepping stops when the mole fraction of distillate leaving the top of the column is reached or surpassed. Usually the opposite case is practiced (starting form the distillate stream and enriching line and descending down); however, this would have required non-linear root solving algorithms to be additionally implemented, and as such was avoided in favor of more robust and sophisticated data interpolation techniques. However, it could also be argued that this methodology ensures a more conservative design due its stopping criteria being only reached once the required fraction is reached or surpassed.

3 Object-Oriented Design and Discussion

With the use of object-oriented programming, the design of this simulator mainly revolved around the object concept of a column and streams, which were broken down into separate classes with deep levels of inheritance representing the multiple and modularized components that make up process operations such as distillation. Due to abundance of OOP constructs present in this simulator, package structures were used to compartmentalize groups of similar or somehow related classes and interfaces. A unified modeling language diagram illustrating the package interactions and class hierarchy is shown on the next page in Figure A-3.

Some of the strides taken in the design which are reflected in nearly every package are:

- Encapsulation: instance variables of all classes are defined as `private` to ensure rigid encapsulation techniques, prevention of security leaks, and mandatory argument checking through the use of mutator methods. Methods were always declared as `public` unless such method acted solely as an intermediate step within an algorithm (e.g. the `pchipslopes` method in the `PchipInterpolator` class or the `calcFm` method in the `ModuleCosting` class);
- Versatility: although we used the `ExcelExtractor` class as the venture data storage intermediary and constructor argument for many classes, default constructors which made no assumptions of using such a unique object for setting instance variables of class instances.

3.1 `validation` Package

3.1.1 `Validate` Class

A number of Exception children classes, which are extensions of the parent Exception class, were created to handle various errors that can occur within the all the packages caused from common logic implemented For details on the exception classes, refer to Appendix A.

3.2 `numerical.analysis` Package

3.2.1 `Maths` Class

The Maths class implements various static methods useful for the various numerical package classes. These methods are specific mathematical functions that can be used in a variety of applications, but are not inherent to Java. Having these methods in a separate class allows for any of the other classes to implement them without having to repeat code every time a specific mathematical function is required in a similar way that the Java Math class is used.

3.2.2 `Function` *Interface*

The Function interface aims to design the data structure behind the concept of a function; given an x value, a y value can be calculated. The Function interface was implemented as an interface because it only requires two methods and has no instance variables. When other classes implement the Function interface, it ensures that a method to calculate a y value given an x is included.

3.2.3 `LinearFunction` *Class*

The LinearFunction class implements the Function interface to extend the concept of a function to lines specifically. It also incorporates various instance variables, which are inherent to linear functions – the slope and intercept. This class inherited method to determine a y value given an x, but it also includes a method to calculate the value of x given a y. For a linear function this is a simple calculation and is easily implemented. The LinearFunction class instantiates a method to calculate the intersection point of two linear functions. This is important when determining the number of trays in the distillation column. It is included in the LinearFunction class since it is a component of linear functions and not of a distillation column specifically.

3.2.4 `PolynomialFunction` *Class*

The `PolynomialFunction` class implements the Function interface to extend the concept of a function to polynomial functions. The additional instance variable included in a polynomial function are the coefficients, which are defined depending on the degree of the polynomial. The inherited methods from the Function interface are instantiated to calculate the value of y given an x. In the `PolynomialFunction` class a method to determine the value of x given an y is not included since this would required additional numerical methods to solve for x. In this simulator, the McCabe Thiele method starts counting the number of trays from the component fraction at the bottoms, the value of x of the equilibrium line is always provided and never needs to be solved.

3.2.5 `PolynomialSplineFunction` *Class*

The `PolynomialSplineFunction` class implements the Function interface to extend the concept of a function to polynomial spline functions. This class includes the instance variable that are necessary for splines; knots, polynomials of type `PolynomialFunction`, and the number of spline segments.

3.3 `numerical.interpolation` Package

3.3.1 `Interpolator` *Interface*

The Interpolator interface is used to define the interpolate method, which all the classes that implement the Interpolator function will use to implement their own interpolation algorithm. Its purpose is to serve as an intermediate between a class of Function descendent, interpolant fitting numerical algorithm, and supplied data point array as an interpolation solver. This allows other interpolation classes to be used, if desired, then the ones present in this simulator.

3.3.2 `PchipInterpolator` *Class*

The `PchipInterpolator` class implements the Interpolator interface. This class implements the shape-preserving cubic Hermite interpolant, which was heavily inspired by the MATLAB implement function 'pchip' (MathWorks, 2015). This class only implements the interpolate method from the Interpolator interface, which returns a `PolynomialSplineFunction` class type. This class is used in the `DistillationColumn` class to fit the equilibrium data to allow the McCabe Thiele method to be used.

Before this algorithm was selected, different studies were performed on the three venture equilibrium sets using 6-th order polynomial regression (poly6), traditional cubic spline interpolation (cspline), and finally shape-preserving PCHIP (pchip) for Venture 1, Venture 2, and Venture 3 equilibrium data sets with the resulting fits represented in Figure A.2 (a), (b), and (c), respectively, in Appendix A. Figure A.2 (d) is a scaled version of (c), representing the Venture 3 data set. Figure (e) below shows the trade-off between less curvature (pchip) and more oscillation (cspline), which may be critical to avoid in areas where marginal changes make big impacts (such as McCabe-Thiele's graphical method). Without Figure A.2 (d), the differences between these fits are nearly undistinguishable; however, the high order polynomial appears to stray at higher values than PCHIP and splines, whereas the spline fit seems to produce higher curvature at the knots. Thus, by selecting PCHIP, a middle ground of the more conservative splines is obtained while still maintaining 'visually pleasing' curvature typically seen with statistically regressed polynomials.

3.4 `process.unitoperation` Package

3.4.1 `OperatingLine` *Interface*

`OperatingLine` is presented as an interface here and include the methods `calcSlope` can `calcIntercept`, which would be initialized in the line classes. It is an interface here because

all the line classes require the function for calculation of their slopes and intercepts. Initially, `OperatingLine` was presented as a present class of all the line classes. However, it was concerned that the line classes also have the "is-a" relationship with the `LinearFunction`. `LinearFuction` is the child of `Function`, which already include the method for intercept calculation (`calcY`), and have the instance variables slop and intercept. Thus, it was determined that line classes would be children of the `LinearFunction` class and implements the `OperatingLine` interface. There child classes of the `LinearFunction` include all the lines required in the McCabe-Thiele methods for obtaining the number of trays of the distillation column.

3.4.2 `QLine` *Class*

The `Qline` class here is a child of `LinearFunction` and implements the `OperatingLine` interface. It is used for developing the q line used in the McCabe-Thiele method. It includes the methods for calculation of the slope and intercept, which are inherited from the parent class `LinearFunction`. These methods are overwritten in the child methods.

3.4.3 `EnrichingLine` *Class*

The `EnrichingLine` class is another child class of the LinearFunction and implements the `OperatingLine` interface. It is used for developing the enriching line used in the McCabe-Thiele method. It includes the methods for calculation of the slope and intercept, which are inherited from the parent class `LinearFunction`. These methods are overwritten in the child methods.

3.4.4 `StrippingLine` *Class*

The `StrippingLine` class is the child class of the LinearFunction and implements the `OperatingLine` interface. Just as the other two children classes of LinearFunction. It inherits the instance variables, slope and intercept, and the `calcMethods` from parent. Here, the `calcSlope` and `calcIntercept` are overwritten in the child class. All the line classes are later implemented in `DistillationColumn` for obtaining the number of trays required.

3.4.5 `Tray` *Class*

The Tray class is the parent class of `SieveTray`. The Tray class has two instance variables, material and diameter. These two instance variables are properties of all types of trays,

which allows the children to inherit these instance variables from the Tray class. The Tray class accesses these values from the `ExcelExtractor` class

3.4.6 `SieveTray` *Class*

The `SieveTray` class is the child class of Tray and therefore extends it. By having the name of the child class defined as the specific type of tray, this allows for the `ModuleCosting` class to accurately determine the bare module cost of the tray based on the type. In the simulator, the `SieveTray` class does require any additional information other than what it inherits from the Tray class.

3.4.7 `McCabeThiele` *Interface*

The `McCabeThiele` interface creates a method to calculate the number of trays. In this situation the McCabe-Thiele method was used to determined the number of trays in the distillation column; however, it is not the only method for calculated the number of trays. Having an interface define the method used to calculate the number of trays allows for other method to be easily implemented and used instead if desired.

3.4.8 `Column` *Class*

The `Column` class is the parent class to `DistillationColumn`. This class instantiates the instance variables that all columns include; diameter, length, gauge pressure, material, and streams. This class accesses the entered parameters for these instance variables from the `ExcelExtractor` class. in this class the number of streams that are entering or exiting the column are determined and the cost associated with each stream is also accessed from the `ExcelExtractor` class. In the `Column` class a method to calculate the flow rate of all the stream and the profit associated with the purchase and sale of the streams is implemented. These parameters are calculated in this class since they are based on the parameters of a general column and not the specific type. This allows for the `DistillationColumn` class, or any other type of column, to use these methods since they are not specific to a distillation column.

3.4.9 `DistillationColumn` *Class*

The `DistillationColumn` class is the child of `Column`, therefore extends it. It also implements the `McCabeThiele` interface to calculate the number of trays. The `DistillationColumn` class inherits the instance variable from `Column` that are general to all types of column. It also instantiates additional instance variables that are specific to a distillation

column; reflux ratio, trays, number of trays, and lines. The reflux ratio is accessed from the `ExcelExtractor` class. The information about the trays come from the Tray class, which uses polymorphism to use the information from the `SieveTray` class. The equation of the lines are accessed through the `EnrichingLine`, `Qline`, and `StrippingLine` classes. The line equations are implemented in the `DistillationColumn` class so the McCabe Thiele method can be used. The additional method that is implemented in this class is the `calcNumberOfTrays` method, since the number of trays in inherent to a distillation column. This method uses the component fraction from the bottoms stream as the starting point. It then calculates the y value on the equilibrium line, using the `PchipInterpolator` object, given then initial x value. Then, using the new value of y, the x value on the operating lines are determined using the line instance variables of type Line. This is repeated until the component fraction of the distillation stream is achieved. The `calcNumberOfTrays` method is assigned to the instance variable for number of trays. This allows for the `ModuleCosting` class to access the number of trays without having to enter all the input parameters required to calculate the number of trays.

3.5 `process.flow` Package

3.5.1 `Stream` *Class*

The Stream class is designed as an abstract class, because it represents a generic process stream that may be defined when it is extended to create children Streams that are specific process streams – feed stream, distillate stream, bottoms stream, and other streams that may be created that are out of the scope of this particular project. The abstract Stream class contains protected instance variables that are represent characteristics of any generic stream (flow rate, temperature, component fraction, etc.), as well as concrete accessor and mutator methods for the instance variables that will be inherited by its children classes. The instance variables are protected so that the children classes will have access to these instance variables, because each child will be required to assign its instance variables. The abstract Stream class also contains the two concrete methods `calcStreamBP` and `calcStreamCP`. These methods may be defined because the method for calculating stream heat capacity and boiling point will be the same regardless of what stream it is. The main reason that the Stream class is made abstract is because a generic Stream object will never be constructed, rather children stream objects will be constructed and used by other objects. The `Column` class expects Stream objects, and due to inheritance it may receive any of the Stream class' children, which maximizes the Stream class versatility.

3.5.2 `FeedStream` *Class*

The concrete child `FeedStream` class extends the abstract Stream class, and thus inherits all of the parent Stream class methods, as well as its protected instance variables. The `FeedStream` constructor method is used to assign the venture specific instance variables from the `ExcelExtractor` object. `FeedStream` also provides definition for the `calcQ` method, which is a method that calculates the feed quality, which is necessary for the `DistillationColumn` class.

3.5.3 `DistillateStream` *Class*

`DistillateStream` class is another child of the abstract Stream parent class, and inherits the parent Stream class' methods and protected instance variables. As with `FeedStream` class, instance variables are assigned, in the `DistillationStream` constructor. The feed quality method `calcQ` is set to return a dummy value of -9999. It was decided to include this method in all Stream classes because in other situations it is possible that the distillate stream will serve as a feed stream to another column, so in that situation it will require a `calcQ` method. Assigning a dummy value of -9999 indicates to any client that this is simply a place holder value, and to be ignored.

3.5.4 `BottomsStream` *Class*

The `BottomsStream` Class is the third child that implements the Stream abstract class, just like with the other child Stream classes, it too inherits all methods and protected instance variables, and assigns its instance variables using `ExcelExtractor` object within its constructor method. `BottomsStream` class also returns a dummy value of -9999 from the `calcQ` method with the same reasoning as `DistillationStream` class.

3.6 `costing` Package

3.6.1 `CostingException` *Class*

The `CostingException` class extends the `Exception` class and implements the `printStackTrace` method after calling its superclass constructor along with a custom message supplied via a `String` argument. This class is used to identify uncheck exceptions in the `ModuleCosting` class. Due to the many restricted bounds supported in the `ModuleCosting` algorithm, many runtime errors need to be verified and dealt with accordingly by throwing an

appropriate message corresponding to the location and situation (logic) for which it was triggered. This allows easy catching and handling when designing the program's main method.

3.6.2 `ModuleCosting` *Class*

The `ModuleCosting` class calculates all the parameters required to determine the total grassroots cost of the distillation column. All the equations and correlation used in this class were taken from Turton *et al*. (1998) and Ulrich (1984). Since these equations and correlations were developed in 1996 the cost index from the Chemical Engineering Plant Cost Index (CEPCI) for 1996 and 2014 were used as static final variables to update the cost for changing economic conditions. This class takes in all the instance variables required to calculate the cost from the `DistillationColumn` class. The simulator is structured as such because it allows for only an object of type `DistillationColumn` to be sent to the `ModuleCosting` class instead of individually sending all of the components that make up a distillation column.

3.7 `jexcel.excelextractor` Package

Disclaimer: the `jexcel` package takes advantage of the publicly available and open-source Java Excel API developed originally by Andy Khan (Khan, 2015). Although many efforts went into extending its application, Group 7 makes no claim of it being original work.

3.7.1 `Prompt` *Interface*

The Prompt interface was designed as an interface because it does not need any instance variables, and it is desired to keep its methods undefined, to be defined in whichever class that is to be implement it. Designing Prompt as an interface allows it to remain generic, as it is foreseeable that many different classes will require prompt methods, so it will be convenient to ensure classes such as `ExcelExtractor` will be required to give definition to these methods. It is essential that the `ExcelExtractor` class contains methods of retrieving the user specified file name to which file it will be opening and extracting venture data from. The implementation of the prompt interface ensures the provision of such methods. The Prompt interface contains two different prompt methods, one of which requires a String and integer as parameter list, while the other requires a Scanner object. The reason for these two methods are so that the client may either enter the parameters directly, or pass in a Scanner object from the main method.

3.7.2 `ExcelExtractor` *Class*

The `ExcelExtractor` class was designed to be the only class that would interact directly with the venture data file. It is essentially the gatekeeper for all of the relevant project data. Its role is to first prompt for and be passed the venture file name, validate if this file exists and if its contents are correct, then populate all of the class' instance variables with the information contained within the file. The `ExcelExtractor` class contains all of the instance variables that will be needed by other classes to complete their computations. The instance variables are all private to maintain their integrity, however the class provides accessor methods that may be used by all other classes to access their values. Arrays are deep copied within accessor methods to ensure there are no security leaks, and appropriate exception handling is performed inside accessor methods to ensure that the data is validated for type and bounds conformation before being assigned to an instance variable. The `ExcelExtractor` class implements the Prompt interface, and thus it is required to provide definition to the prompt methods. The prompt method that accepts a Scanner object will first prompt the user to input a file name in which the venture data is contained. The file name is then checked to see if it exists in a try catch loop. If the file does exist, the user is then prompted to enter the worksheet number. The worksheet is then validated in a try catch block, with an exception thrown if the worksheet provided is out of bounds. Upon validation of filename and sheet number, the filename String and sheet number integer are passed as input parameters into the `ExcelException` class methods `extractEq` and `extractVentureData`, which will then proceed to extract the venture data from the excel file, and assign them to instance variables (after being validated, as previously mentioned). The `ExcelExtractor` was designed as a class so that an `ExcelExtractor` object can be constructed, and passed into other objects so that any object may have direct access to all of the venture data, as provided through the `ExcelExtractor` accessor methods.

3.7.3 Exception Classes

A number of Exception children classes, which are extensions of the parent Exception class, were created to handle various errors that can occur within the `ExcelExtractor`, since it is in this class that the venture data file is opened, and data is extracted.

- `InvalidConsoleEntryException`: subclass of `IOException` class and is to be thrown if the user inputs an invalid data type i.e. a String is entered when an integer is expected, such as when the user is prompted to enter the worksheet number.

- `InvalidExcelCellException`: subclass of `InvalidArgumentException` class and is to be thrown if the excel file contains unexpected data type, i.e. a String when an integer is expected, such as if a String is contained in the file instead of the equilibrium numerical values.
- `InvalidExcelFileException`: subclass of `FileNotFoundException` and is to be thrown if the user enters an invalid text file name, i.e. the .xls data type is omitted.
- `InvalidExcelSheetException`: subclass of `IOException` class and is to be thrown if the user enters a sheet number that is outside of the range of worksheets contained in their venture workbook i.e. for a worksheet containing 3 ventures, if the user enters 4, this exception would be thrown.

3.8 `jexcel.excelwriter` Package

3.8.1 `ExcelWriter` *Class*

The `ExcelWriter` class is used only during execution of the `Main's main` method. Its sole purpose is to serve as file output (`ExcelWriter` being the counterpart file input class).

3.9 `bootstrap` Package

3.9.1 `Main` *Class*

The `Main` class is the control center of this simulator where the `main` method operates as the forefront between user interaction and program functionality. For this reason, as with any other well designed application entry point, all exception handling should be dealt with dynamically while running. Evidently, upon startup, the main method will prompt the user to enter an Excel file were the venture data is located. Following this, assuming no exceptions are caught, the sheet number is inquired about, and then the output is generated in a short summary on screen with an additional prompt asking if the user wishes to output the detailed simulation result to an Excel file of his or her choosing. If so, the user must enter the name of the output Excel file, and it will append it to a new sheet at the end of the document or to a new file entirely if the file name entered does not exist. Afterwards, the user is ushered to a prompt asking whether he or she would like to return to the main menu and analyze a new venture. If so, the cycle is repeated; if not, the user is given a farewell message and the application terminates.

4 Validations

Validation is a crucial part of this project. This process ensures that the designed program meet the requirements – to obtain the number of trays of a distillation column and compare the results of the three ventures available. All the methods created in the presented simulator were tested with ranges of appropriate and extreme values, respectively. The purpose of this is to ensure that the input being passed to the simulator comply with design assumptions and limitations, and that if not, the program is able to catch and handle such expected exceptions appropriately without producing any inadvertent terminations. The outcome of these validations are presented in test case matrices, which compare the output values from the designed simulator to those obtained with MATLAB/Excel. Due to the complexity and length of the code and hence amount of testing required, the test case matrices for each package were added to Appendix D and relate mostly to Venture 1 data for cases of valid input/output. Discussion relating to these are, however, detailed in the sections that follow.

4.1 `validation` Package

Throughout this section, validity of the exceptions classes within this package will be exemplified through the outcome of the test cases used by every other classes were exceptions should be thrown. It is important to note that, except for the bootstrap package's `Main` class, the unit testing (individual classes tested separately) does not aim to catch and handle exceptions. In contrast, the `Main` class' `main` method will do just that since it is the intermediary between user input and simulator output. This allows easy debugging and catching of anticipated exceptions for later use in when designing the main method where the handling actually occurs thanks to the distinguishable throws that narrow range of unexpected outcomes. As seen in all the validation tables of Appendix D, any exceptions thrown are the custom defined extensions designed for this simulator as the result of either the methods logic or through the help of utility methods such as the validate class' `checkNotNull`, `checkNotEmpty` static methods which further make use of this class' `isNotNull`, `isLargerThan1`, `isNonNegative`, and `isType` static utility methods.

4.2 `numerical.analysis` Package

Table D-1 to Table D-5 show the test matrices for the methods of the LinearFunction, `PolynomialFunction`, `PolynomialSplineFunction` classes and `Function` interface. As expected, all methods throw one of the custom exception classes defined in the `validation` package. Notable discrepancies observed between the simulators output and the MATLAB values

tested with are in the cases of methods `calcIntersection` of LinearFunction class (Table D-1), diff of Maths class (Table D-2), and `calcY` of `PolynomialSplineFunction` class (Table D-5), When two `LinearFunction` objects with the same `Slope` field value are given as arguments to `calcIntersection`, an `InvalidArgumentException` is thrown displaying the cause of error; however, in MATLAB non-standard arithmetic are classified have their associated data types, such as `NaN` (not a number), `Inf` (infinity), etc., hence why [`NaN`, `NaN`] was obtained from division by zero. The diff method returns a 1x3 array in Java but a 1x2 array in MATLAB, which its implementation was heavily relied on. This is expected also for the intended purposes here; this was not required since the dropped dimension (last element) was never used subsequently (i.e. this was a design intent) and resizing its array (creating a newly sized one) like MATLAB's version would require ensuring unnecessary heap management and the like, thus it was omitted. Finally, in MATLAB's polynomial spline algorithms, it is permitted to use the structures generated (structure being a data constructs holding various data types similar to user defined classes) and extrapolate the polynomials outside the range defined by the exterior knots (min and max data points of dependent variable array x). This is however very dangerous since an interpolation method is never guaranteed to extend beyond this range. For this reason, the method was implemented to restrict this possibility and instead throw an `OutOfRangeArgumentException` when catching such invalid argument inside its logic.

4.3 `numerical.interpolation` Package

Table D-6 shows the validation of `PchipInterpolator`'s `interpolate` method. As expected, all exceptions thrown were excepted. Notably, the exceptions noticed the two special cases of dimension mismatch (x and y arrays of unequal sizes sent as arguments) and insufficient data points supplied (when both x and/or y arrays are less than 3 double elements).

4.4 `process.unitoperation` Package

4.4.1 Slopes and Intercepts: `calcSlope` *and* `calcIntercept`

The calculation for the slopes and the intercepts for the line, q line, and stripping line is vital for obtaining number of trays in the column. These two methods are incorporated in the line classes: `Strippingline`, `Qline`, and `EnrichingLine` classes, respectively. The only parameter passed to the slope method and intercept method is the object of column, which would be able to call the required parameters using the getters methods. The values required for the

stripping line methods are the mole fraction of the stream and the intersection of the q line and enriching. For the q line, the q value and mole fraction of the feed stream are required; and for the enriching line, mole fraction of the feed stream and the reflux ratio are required. To test these values, exceptions are incorporated in each of the methods. As seen in Table D-19, the parameters values are varied to out of bound, as well as incorrected data type. For all of the tested parameters, when the mole fractions are out of bound (any numbers other than within the range 0 to 1), the out of bound exception is called and the user is asked to input a correct number. When negative numbers are input, the same exception would be thrown. This make sure that the proper and reasonable values are being passed to the column for tray calculation. When the values input are not the correct data, from Table D-19 it can be seen that an exception would be thrown by the system and ask the user for correct data type input. This ensure that even incorrect data type is passed to the column, the system would not shut down immediately.

4.4.2 Number of Trays: `calcNumberOfTrays`

The calculation of the number of trays is important because it establishes the final purity of the distillate and bottoms streams. The two main parameters that are used to calculate the number of trays is the equilibrium data and the lines. The equilibrium data is fitted with the `PchipInterpolator` class. The other two parameters that are used to calculate the number of trays are the distillate and bottoms component fraction, which indicate the points on the operating lines of where to stop and start stepping, respectively. The equilibrium data, and component fractions were varied to determine the limits of the number of trays calculation, as seen in Table D-9. The equilibrium data and the component fractions were varied to out of bounds limits as well and incorrect data types. For both parameters, when an out of bounds value was present, the simulator would catch the exception and proceed to output an message stating that the value was output bounds and required to be in a specific range; 0 to 1 for both parameters. This ensures that unrealistic values for equilibrium data are present as well as for the component fraction. When an incorrect data type is entered the simulator also catches this exception asking the user to enter the correct data type. All of the different types of exceptions are caught, allowing for the simulator to run without crashing.

4.4.3 Distillate and Bottoms Molar Flow Rates: `calcFlowRates`

To parameters that are used to calculate the distillate and bottoms molar flow rates, in the `calcFlowRates` method, are the feed flow rate, and the feed, distillate, and bottoms component

fraction. All of these values were tested within the appropriate range, at out of bounds values, and incorrect data types. The variation in the parameters and the output from the simulator compared to the expected output is presented in Table D-7 in the Appendix. When all the parameters were entered in the appropriate range the simulator produced the desired outcome. When the feed flow rate was varied to test a much larger values, the simulator was able to calculate the appropriate flow rates. This was expected since the distillate and bottoms flow rates increased proportionally to the increase in the feed flow rate. The lower limits of the simulator for the feed flow rate is 0. This limit was developed since a negative feed flow rate is not a realistic phenomenon. By entered a negative flow rate the simulator was able to catch this exception and output that the flow rate need to be positive to determine the flow rates of the distillate and bottoms. Mathematically, the equation to calculate the distillate and bottoms flow rate can handle negative values and therefore the hand solution was able to output negative flow rates. When an incorrect data type was entered the simulator was able to catch this exception and output that a numeric value is required. The limits of the component fraction for all the streams is from 0 to 1 since values outside this range do not make physical sense. By changing the component fraction of the feed, distillate, and bottoms streams to outside the limits and to a incorrect data type the simulator was able to catch all of these exception and output a string stating that the component fraction needs to be in the accepted limits and of the correct data type, respectively. Mathematically, entering a value for component fraction outside of the accepted range values of the distillate and bottoms streams can be calculated, but one of the streams will be negative, which does not make physical sense. This validation shows that the `calcFlowRates` method in the `Column` class is able to catch any exceptions that are incorrectly entered.

4.4.4 Profit: `calcProfit`

The calculation of the profit, using the `calcProfit` method, is done in the `DistillationColumn` class. The profit is calculated using the flow rate of all the streams and their respective costs. The validation of this method is presented in Table D-8. Changing the cost of the streams will directly affect the profit, but any numerical number above 0 is valid for the cost of any of the stream. If a negative number is entered for the cost than the simulator will prompt the user to enter a non negative value. When a non numeric value is entered for the cost of any of the streams, the simulator will catch this exception and output to the user that the wrong data type was entered. This ensures that the simulator will run correctly without crashing. The flow rates

of the streams were also tested. As with the cost of the stream, mathematically, the stream flow rates can be any numeric value, however, if a negative value is entered into the simulator than an exception message will appear stating that a non negative value is required for the flow rates. This ensures that the profit is being accurately calculated. Again, non numeric values are properly caught and dealt with.

4.5 `process.flow` Package

All stream classes (feed, distillate, and bottoms) contain methods to calculate the stream boiling point temperature and the stream heat capacity. These values are used by other objects throughout the program, so it is necessary that they make physical sense or else no meaningful solution may be calculated. The input parameters to the `calcStreamBP` *and* `calcStreamCP` methods are the component mole fractions, component normal boiling points, and component heat capacities. Exception handling is implemented to ensure that no negative component boiling points or heat capacities, nor invalid data types such as Strings may be input to these methods. Exception handling is also carried out to ensure that only numeric data type mole fractions in the range of between 0 and 1 may be input as parameters to these methods. Validation results for the methods `calcStreamBP` and `calcStreamCP` for all three Stream classes are summarized in Table D-12 to Table D-18. Invalid data types were entered along with out of bound numeric values, and hand solved solutions were compared to Java output values. All cases were handled with the appropriate `InvalidExcelCellException`, which specified the parameter that was invalid, as well as the required data type and range. Although hand solutions were able to be calculated, they had no physical meaning when a negative component boiling point or heat capacity were entered, and the program was able to recognize this and throw an exception.

The `FeedStream` class had an additional method to calculate the feed quality q in a method called `calcQ`. This method takes in the returned values from the `calcStreamCP` and `calcStreamBP` methods, which had already been checked for data type and range for component heat capacity, boiling points, and mole fractions. An additional input parameter is required for `calcQ`, which is the latent heat. Validation for invalid data type (String) and out of bounds (negative) was conducted (Table D-14), and all scenarios were handled by throwing the correct `InvalidExcelCellException`, and notifying the user of their error, and indicating the necessary data type and range for latent heat.

4.6 `costing` Package

4.6.1 *Purchase Cost:* `calcCapitalPurchase`

The `calcCapitalPurchase` method in the ModuleCosting class was validated to test the limits of the methods, shown in Table D-22. Based on the equations used for the calculation of the capiltal purchase cost from Turton (1998), the diameter of the column, the parameter used in the `calcCapitalPurchase` method must be between 0 and 4. When high and low out of bounds values were used in this methods, the simulator was able to catch these exceptions, using the `CostingException` class, and output to the user that the diameter is required to be within the correct range. This ensures that the equations are only used for the correct range to ensure that the capital purchase cost is calculated correctly.

4.6.2 Bare Module Cost: `calcBareModule`

The `calcBareModule` method was used for both the column and the trays to determine the individual bare module cost for these components. The parameters that are used in the `calcBareModule` method for the column are the capital purchase cost, and the material and pressure factors, seen in Table D-25. The parameters for the trays are the number of trays and material of construction, seen in Table D-26. For the column, mathematically, any numeric value can be entered into the method and the bare module cost will be determined. It is the method for calculating the material and pressure factors that will catch these exceptions, which are shown in Table D-23 and Table D-24. For the `calcBareModule` method for the trays, the number of equations used limit the number of trays to be greater than 0 and to have only three types of materials. When validating the limits of this methods, the simulator was able to catch these exceptions and prompt the user to enter a value within the correct limits. Again, only values within the limits of the equations will be used ensuring that the bare module costs are calculated accurately.

4.6.3 Grassroots Cost: `calcGrassRoot`

The `calcGrassRoot` method was used for both the column and trays individually. The grass roots cost is calculated using the bare module costs of the column and trays, for which the validation is shown in Table D-27 and Table D-28, respectively. The calculation of the grass roots is a basic mathematical equation so any numeric value can be entered for the bare module cost and a grass roots cost will be determined. These methods do not have specific exception handling, but

rely on the exceptions being caught in the methods leading up to this point. Since the grass roots cost cannot be calculated with the previous costs being calculated first, there was no need to add exception handling specifically in this case.

4.6.4 Total Cost: `calcGrassRootTotal`

The `calcGrassRootTotal` method uses the grass roots cost for the column and trays and sums these values together. As for the `calcGrassRoot` methods, any numeric value can be entered into this method and a total grass roots cost will be calculated, as seen in Table D-29. Again, this method relies on the previous methods to catch any exceptions that are entered into the simulator and since this method is dependant on all the previous methods in the `costing` package there is no need for additional exception handling at this point.

4.7 `jexcel.excelextractor` Package

4.7.1 Prompt Excel Data: *`Prompt`* `and` *`extractVentureData`*

The `prompt` method is the way in which a user will specify which excel file and worksheet number they would like to have analyzed. The file name and worksheet number are specified as a String and integer that are input through use of a Scanner object as input parameter into the *prompt* method. It is important that both file name String and worksheet integer are checked for data type, range, and existence before being passed as parameters into other methods in the `ExcelExtractor` class that will extract the venture data, or else the program will encounter an error when it tries to open a file that does not exist. Appropriate exception handing was implemented in the *prompt* method to handle all foreseeable situations, and validation testing of these scenarios was carried out and is summarized in Table D-35. The tested file name scenarios were invalid file name entry (correct String data type entered, but not specifying an existing file) and wrong data type (a number type entered instead of a String). For the sheet number, the test cases included a work sheet number that was outside of the range contained within the excel workbook (i.e. if the user entered sheet number 999 when the workbook only contained 3 venture worksheets) and invalid data type such as a String instead of a number. These scenarios were seen to represent realistic errors that a user may make when entering the name of a file or worksheet number. The exceptions were all caught with the customized thrown exception `InvalidExcelFileException`. During program execution, if the user enters either an invalid venture file name they will be told they have done so, and asked if they would like to return to the

main menu or not. Comparably, if the user has input an invalid worksheet number, they will be told, and given the same main menu prompt.

4.7.2 Extract Equilibrium Data: `extractEq`

Once the excel file and worksheet number have been checked for validity, they are passed in as input parameters into the `extractEq` method, which will extract the equilibrium data from the user specified venture worksheet. It is imperative that the equilibrium data is checked for correct data type before being imported into the class and assigned to instance variable arrays, because the equilibrium data is fundamental for calculating the number of trays in the `DistillationColumn` class – if the data is either the wrong data type or out of bounds, the program will not be able to run. Inside of the `extractEq` method, both liquid and vapour equilibrium data are first checked to be of number data type before being assigned to an array, with an `InvalidExcelCellException` thrown otherwise. After data type checking, the range of the equilibrium data is checked inside the mutator methods to be between 0 and 1, with an `OutOfRangeArgumentException` thrown if any number is out of bounds. If both data type and range satisfy the specified conditions, the extracted liquid and vapour equilibrium data are assigned to double arrays `eqX[]` and `eqY[]`. Validation was conducted by entering String and out of bound numbers into the excel spreadsheet, and seeing if the custom exceptions were able to handle these test cases. Results for this validation are summarized in Table D-30 to, all cases were handled with the appropriate exception being thrown.

4.7.3 Extract Venture Data: `extractVentureData`

Identical to the `extractEq` method mentioned above, the `extractVentureData` method also takes in the file name String and worksheet integer that have been validated in the *prompt* method to specify which excel file and worksheet contains the venture data to be extracted and assigned to the appropriate class instance variables. Once again, before assignment to instance variables, the data type and bounds must be checked because these instance variables will be used by several other project classes and methods, thus they must be both the appropriate data type and within a specific range. Validation was completed, for the `extractVentureData` method, and are summarized in Table D-36 to Table D-40. All invalid data type and out of bound entries were caught with `InvalidExcelCellException`. This exception also states explicitly which parameter is wrong, and what the necessary data type or range should be for successful venture analysis.

4.8 `jexcel.excelwriter` Package

`ExcelWriter`'s sole use is through the `bootstrap`'s `Main` `main` method. For this reason, validation for both these classes are combined in the next section.

4.9 `bootstrap` Package

`4.9.1` *Application entry point:* `main`

Because the entry point of the main method usually entails many different exceptions and variations in ways the sequence of actions can generate different outcome, an activity UML is presented on the following page in the hope to be succinct enough to represent all test case outcomes achievable in conducting validation.

This activity UML diagram outlines the states of exception catching and handling conducted through sequences of various user input. This validation also includes the final output summary of the MATLAB designed code used for this validation, shown below in Figure 4-1.

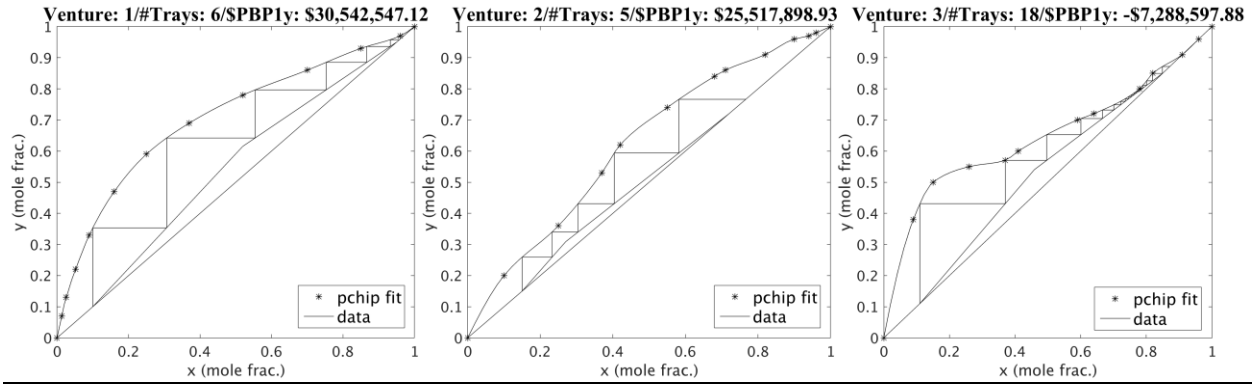

Figure 4-1. Output results generated from the MATLAB code (seen in Appendix C) used for validating the JAVA code's overall results.

The results seen from the above figure are consistent with the output generated from venture 1, 2, 3 simulation tests. The detailed JAVA simulator results may be seen in more detail in Sheets 3 to 6 in the Excel file titled "Venture.xls" provided with this report or by running the simulator and generating the exports to a new excel sheet.

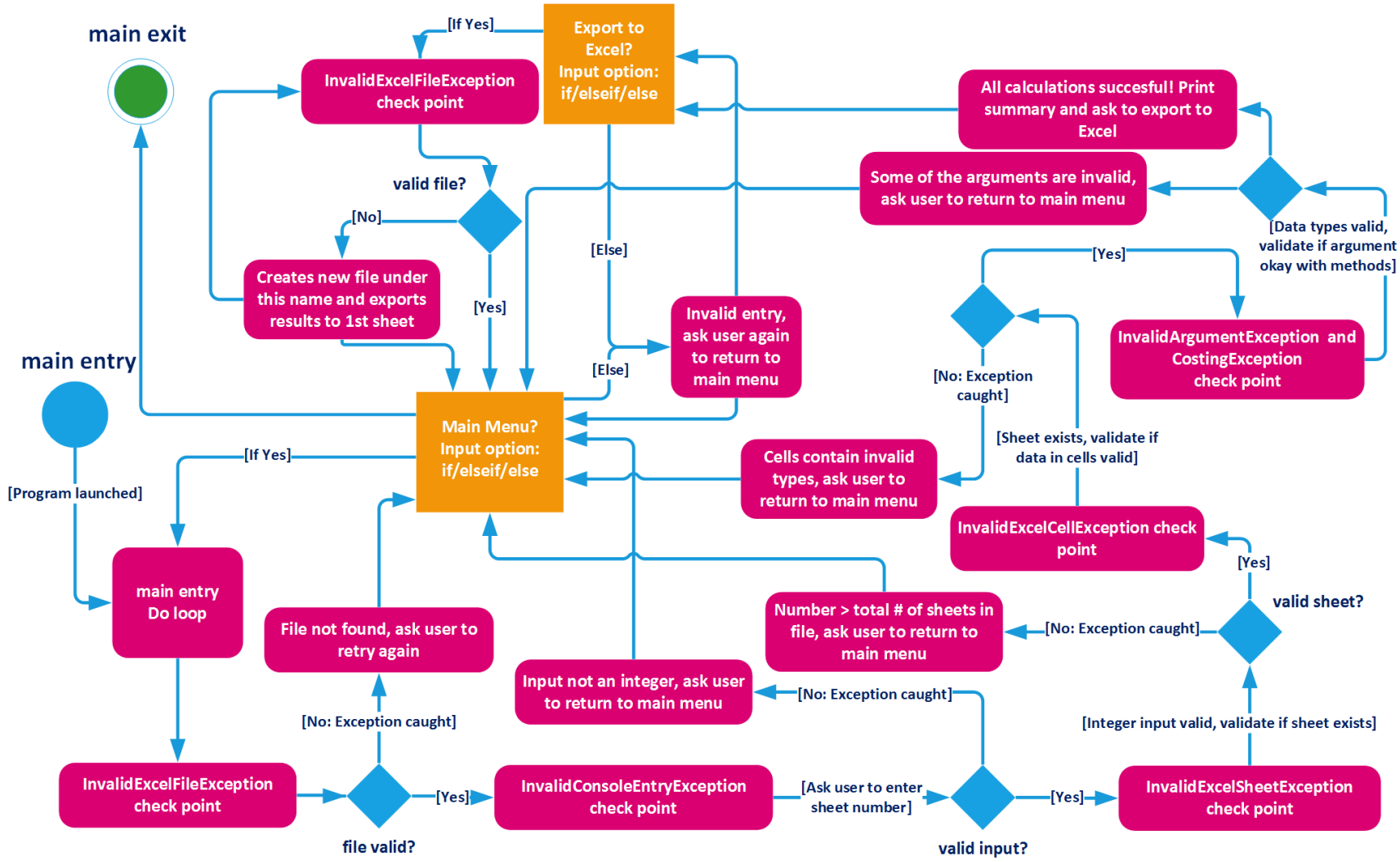

Figure 4.2. Activity UML diagram outlining the states of exception catching and handling conducted through sequences of various user input.

5 Improvements and Extensions

Over the course of the design stages, a few key mentions may be made regarding how this simulator could be further improved and/or extended:

a) The incorporation of a degree of freedom solver, similar to more sophisticated software, would permit the user to enter non pre-established values into the Excel file. For example, having a DOF solver, the algorithm would allow the user to enter any number of variables as long as the system had as many equations as unknowns. Particularly, mandating that the feed stream flow rate be supplied, instead of any one of the three streams, for example, was a limitation of not having a more rigorous code structure. Since the `Column` class has `Stream` array which themselves have `flowRate` properties, usually N-1 variables are specified, and 1 must be solved. However, due to the nature of the problem, having an array of streams with some always set and some which depend on the existence of the others inside that object makes one of the streams to have a very distinct set of methods (which aren't independent of others like this one – i.e. the `StrippingLine` class' `calcSlope` and `calcIntersept` methods are hard coded to be depended on `EnrichingLine` and `QLine` having been set previously, else the methods will fail).
b) In the JAVA programming language, the notion of dependent variables is not as common as other languages such as MATLAB. When a class variable is not directly set but must be instead calculated based on other known values of its instance variables, JAVA tends to support the practice of having these implemented as 'calc' methods and omit creating them as instance variables. This is often disadvantageous because when a class truly holds a "has a" relationship with variable property, the lack of standardizing a set method is tricky and thus always necessitates the use of 'calc' methods to both either 'get' or 'set' the values. In MATLAB, however, when designing OOP constructs such as classes, a developer has the option to define its class properties (instance variables) as dependent. This allows them to define standardized setters and getters that function truly as its own class instance variables but without the need for discerning between a 'calc' or get/set method. This is done by calling the get method of the object and if MATLAB realizes that the property in question is dependent, it executes the method's algorithm and calls upon any other of its properties necessary for instant computation (similar to a 'calc' method); however, the difference is as soon as the call is finished, MATLAB

enters the properties' hidden set method to effectively set its property to its new value as just calculated. This is a limitation of the programming language and its common practices.

c) During the design stages, the possibility arose for implementing all three categories of curve fitting algorithms in the final code. Even though these were at some stage feasible, since all the different classes functioned and calculated expected results, due to the shear volume of classes and dedication required to maintain such classes and interfaces, namely the upkeeping of validation and debugging was overzealous for the scope of this design. For this reason, of the three (polynomial regression, cubic splines, and pchip), analyses were conducted into the best candidate and pchip was chosen.

d) Due to the overwhelming workload associated with the final simulator, attempts at developing a graphical user interface (GUI) were envisioned but never realized. This again is simply due to the time constraints and need for prioritization over the algorithms and code interplay over user input dressings. Although GUIs are nice, fully functional and well developed code is nicer.

6 Conclusions

The most profitable venture for 2PS was determined using the developed simulator, which was designed using a JAVA program. By providing certain characteristics of the distillation column the program is able to generate the required number of trays, the flow rate of the distillate and bottoms, the purchase cost, bare module cost, total module cost, grassroots cost and yearly profit of the venture. The validations of the designed simulator, presented in section 4, confirm that the calculated results are accurate and satisfy the design requirements sought after the client, 2PS. Figure 6.1 presents a summary of the grassroots cost, the profit based on production for a single year, and the total profit over a 1-year payback period (PBP) for three ventures that 2PS is considering.

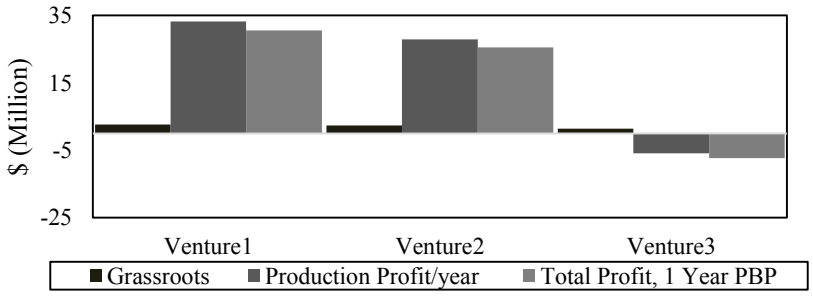

Figure 6.1. Summary of the grassroots cost, the profit based on production for a single year, and the total profit of the venture over a 1-year payback period.

It can be seen that Venture 1 produces the highest total profit over a 1-year payback period of $30,542,547.12 with a tray requirement of 6. Although Venture 2 was similar with a total profit over a 1-year payback period of $25,517,898.93 and a tray requirement of 5, Venture 3 resulted in a deficit. Since Venture 1 has the largest grassroots cost, even over a longer payback period, Venture 1 would remain the most profitable.

Although many of the specific implementations of this simulator were simplified due to the simple set of assumptions brought forward by the methodology used, such as the use of the McCabe Thiele method, many of the data constructs were designed with versatility in mind. Notably, the extensive level of abstraction and modularity brought forward by deep a class hierarchy and multiple levels of object composition, which are akin to OOP, are simply to name a few. These would allow easy reuse of the designed OOP constructs presented in this report (classes, interfaces, etc.) and perhaps incorporate them into a more complex and multi-faceted simulator such as those seen in industrial setting such as Honeywell's UniSim and Aspen's Hysys.

Appendices

A Additional Design and Discussion

A.1 `Validate` Class

A number of Exception children classes, which are extensions of the parent Exception class, were created to handle various errors that can occur within the all the packages caused from common logic implemented such as in `Validate`'s `checkNotNull`, `checkNonEmpty`, `isNonNegative`, and other methods.

- `InvalidArgumentException`: extends Java's native runtime exception class `IllegalArgumentException` and is the superclass of the following subclasses.
- `EmptyArrayArgumentException`: thrown when array has length of zero.
- `NullArgumentException`: thrown when argument is null type.
- `OutOfRangeArgumentException`: thrown when argument outside bounds.

A.2 UML

The UML is presented on the following page in Figure A-1.

A.3 PchipImterpolator Class

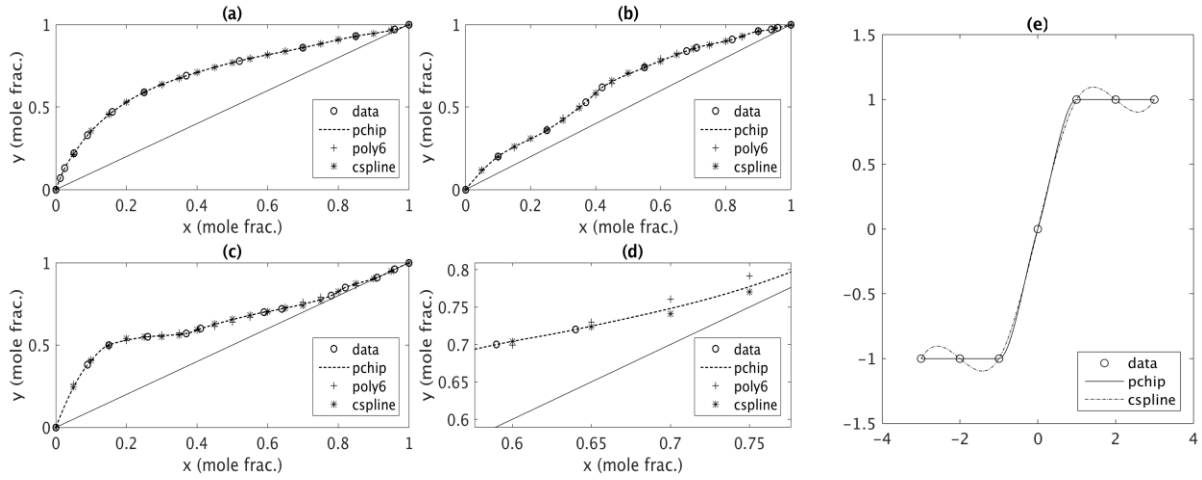

Figure A.2. Polynomial regression and interpolation fits for (a) Venture 1, (b) Venture 2, (c) Venture 3, and (d) scaled Venture 3 equilibrium data sets, and (e) representative comparison of interpolants.

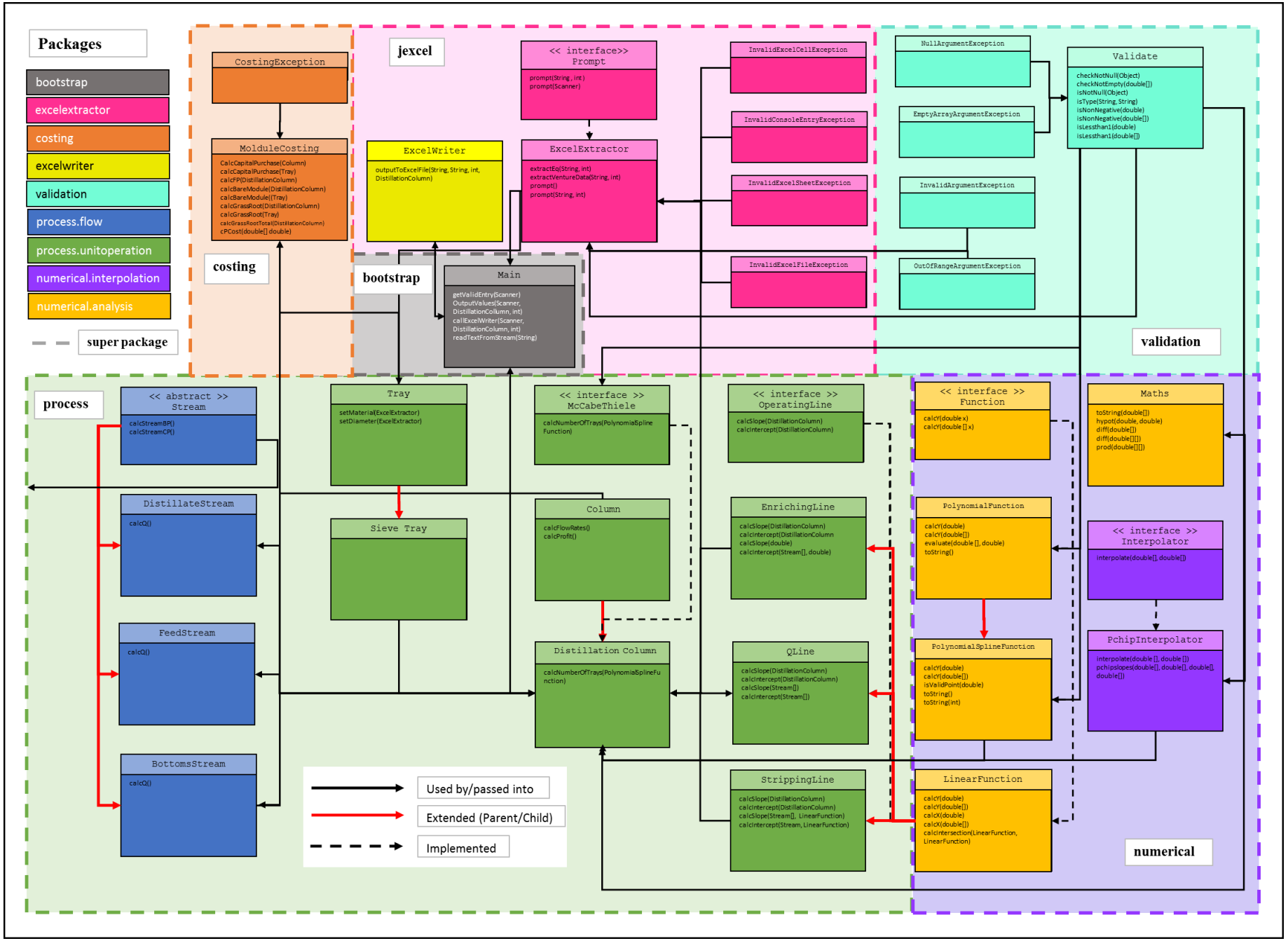

Figure A-3. Unified Modeling Language (UML) Diagram illustrating the package and class hierarchy.

B Java Code

B.1 bootstrap: Main.java

```
package bootstrap;

import java.text.DecimalFormat;
import process.unitoperation.*;

import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;

import jxl.read.biff.BiffException;
import jxl.write.WriteException;

import java.util.Scanner;

import costing.CostingException;
import costing.ModuleCosting;
import jexcel.excelextractor.*;
import jexcel.excelwriter.ExcelWriter;

public class Main {

	public static void main(String [] args) throws BiffException, IOException,
	WriteException, FileNotFoundException, CostingException {

		ExcelExtractor ee = new ExcelExtractor();
		boolean redoMain = false;
		String answer = "";
		Scanner userInput = new Scanner(System.in);
		int sheetNum;

		do {
			readTextFromStream("Resources/Image4.txt");
			readTextFromStream("Resources/Title2.txt");

			try {
				sheetNum = ee.prompt(userInput);
				DistillationColumn column = new DistillationColumn(ee);
				OutputValues(userInput, column, sheetNum);
				System.out.println("\n-----------------------------------"
						+ "--------------------");
				System.out.println("\tANALYZE A NEW VENTURE?");
				System.out.println("-----------------------------------"
						+ "--------------------");
				System.out.println("Would you like to analyze another
					venture?");
				System.out.print("(Y)es or (N)o: ");
				answer = userInput.nextLine();
				System.out.println("-----------------------------------"
						+ "--------------------");
				if (answer.equalsIgnoreCase("y"))
					redoMain = true;
				else if (answer.equalsIgnoreCase("n")) {
					redoMain = false;
				} else redoMain = getValidEntry(userInput);
			} catch (FileNotFoundException e) {
				System.out.println("\n-----------------------------------"
						+ "--------------------");
				System.out.println("\tINVALID FILE!");
				System.out.println("-----------------------------------"
						+ "--------------------");
				System.out.println("That file was incorrectly entered or
					does "
					+ "not exist! \nWould you like to try again?");
```

```
                System.out.print("(Y)es or (N)o: ");
                answer = userInput.nextLine();
                System.out.println("-----------------------------------"
                                + "--------------------");
                if (answer.equalsIgnoreCase("y"))
                        redoMain = true;
                else if (answer.equalsIgnoreCase("n")) {
                        redoMain = false;
                } else redoMain = getValidEntry(userInput);
        } catch (InvalidExcelSheetException e) {
                System.out.println("\n-------------------------------------"
                                + "--------------------");
                System.out.println("\tSHEET NUMBER TOO LARGE!");
                System.out.println("-------------------------------------"
                                + "--------------------");
                System.out.println("Sheet number is larger than the total
                                        sheets "
                                + "this file contains!"
                                + "\nWould you like to return to "
                                + " the main menu?");
                System.out.print("(Y)es or (N)o: ");
                answer = userInput.nextLine();
                System.out.println("-------------------------------------"
                                + "--------------------");
                if (answer.equalsIgnoreCase("y"))
                        redoMain = true;
                else if (answer.equalsIgnoreCase("n")) {
                        redoMain = false;
                } else redoMain = getValidEntry(userInput);
        } catch (InvalidConsoleEntryException e) {
                System.out.println("\n-------------------------------------"
                                + "--------------------");
                System.out.println("\tINVALID INPUT!");
                System.out.println("-------------------------------------"
                                + "--------------------");
                System.out.println("Sheet number entered is invalid! Make
                                        sure "
                                + "you enter an integer value! ");
                System.out.println("\nWould you like to return to "
                                + "the main menu?");
                System.out.print("(Y)es or (N)o: ");
                answer = userInput.nextLine();
                System.out.println("-------------------------------------"
                                + "--------------------");
                if (answer.equalsIgnoreCase("y"))
                        redoMain = true;
                else if (answer.equalsIgnoreCase("n")) {
                        redoMain = false;
                } else redoMain = getValidEntry(userInput);
        } catch (InvalidExcelCellException e) {
                System.out.println("-------------------------------------"
                                + "--------------------");
                System.out.println("\tINVALID SHEET NUMBER");
                System.out.println("-------------------------------------"
                                + "--------------------");
                System.out.println("Sheet number entered appears to contain
                                + "invalid data! \nPlease refer to the user
                                        manual to "
                                + "ensure proper placement of requirement
                                        data! \nWould "
                                + "you like to return to the main menu?");
                System.out.print("(Y)es or (N)o: ");
                answer = userInput.nextLine();
                if (answer.equalsIgnoreCase("y"))
                        redoMain = true;
                else if (answer.equalsIgnoreCase("n")) {
                        redoMain = false;
                } else redoMain = getValidEntry(userInput);
        } catch (BiffException  e) {
                System.out.println("\n-------------------------------------"
                                + "--------------------");
```

```
                        System.out.println("\tBROKEN OR CORRUPT SHEET!");
                        System.out.println("-------------------------------------"
                                + "--------------------");
                        System.out.println("File appears to be broken or corrupt
                                or"
                                + "sheet does not exist. Please try another file!");
                        System.out.println("\nWould you like to return to "
                                + "the main menu?");
                        System.out.print("(Y)es or (N)o: ");
                        answer = userInput.nextLine();
                        System.out.println("-------------------------------------"
                                + "--------------------");
                        if (answer.equalsIgnoreCase("y"))
                                redoMain = true;
                        else if (answer.equalsIgnoreCase("n")) {
                                redoMain = false;
                        } else redoMain = getValidEntry(userInput);
                } catch (CostingException e) {
                        System.out.println("\n-------------------------------------"
                                + "--------------------");
                        System.out.println("\tINVALID VENTURE DATA FOR COSTING
                                CORRELATIONS!");
                        System.out.println("-------------------------------------"
                                + "--------------------");
                        System.out.println(e.getMessage());
                        System.out.println("\nWould "
                                + "you like to return to the main menu?");
                        System.out.print("(Y)es or (N)o: ");
                        answer = userInput.nextLine();
                        System.out.println("-------------------------------------"
                                + "--------------------");
                        if (answer.equalsIgnoreCase("y"))
                                redoMain = true;
                        else if (answer.equalsIgnoreCase("n")) {
                                redoMain = false;
                        }
                }
        } while (redoMain);
        readTextFromStream("Resources/Sad.txt");
        readTextFromStream("Resources/Title.txt");
}

private static boolean getValidEntry(Scanner userInput) {
        boolean validEntry = false;
        boolean redoMain = false;
        String answer = "";
        do {
                System.out.println("\n-----------------------------------------"
                                + "--------------------");
                System.out.println("\tINVALID ENTRY!");
                System.out.println("-----------------------------------------"
                                + "--------------------");
                System.out.println("Invalid entry! Would you like to return to the"
                                + "main menu?");
                System.out.print("(Y)es or (N)o: ");
                answer = userInput.nextLine();
                System.out.println("-----------------------------------------"
                                + "--------------------");
                if (answer.equalsIgnoreCase("Y")) {
                        validEntry = true;
                        redoMain = true;
                }
                else if (answer.equalsIgnoreCase("N")) {
                        validEntry = true;
                        redoMain = false;
                } else {
                        validEntry = false;
                }
        } while (!validEntry);
        return redoMain;
}
```

```
private static void OutputValues (Scanner scan, DistillationColumn column,
		int sheetNum) throws CostingException, BiffException,
		WriteException,
FileNotFoundException, IOException {
	DecimalFormat currency = new DecimalFormat("$#,###.00");
	String answer = "";
	boolean doExport = false;

	System.out.println("\n-----------------------------------------------"
			+ "--------------------");
	System.out.println("\tSHORT VENTURE RESULT SUMMARY:");
	System.out.println("-----------------------------------------------"
			+ "--------------------");
	System.out.println("The number of trays is " + column.getNumberOfTrays());
	System.out.println("The total grassroots is " + currency.format((
			new ModuleCosting().calcGrassRootTotal(column))));
	System.out.println("The yearly profit based on production is " +
			currency.format(column.calcProfit()) + "/year");
	System.out.println("The total profit is " + currency.format(
			column.calcProfit() - new
			ModuleCosting().calcGrassRootTotal(
			column))+" over a one year payback period.");
	System.out.println("-----------------------------------------------"
			+ "--------------------");

	System.out.println("\n-----------------------------------------------"
			+ "--------------------");
	System.out.println("\tEXPORT TO EXCEL?");
	System.out.println("-----------------------------------------------"
			+ "--------------------");
	System.out.println("Would you like to export these results to an Excel "
			+ "file? We'll even \nexport the flow rates and the various
				costs associated with "
			+ "both \nthe column and trays!");
	System.out.print("(Y)es or (N)o: ");
	answer = scan.nextLine();
	System.out.println("-----------------------------------------------"
			+ "--------------------");
	if (answer.equalsIgnoreCase("y"))
		doExport = true;
	else if (answer.equalsIgnoreCase("n")) {
		doExport = false;
	} else doExport = getValidEntry(scan);

	if (doExport)
		callExcelWriter(scan, column, sheetNum);
}

private static void callExcelWriter (Scanner scan, DistillationColumn col,
		int sheetNum) throws BiffException, WriteException,
FileNotFoundException, IOException, CostingException {
	String filepath;
	File excelFile;

	try {
		System.out.println("\n-------------------------------------------"
				+ "--------------------");
		System.out.println("\tCHOOSE EXCEL OUTPUT FILE");
		System.out.println("-----------------------------------------------"
				+ "--------------------");
		System.out.println("Please enter the name of the excel file you "
				+ "would like to output \nresults to and ensure
					extension .xls "
				+ "is included.");
		System.out.println("Also make sure that the file is located inside"
				+ "the current path \nas shown below or enter a "
				+ "file location with its full path.");
		System.out.println("\nCurrent path: "
				+ System.getProperty("user.dir") + "/");
		System.out.print("File name: ");
```

```
                        filepath = scan.nextLine();
                        System.out.println("-----------------------------------------------"
                                + "--------------------");
                        excelFile = new File(filepath);
                } catch (Exception e) {
                        throw e;
                }
                if (excelFile.exists())
                        ExcelWriter.outputToExcelFile(filepath, filepath, sheetNum, col);
                else {
                        System.out.println("\n-----------------------------------------------"
                                + "--------------------");
                        System.out.println("\tINVALID EXCEL FILE! BUT WE GOT YOU
                                                                COVERED.");
                        System.out.println("-----------------------------------------------"
                                + "--------------------");
                        System.out.println("File entered does not appear to exist, "
                                + "creating new excel file \ninstead and "
                                + "appending results to first sheet!");
                        ExcelWriter.outputToExcelFile(filepath, filepath, sheetNum, col);
                        System.out.println("-----------------------------------------------"
                                + "--------------------");
                }
        }
        private static void readTextFromStream(String filepath) {
                try {
                        InputStream is =

        Main.class.getClassLoader().getResourceAsStream(filepath);
                        BufferedReader bufferedReader = new BufferedReader(
                                new InputStreamReader(is, "UTF-8"));

                        String line;

                        while ((line = bufferedReader.readLine()) != null) {
                                System.out.println(line);
                        }

                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
}
```

B.2 costing: CostingException.java

```
package costing;

@SuppressWarnings("serial")
public class CostingException extends Exception {
        public CostingException(String msg) {
                super(msg);
        }
}
```

B.3 costing: ModuleCosting.java

```
package costing;

import java.util.*;

import process.unitoperation.*;

/** This class aims at calculating the capital cost of
 *  the distillation equipment based on the module costing
 *  method. All equations and correlation coefficients were
 *  taken from Annex A from Turton et al. (1998)
 *  and from Ulrich (1984).
 */
```

```
public class ModuleCosting {
	private static final double CEPCI_2014 = 579.7;
	private static final double CEPCI_1996 = 382;

	// capital purchase cost of individual units
	public double calcCapitalPurchase(Column col) throws CostingException {
		double k1, k2, k3;

		if	(col.getDiameter() <= 0.3)
		{k1 = 3.3392; k2 =-0.5538; k3 = 0.2851;}
		else if (col.getDiameter() <= 0.5 && col.getDiameter() >  0.3)
		{k1 = 3.4746; k2 = 0.5893; k3 = 0.2053;}
		else if (col.getDiameter() <= 1.0 && col.getDiameter() >  0.5)
		{k1 = 3.6237; k2 = 0.5262; k3 = 0.2146;}
		else if (col.getDiameter() <= 1.5 && col.getDiameter() >  1.0)
		{k1 = 3.7559; k2 = 0.6361; k3 = 0.1069;}
		else if (col.getDiameter() <= 2.0 && col.getDiameter() >  1.5)
		{k1 = 3.9484; k2 = 0.4623; k3 = 0.1717;}
		else if (col.getDiameter() <= 2.5 && col.getDiameter() >  2.0)
		{k1 = 4.0547; k2 = 0.4620; k3 = 0.1558;}
		else if (col.getDiameter() <= 3.0 && col.getDiameter() >  2.5)
		{k1 = 4.1110; k2 = 0.6094; k3 = 0.0490;}
		else if (col.getDiameter() <= 4.0 && col.getDiameter() >  3.0)
		{k1 = 4.3919; k2 = 0.2859; k3 = 0.1842;}
		else {
			throw new CostingException("The column diameter is outside "
					+ "the applicable range."
					+ "\nRange Required:0 m < diameter <= 4 m");}

		return cpCost(new double[]{k1,k2,k3},col.getLength());
	}
	public double calcCapitalPurchase(Tray[] trays) {
		return 235+19.80*trays[0].getDiameter() +
75.07*Math.pow(trays[0].getDiameter(),2);
	}
	// pressure factor of individual units
	private double[] calcFp(DistillationColumn col) throws CostingException {
		double fp0 = 1.0, fp;

		if (col.getGaugePressure() < 400 && col.getGaugePressure() >= 3.7)  {
			fp = (0.5146
			+ 0.6838*Math.pow((Math.log10(col.getGaugePressure())),1)
			+ 0.2970*Math.pow((Math.log10(col.getGaugePressure())),2)
			+ 0.0235*Math.pow((Math.log10(col.getGaugePressure())),6)
			+ 0.0020*Math.pow((Math.log10(col.getGaugePressure())),8));}


		else if (col.getGaugePressure() < 3.7 && col.getGaugePressure() >= -0.5)
		{fp = 1.0;}
		else if (col.getGaugePressure() < -0.5)
		{fp = 1.25;}
		else {
			throw new CostingException("The column pressure is "
					+ "outside the applicable range."
					+ "\nRange Required: pressure <= 400 psig");}

		return new double[]{fp0,fp};
	}
	// material factor of individual units
	private double[] calcFm(DistillationColumn col) throws CostingException {
		double fm0 = 1.0, fm;

		if	(Arrays.asList("carbon steel","cs").contains(col.getMaterial(
				).toLowerCase()))
		{fm = 1.0;}
		else if (Arrays.asList("stainless steel clad","ss, clad","ss clad").contains(
				col.getMaterial().toLowerCase()))
		{fm = 2.5;}
		else if (Arrays.asList("stainless steel","ss").contains(col.getMaterial(
				).toLowerCase()))
		{fm = 4.0;}
```

```
        else if (Arrays.asList("nickel clad","nickel alloy, clad","nickel, clad",
                        "ni clad","ni, clad").contains(col.getMaterial().toLowerCase()))

        {fm = 4.5;}
        else if (Arrays.asList("nickel","ni").contains(col.getMaterial().toLowerCase()))

        {fm = 9.8;}
        else if (Arrays.asList("titanium clad","ti clad","ti, clad","Titanium, "
                        + "clad").contains(col.getMaterial().toLowerCase()))
        {fm = 4.9;}
        else if (Arrays.asList("titanium","ti").contains(col.getMaterial().toLowerCase()))

        {fm = 10.6;}
        else {
                throw new CostingException("The selected column material does "
                                + "not have any "
                                + "associated costing coefficients: "
                                + col.getMaterial().toLowerCase());}

        return new double[]{fm0,fm};
}
// bare module cost of individual units
public double[] calcBareModule(DistillationColumn col) throws CostingException {
        // only column
        double cbm0, cbm, b1 = 2.50, b2 = 1.72;

        cbm0 = calcCapitalPurchase(col)*(b1 + b2*calcFm(col)[0]*calcFp(col)[0])*(
                        CEPCI_2014/CEPCI_1996);;
        cbm  = calcCapitalPurchase(col)*(b1 + b2*calcFm(col)[1]*calcFp(col)[1])*(
                        CEPCI_2014/CEPCI_1996);;
        return new double[]{cbm0,cbm};
}
public double[] calcBareModule(Tray[] trays) throws CostingException {
        // only trays
        double fbm0 = 1.2, fbm, fq;
        double n = trays.length;
        if              (n >= 1.0  && n < (4.0  + 1.0 )/2.0)
        {fq = 3.0;}
        else if (n >= (4.0  + 1.0)/2.0  && n < (7.0  + 4.0 )/2.0)
        {fq = 2.5;}
        else if (n >= (7.0  + 4.0)/2.0  && n < (10.0 + 7.0)/2.0)
        {fq = 2.0;}
        else if (n >= (10.0 + 7.0)/2.0  && n < 20.0)
        {fq = 1.5;}
        else if (n >= 20.0)
        {fq = 1.0;}
        else {throw new CostingException("Error while determining "
                        + "Fq value for tray costing.");}

        if              (trays[0].getMaterial().equalsIgnoreCase("carbon steel"))
        {fbm = 1.2;}
        else if (trays[0].getMaterial().equalsIgnoreCase("ss, clad"))
        {fbm = 2.0;}
        else if (trays[0].getMaterial().equalsIgnoreCase("nickel alloy"))
        {fbm = 5.0;}
        else {
                throw new CostingException("The selected tray material does not have "
                                + "any associated costing coefficients:
                                "+trays[0].getMaterial());}

        double cbm0 = calcCapitalPurchase(trays)*n*fbm0*fq*(CEPCI_2014/CEPCI_1996);
        double cbm  = calcCapitalPurchase(trays)*n*fbm *fq*(CEPCI_2014/CEPCI_1996);;

        return new double[]{cbm0,cbm};
}
// total module costs of individual units
public double calcTotalModule(DistillationColumn col) throws CostingException {
        return 1.18*calcBareModule(col)[1];
}
public double calcTotalModule(Tray[] trays) throws CostingException {
        return 1.18*calcBareModule(trays)[1];
```

```
        }
        // grass root costs of individual units
        public double calcGrassRoot(DistillationColumn col) throws CostingException {
                return calcTotalModule(col) + 0.50*calcBareModule(col)[0];
        }
        public double calcGrassRoot(Tray[] trays) throws CostingException {
                return calcTotalModule(trays) + 0.50*calcBareModule(trays)[0];
        }
        // total grass root costs
        public double calcGrassRootTotal(DistillationColumn col) throws CostingException {
                return calcGrassRoot(col) + calcGrassRoot(col.getTrays());
        }

        // auxillary method representing function for calculating Cp in log-termed expression
        private double cpCost(double[] kArray,double capParameter) {
                return Math.pow(10, kArray[0] + kArray[1]*Math.log10(capParameter) +
                                kArray[2]*Math.pow(Math.log10(capParameter),2));
        }
}
```

B.4 jexcel.excelextractor: ExcelExtractor.java

```
package excelextractor;

import java.io.File;
import java.io.IOException;
import java.util.Arrays;
import java.util.Scanner;
import java.io.FileNotFoundException;

import jxl.Cell;
import jxl.CellType;
import jxl.Sheet;
import jxl.Workbook;
import jxl.NumberCell;
import jxl.read.biff.BiffException;
import jxl.write.WriteException;

import validation.InvalidArgumentException;
import validation.OutOfRangeArgumentException;
import validation.Validate;

public class ExcelExtractor implements Prompt {

        // ------------------------------------------------
        // INSTANCE VARIABLES
        // ------------------------------------------------

        private double[] eqX;
        private double[] eqY;

        private double feedFlowRate;
        private double[] feedFraction;
        private double[] distillateFraction;
        private double[] bottomsFraction;
        private double[] heatCapacity;
        private double[] normalBP;
        private double latentHeat;
        private double costRaw;
        private double saleDistillate;
        private double saleBottoms;
        private double diameter;
        private double length;
        private double gaugePressure;
        private double refluxRatio;
        private double temperature;
        private String mocColumn;
        private String mocTrays;
```

```
private String trayType;

// ------------------------------------------------
// CONSTRUCTOR(S)
// ------------------------------------------------

public ExcelExtractor() {

        this.feedFraction = new double[] { 0.0 };
        this.distillateFraction = new double[] { 0.0 };
        this.bottomsFraction = new double[] { 0.0 };
        this.heatCapacity = new double[] { 0.0 };
        this.normalBP = new double[] { 0.0 };

        this.feedFlowRate = 0;
        this.latentHeat = 0;
        this.costRaw = 0;
        this.saleDistillate = 0;
        this.saleBottoms = 0;
        this.diameter = 0;
        this.length = 0;
        this.gaugePressure = 0;
        this.refluxRatio = 0;
        this.temperature = 0;
        this.mocColumn = null;
        this.mocTrays = null;
        this.trayType = null;
} // end of ExcelExtractor empty constructor

public ExcelExtractor(double[] eqX, double feedFlowRate, double latentHeat, double
        costRaw, double saleDistillate,
                double saleBottoms, double diameter, double length, double gaugePressure,
        double refluxRatio,
                double temperature, String mocColumn, String mocTrays, String trayType) {
        setEqX(eqX);
        setEqY(eqY);

        setFeedFraction(this.feedFraction);
        setDistillateFraction(this.distillateFraction);
        setBottomsFraction(this.bottomsFraction);
        setHeatCapacity(this.heatCapacity);
        setNormalBP(this.normalBP);

        this.feedFlowRate = feedFlowRate;
        this.latentHeat = latentHeat;
        this.costRaw = costRaw;
        this.saleDistillate = saleDistillate;
        this.saleBottoms = saleBottoms;
        this.diameter = diameter;
        this.length = length;
        this.gaugePressure = gaugePressure;
        this.refluxRatio = refluxRatio;
        this.temperature = temperature;
        this.mocColumn = mocColumn;
        this.mocTrays = mocTrays;
        this.trayType = trayType;
} // end of ExcelExtractor loaded constructor

public ExcelExtractor(ExcelExtractor ee) {
        this(ee.eqX, ee.feedFlowRate, ee.latentHeat, ee.costRaw, ee.saleDistillate,
                ee.saleBottoms, ee.diameter,
                        ee.length, ee.gaugePressure, ee.refluxRatio, ee.temperature,
                ee.mocColumn, ee.mocTrays, ee.trayType);
} // end of ExcelExtractor copy constructor

// ------------------------------------------------
// ACCESSOR AND MUTATOR METHOD(S)
// ------------------------------------------------

// accessor and mutator methods for eqX[]
public double[] getEqX() {
```

```
        double copyEqX[] = new double[this.eqX.length];
        for (int i = 0; i < this.eqX.length; i++)
                copyEqX[i] = this.eqX[i];
        return copyEqX;
}

public void setEqX(double[] eqX) {
        if (!Validate.isLessThan1(eqX) || !Validate.isNonNegative(eqX))
                throw new OutOfRangeArgumentException("Liquid equilibrium fraction",
        Arrays.toString(eqX), 0.0, 1.0);
        this.eqX = new double[this.eqX.length];
        for (int i = 0; i < this.eqX.length; i++)
                this.eqX[i] = eqX[i];
}

// accessor and mutator methods for eqY[]
public double[] getEqY() {
        double copyEqY[] = new double[this.eqY.length];
        for (int i = 0; i < this.eqY.length; i++)
                copyEqY[i] = this.eqY[i];
        return copyEqY;
}

public void setEqY(double[] eqY) {
        if (!Validate.isLessThan1(eqY) || !Validate.isNonNegative(eqX))
                throw new OutOfRangeArgumentException("Vapor equilibrium fraction",
        Arrays.toString(eqY), 0.0, 1.0);
        this.eqY = new double[this.eqY.length];
        for (int i = 0; i < this.eqY.length; i++) {
                this.eqY[i] = eqY[i];
        }
}

// accessor and mutator for feedFlowRate
public double getFeedFlowRate() {
        return this.feedFlowRate;
}

public void setFeedFlowRate(double feedFlowRate) {
        if (!Validate.isNonNegative(feedFlowRate))
                throw new InvalidArgumentException("feed flow rate", feedFlowRate, "non
negative");
        this.feedFlowRate = feedFlowRate;
}

// accessor and mutator for feedFraction[]
public double[] getFeedFraction() {
        double[] copyFeedFraction = new double[this.feedFraction.length];
        for (int i = 0; i < this.feedFraction.length; i++)
                copyFeedFraction[i] = this.feedFraction[i];
        return copyFeedFraction;
}

public void setFeedFraction(double[] feedFraction) {
        if (!Validate.isLessThan1(feedFraction) || !Validate.isNonNegative(feedFraction))
                throw new OutOfRangeArgumentException("feed fraction",
        Arrays.toString(feedFraction), 0.0, 1.0);
        this.feedFraction = new double[this.feedFraction.length];
        for (int i = 0; i < this.feedFraction.length; i++)
                this.feedFraction[i] = feedFraction[i];
}

// accessor and mutator methods for distillateFraction[]
public double[] getDistillateFraction() {
        double[] copyDistillateFraction = new double[this.distillateFraction.length];
        for (int i = 0; i < this.distillateFraction.length; i++)
                copyDistillateFraction[i] = this.distillateFraction[i];
        return copyDistillateFraction;
}

public void setDistillateFraction(double[] distillateFraction) {
```

```
        if (!Validate.isLessThan1(distillateFraction) ||
        !Validate.isNonNegative(distillateFraction))
                throw new OutOfRangeArgumentException("Distillate fraction",
        Arrays.toString(distillateFraction), 0.0, 1.0);
        this.distillateFraction = new double[this.distillateFraction.length];
        for (int i = 0; i < this.distillateFraction.length; i++)
                this.distillateFraction[i] = distillateFraction[i];
}

// accessor and mutator methods for bottomsFraction[]
public double[] getBottomsFraction() {
        double[] copyBottomsFraction = new double[this.bottomsFraction.length];
        for (int i = 0; i < this.bottomsFraction.length; i++)
                copyBottomsFraction[i] = this.bottomsFraction[i];
        return copyBottomsFraction;
}

public void setBottomsFraction(double[] bottomsFraction) {
        if (!Validate.isLessThan1(bottomsFraction) ||
        !Validate.isNonNegative(bottomsFraction))
                throw new InvalidExcelCellException("Bottoms fraction data out of
        bounds\nRange required: 0 to 1\n");
        this.bottomsFraction = new double[this.bottomsFraction.length];
        for (int i = 0; i < this.bottomsFraction.length; i++)
                this.bottomsFraction[i] = bottomsFraction[i];
}

// accessor and mutator methods for heatCapacity[]
public double[] getHeatCapacity() {
        double[] copyHeatCapacity = new double[this.heatCapacity.length];
        for (int i = 0; i < this.heatCapacity.length; i++)
                copyHeatCapacity[i] = this.heatCapacity[i];
        return copyHeatCapacity;
}

public void setHeatCapacity(double[] heatCapacity) {
        if (!Validate.isNonNegative(heatCapacity))
                throw new InvalidArgumentException("heat capacity",
        Arrays.toString(heatCapacity), "non negative");
        this.heatCapacity = new double[this.heatCapacity.length];
        for (int i = 0; i < this.heatCapacity.length; i++)
                this.heatCapacity[i] = heatCapacity[i];
}

// accessor and mutator methods for normalBP[]
public double[] getNormalBP() {
        double[] copyNormalBP = new double[this.normalBP.length];
        for (int i = 0; i < this.normalBP.length; i++)
                copyNormalBP[i] = this.normalBP[i];
        return copyNormalBP;
}

public void setNormalBP(double[] normalBP) {
        if (!Validate.isNonNegative(normalBP))
                throw new InvalidArgumentException("normal BP", Arrays.toString(normalBP),
        "non negative");
        this.normalBP = new double[this.normalBP.length];
        for (int i = 0; i < this.normalBP.length; i++)
                this.normalBP[i] = normalBP[i];
}

// accessor and mutator methods for latentHeat
public double getLatentHeat() {
        return this.latentHeat;
}

public void setLatentHeat(double latentHeat) {
        if (!Validate.isNonNegative(latentHeat))
                throw new InvalidArgumentException("latent heat", latentHeat, "non
        negative");
        this.latentHeat = latentHeat;
```

```
}

// accessor and mutator methods for costRaw
public double getCostRaw() {
	return this.costRaw;
}

public void setCostRaw(double costRaw) {
	if (!Validate.isNonNegative(costRaw))
		throw new InvalidArgumentException("feed cost", costRaw, "non negative");
	this.costRaw = costRaw;
}

// accessor and mutator methods for saleDistillate
public double getSaleDistillate() {
	return this.saleDistillate;
}

public void setSaleDistillate(double saleDistillate) {
	if (!Validate.isNonNegative(saleDistillate))
		throw new InvalidArgumentException("distillate price", saleDistillate,
	"non negative");
	this.saleDistillate = saleDistillate;
}

// accessor and mutator methods for saleBottoms
public double getSaleBottoms() {
	return this.saleBottoms;
}

public void setSaleBottoms(double saleBottoms) {
	if (!Validate.isNonNegative(saleBottoms))
		throw new InvalidArgumentException("bottoms price", saleBottoms, "non
	negative");
	this.saleBottoms = saleBottoms;
}

// accessor and mutator methods for diameter
public double getDiameter() {
	return this.diameter;
}

public void setDiameter(double diameter) {
	if (!Validate.isNonNegative(diameter))
		throw new InvalidArgumentException("diameter", diameter, "non negative");
	this.diameter = diameter;
}

// accessor and mutator methods for length
public double getLength() {
	return this.length;
}

public void setLength(double length) {
	if (!Validate.isNonNegative(length))
		throw new InvalidArgumentException("length", length, "non negative");
	this.length = length;
}

// accessor and mutator methods for gaugePressure
public double getGaugePressure() {
	return this.gaugePressure;
}

public void setGaugePressure(double gaugePressure) {
	if (!Validate.isNonNegative(gaugePressure))
		throw new InvalidArgumentException("gauge pressure", gaugePressure, "non
	negative");
	this.gaugePressure = gaugePressure;
}
```

```
// accessor and mutator methods for refluxRatio
public double getRefluxRatio() {
	return this.refluxRatio;
}

public void setRefluxRatio(double refluxRatio) {
	if (!Validate.isNonNegative(refluxRatio))
		throw new InvalidArgumentException("reflux ratio", refluxRatio, "non
	negative");
	this.refluxRatio = refluxRatio;
}

// accessor and mutator methods for temperature
public double getTemperature() {
	return this.temperature;
}

public void setTemperature(double temperature) {
	if (!Validate.isNonNegative(temperature))
		throw new InvalidArgumentException("temperature", temperature, "non
	negative");
	this.temperature = temperature;
}

// accessor and mutator methods for mocColumn
public String getMocColumn() {
	return this.mocColumn;
}

public void setMocColumn(String mocColumn) {
	this.mocColumn = mocColumn;
}

// accessor and mutator methods for mocTrays
public String getMocTrays() {
	return this.mocTrays;
}

public void setMocTrays(String mocTrays) {
	this.mocTrays = mocTrays;
}

// accessor and mutator methods for trayType
public String getTrayType() {
	return this.trayType;
}

public void setTrayType(String trayType) {
	this.trayType = trayType;
}

// ------------------------------------------------
// EXCEL EXTRACTION METHODS
// ------------------------------------------------
// method to extract equilibrium data from excel
public void extractEq(String workbook, int eqSheet)
		throws BiffException, IOException, WriteException, FileNotFoundException {
	Workbook venture = Workbook.getWorkbook(new File(workbook)); // instantiate
	// Workbook
	// object
	// venture
	Sheet vEq = venture.getSheet(eqSheet);

	this.eqX = new double[vEq.getColumn(5).length - 4];
	this.eqY = new double[vEq.getColumn(5).length - 4];

	// NumberCell nc = null;
	double[] eqX = new double[this.eqX.length];
	double[] eqY = new double[this.eqY.length];

	// check for data type and range for x and y equilibrium mole fraction
```

```
        // assign to instance variables eqX[] and eqY[] if acceptable
        for (int i = 0; i < vEq.getColumn(6).length - 4; i++) {

                Cell cell1 = vEq.getCell(5, i + 4);

                if (cell1.getType() != CellType.NUMBER)
                        throw new InvalidExcelCellException(
                                        "Invalid data type for liquid equilibrium mole
                fraction\nRequired: numeric\n");
                eqX[i] = ((NumberCell) cell1).getValue();

                Cell cell2 = vEq.getCell(6, i + 4);

                if (cell2.getType() != CellType.NUMBER)
                        throw new InvalidExcelCellException(
                                        "Invalid data type for vapour equilibrium mole
        fraction\nRequired: numeric\n");
                eqY[i] = ((NumberCell) cell2).getValue();

        }
        this.setEqX(eqX);
        this.setEqY(eqY);
} // end of extractEq method

// method to extract venture data from excel
public void extractVentureData(String workbook, int propSheet)
                throws BiffException, IOException, WriteException, FileNotFoundException {

        Workbook venture = Workbook.getWorkbook(new File(workbook));
        Sheet vData = venture.getSheet(propSheet);

        this.feedFraction = new double[2];
        this.distillateFraction = new double[2];
        this.bottomsFraction = new double[2];
        this.heatCapacity = new double[2];
        this.normalBP = new double[2];

        Cell b3 = vData.getCell(1, 2);
        if (b3.getType() != CellType.NUMBER)
                throw new InvalidExcelCellException("Invalid data type for feed flow
        rate\nRequired: numeric\n");
        this.setFeedFlowRate(((NumberCell) b3).getValue());

        // check for data type and range for feed component mole fraction
        // assign to instance variable feedFraction[] if acceptable
        double[] feedFraction = new double[this.feedFraction.length];
        for (int i = 0; i < this.feedFraction.length; i++) {
                Cell cell = vData.getCell(i + 1, 5);

                if (cell.getType() != CellType.NUMBER)
                        throw new InvalidExcelCellException(
                                        "Invalid data type for component feed
        fraction\nRequired: numeric\n");
                feedFraction[i] = ((NumberCell) cell).getValue();
        }
        this.setFeedFraction(feedFraction);

        // check for data type and range for distillate mole fraction
        // assign to instance variable distillateFraction[] if acceptable
        double[] distillateFraction = new double[this.distillateFraction.length];
        for (int i = 0; i < this.distillateFraction.length; i++) {
                Cell cell = vData.getCell(i + 1, 6);
                if (cell.getType() != CellType.NUMBER)
                        throw new InvalidExcelCellException(
                                        "Invalid data type for component distillate
        fraction\nRequired: numeric\n");
                distillateFraction[i] = ((NumberCell) cell).getValue();
        }
        this.setDistillateFraction(distillateFraction);

        // check for data type and range for bottoms mole fraction
```

```
// assign to instance variable bottomsFraction[] if acceptable
double[] bottomsFraction = new double[this.bottomsFraction.length];
for (int i = 0; i < this.bottomsFraction.length; i++) {
	Cell cell = vData.getCell(i + 1, 7);
	if (cell.getType() != CellType.NUMBER)
		throw new InvalidExcelCellException(
				"Invalid data type for component bottoms
fraction\nRequired: numeric\n");
	bottomsFraction[i] = ((NumberCell) cell).getValue();
}
this.setBottomsFraction(bottomsFraction);

// check for data type and range for component heat capacity
// assign to instance variable heatCapacity[] if acceptable
double[] heatCapacity = new double[this.heatCapacity.length];
for (int i = 0; i < 2; i++) {
	Cell cell = vData.getCell(i + 1, 10);
	if (cell.getType() != CellType.NUMBER)
		throw new InvalidExcelCellException(
				"Invalid data type for component heat
capacity\nRequired: numeric\n");
	heatCapacity[i] = ((NumberCell) cell).getValue();
}
this.setHeatCapacity(heatCapacity);

// check for data type and range for component normal boiling point
// assign to instance variable normalBP[] if acceptable
double[] normalBP = new double[this.normalBP.length];
for (int i = 0; i < this.normalBP.length; i++) {
	Cell cell = vData.getCell(i + 1, 11);
	if (cell.getType() != CellType.NUMBER)
		throw new InvalidExcelCellException(
				"Invalid data type for component normal boiling
point\nRequired: numeric\n");
	normalBP[i] = ((NumberCell) cell).getValue();
}
this.setNormalBP(normalBP);

// check for data type and range for feed latent heat
// assign to instance variable latentHeat if acceptable
Cell b12 = vData.getCell(1, 12);
if (b12.getType() != CellType.NUMBER)
	throw new InvalidExcelCellException("Invalid data type for latent
heat\nRequired: numeric\n");
this.setLatentHeat(((NumberCell) b12).getValue());

// check for data type and range for column diameter
// assign to instance variable diameter if acceptable
Cell b15 = vData.getCell(1, 14);
if (b15.getType() != CellType.NUMBER)
	throw new InvalidExcelCellException("Invalid data type for column
diameter\nRequired: numeric\n");
this.setDiameter(((NumberCell) b15).getValue());

// check for data type and range for column length
// assign to instance variable length if acceptable
Cell b16 = vData.getCell(1, 15);
if (b16.getType() != CellType.NUMBER)
	throw new InvalidExcelCellException("Invalid data type for column
length\nRequired: numeric\n");
this.setLength(((NumberCell) b16).getValue());

// check for data type and range for column pressure
// assign to instance gaugePressure if acceptable
Cell b17 = vData.getCell(1, 16);
if (b17.getType() != CellType.NUMBER)
	throw new InvalidExcelCellException("Invalid data type for column
	pressure\nRequired: numeric\n");
this.setGaugePressure(((NumberCell) b17).getValue());

// check for data type and range for reflux ratio
```

```
        // assign to instance variable refluxRatio if acceptable
        Cell b18 = vData.getCell(1, 17);
        if (b18.getType() != CellType.NUMBER)
               throw new InvalidExcelCellException("Invalid data type for reflux
        ratio\nRequired: numeric\n");
        this.setRefluxRatio(((NumberCell) b18).getValue());

        // check for data type and range for column temperature
        // assign to instance variable temperature if acceptable
        Cell b19 = vData.getCell(1, 18);
        if (b19.getType() != CellType.NUMBER)
               throw new InvalidExcelCellException("Invalid data type for
        temperature\nRequired: numeric\n");
        this.setTemperature(((NumberCell) b19).getValue());

        // check for data type for column material of construction
        // assign to instance variable mocColumn if acceptable
        Cell b20 = vData.getCell(1, 19);
        if (b20.getType() != CellType.LABEL)
               throw new InvalidExcelCellException(
                              "Invalid data type for column material of
        construction\nRequired: String\n");
        this.setMocColumn(b20.getContents());

        // check for data type for tray material of construction
        // assign to instance variable mocTrays if acceptable
        Cell b21 = vData.getCell(1, 20);
        if (b21.getType() != CellType.LABEL)
               throw new InvalidExcelCellException(
                              "Invalid data type for tray material of
        construction\nRequired: String\n");
        this.setMocTrays(b21.getContents());

        // check for data type for tray type
        // assign to instance variable trayType if acceptable
        Cell b22 = vData.getCell(1, 21);
        if (b22.getType() != CellType.LABEL)
               throw new InvalidExcelCellException("Invalid data type for tray
        type\nRequired: String\n");
        this.setTrayType(b22.getContents());

        // check for data type and range for raw material cost
        // assign to instance variable costRaw if acceptable
        Cell b24 = vData.getCell(1, 23);
        if (b24.getType() != CellType.NUMBER)
               throw new InvalidExcelCellException("Invalid data type for raw material
        cost\nRequired: numeric\n");
        this.setCostRaw(((NumberCell) b24).getValue());

        // check for data type and range for distillate sale price
        // assign to instance variable saleDistillate if acceptable
        Cell b25 = vData.getCell(1, 24);
        if (b25.getType() != CellType.NUMBER)
               throw new InvalidExcelCellException("Invalid data type for distillate sale
        price\nRequired: numeric\n");
        this.setSaleDistillate((((NumberCell) b25).getValue()));

        // check for data type and range for bottoms sale price
        // assign to instance variable saleBottoms if acceptable
        Cell b26 = vData.getCell(1, 25);
        if (b26.getType() != CellType.NUMBER)
               throw new InvalidExcelCellException("Invalid data type for bottoms sale
        price\nRequired: numeric\n");
        this.setSaleBottoms((((NumberCell) b26).getValue()));

} // end of extractVentureData method

// ------------------------------------------------
// PROMPT INTERFACE METHODS
// ------------------------------------------------
@Override
```

```
public int prompt(Scanner in) throws BiffException, IOException, WriteException,
FileNotFoundException {

	String filepath;
	String buffer;
	File excelFile;
	int sheetNum;

	try {
		System.out.print(
				"\nPlease enter the name of the excel file you would like
				to analyze \nincluding the .xls extension. ");
		System.out.println(

				"Also make sure that the file is \nlocated inside the
		current path as shown below, or enter the full");
		System.out.println("path where the file is located.");
		System.out.print("\nEx: 'Venture.xls' OR ");
		System.out.println("'/Users/username/Desktop/Venture.xls', \nwithout the
		quotes.");
		System.out.println("\nCurrent path: " + System.getProperty("user.dir") +
		"/");
		System.out.print("File name: ");
		filepath = in.nextLine();
		excelFile = new File(filepath);
	} catch (Exception e) {
		throw e;
	}
	if (!excelFile.exists())
		throw new InvalidExcelFileException("File does not exist!"); // exception
	// thrown
	// if
	// the
	// file
	// does
	// not
	// exist

	System.out.println("\nPlease enter the sheet index for the venture specific
	data");
	System.out.print("Sheet index: ");
	try {
		buffer = in.nextLine(); // User input to specify the excel sheet
		// number they would like analyzed
		if (!Validate.isType(buffer, "int"))
			throw new InvalidConsoleEntryException();
		sheetNum = Integer.parseInt(buffer);
		this.extractVentureData(filepath, sheetNum - 1);
		this.extractEq(filepath, sheetNum - 1);
	} catch (IndexOutOfBoundsException e) {
		throw new InvalidExcelSheetException();
	}
	return sheetNum;
}

@Override
public void prompt(String filepath, int sheetNum) throws BiffException, IOException,
WriteException {

	File excelFile = new File(filepath);

	if (excelFile.exists()) {
		Workbook venture = Workbook.getWorkbook(new File(filepath));

		int numSheets = venture.getNumberOfSheets();

		if (sheetNum <= numSheets) {
			this.extractVentureData(filepath, sheetNum - 1);
			this.extractEq(filepath, sheetNum - 1);
		} else
			throw new InvalidExcelSheetException();
```

```
                } else
                        throw new InvalidExcelFileException();
        }
        // ------------------------------------------------
        // CLONE METHOD
        // ------------------------------------------------

        @Override
        public ExcelExtractor clone() {
                return new ExcelExtractor(this);
        }

} // end of excel extractor class
```

B.5 jexcel.excelextractor: Prompt.java

```
package excelextractor;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Scanner;

import jxl.read.biff.BiffException;
import jxl.write.WriteException;

public interface Prompt {
        public void prompt (String filepath, int sheetNum) throws BiffException, IOException,
WriteException, FileNotFoundException;
        public int prompt (Scanner scan) throws BiffException, IOException, WriteException,
FileNotFoundException;
}
```

B.6 jexcel.excelextractor: InvalidConsoleEntryException.java

```
package excelextractor;

import java.io.IOException;

@SuppressWarnings("serial")
public class InvalidConsoleEntryException extends IOException {
        public InvalidConsoleEntryException() {
                super();
        }
        public InvalidConsoleEntryException(String msg) {
                super(msg);
        }
}
```

B.7 jexcel.excelextractor: InvalidExcelCellException.java

```
package excelextractor;

import validation.InvalidArgumentException;

@SuppressWarnings("serial")
public class InvalidExcelCellException extends InvalidArgumentException {
        public InvalidExcelCellException() {
                super();
        }
        public InvalidExcelCellException(String msg) {
                super(msg);
        }
}
```

B.8 jexcel.excelextractor: InvalidExcelFileException.java

```
package excelextractor;

import java.io.FileNotFoundException;
```

```
@SuppressWarnings("serial")
public class InvalidExcelFileException extends FileNotFoundException {
	public InvalidExcelFileException() {
		super();
	}
	public InvalidExcelFileException(String msg) {
		super(msg);
	}
}
```

B.9 jexcel.excelextractor: InvalidExcelSheetException.java

```
package excelextractor;

import java.io.IOException;

@SuppressWarnings("serial")
public class InvalidExcelSheetException extends IOException {
	public InvalidExcelSheetException() {
		super();
	}
	public InvalidExcelSheetException(String msg) {
		super(msg);
	}
}
```

B.10 jexcel.excelwriter: ExcelWriter.java

```
package excelwriter;

import java.io.File;
import java.io.IOException;
import java.text.DecimalFormat;

import costing.CostingException;
import costing.ModuleCosting;

import java.io.FileNotFoundException;

import jxl.Workbook;

import jxl.read.biff.BiffException;

import jxl.write.Label;

import jxl.write.WritableSheet;
import jxl.write.WritableWorkbook;
import jxl.write.WriteException;
import process.unitoperation.DistillationColumn;

public class ExcelWriter {

	public static void outputToExcelFile(String srcpath, String despath, int wkbVentureSheet,
DistillationColumn col) throws BiffException, IOException, WriteException, FileNotFoundException,
CostingException  {
		DecimalFormat currency = new DecimalFormat("$#,###.00");
		DecimalFormat df = new DecimalFormat("####.##");
		DecimalFormat idf = new DecimalFormat("####");
		Workbook srcwkb;
		WritableWorkbook deswkb;
		WritableSheet wks;
		try {
			srcwkb = Workbook.getWorkbook(new File(srcpath));
			deswkb = Workbook.createWorkbook(new File(despath), srcwkb);
			wks = deswkb.createSheet("Output Results - " +
			srcwkb.getSheetNames()[wkbVentureSheet-1],srcwkb.getNumberOfSheets());
			wks.addCell(new Label(1, 1, "Results - " +
			srcwkb.getSheetNames()[wkbVentureSheet-1]));
```

```
//-----------------------------------------------------------------------
//number of trays
wks.addCell(new Label(1, 2, "Number of Trays"));
wks.addCell(new Label(2, 3, idf.format(col.getNumberOfTrays())));

//distillate flow rate
wks.addCell(new Label(1, 4, "Distillate Molar Flow Rate"));
wks.addCell(new Label(2, 5, df.format(col.calcFlowRates()[1])));
wks.addCell(new Label(3, 5, "kmol/h"));

//bottoms flow rate
wks.addCell(new Label(1, 6, "Bottoms Molar Flow Rate"));
wks.addCell(new Label(2, 7, df.format(col.calcFlowRates()[2])));
wks.addCell(new Label(3, 7, "kmol/h"));

//Purchase cost of column and trays
wks.addCell(new Label(1, 9, "Purchase Cost"));
wks.addCell(new Label(1, 10, "Column:"));
wks.addCell(new Label(2, 10, currency.format((new
ModuleCosting()).calcCapitalPurchase(col))));
wks.addCell(new Label(1, 11, "Trays"));
wks.addCell(new Label(2, 11, currency.format((new
ModuleCosting()).calcCapitalPurchase(col.getTrays()))));

//Bare module cost of column and trays
wks.addCell(new Label(1, 12, "Bare Module Cost"));
wks.addCell(new Label(1, 13, "Column"));
wks.addCell(new Label(2, 13, currency.format((new
ModuleCosting()).calcBareModule(col)[1])));
wks.addCell(new Label(1, 14, "Trays"));
wks.addCell(new Label(2, 14, currency.format((new
ModuleCosting()).calcBareModule(col.getTrays())[1])));

//Total module cost
wks.addCell(new Label(1, 16, "Total Module Cost"));
wks.addCell(new Label(2, 17, currency.format(1.18*((new
ModuleCosting()).calcBareModule(col)[1]+ (new
ModuleCosting()).calcBareModule(col.getTrays())[1])));

//Grassroots
wks.addCell(new Label(1, 19, "Grassroots Cost"));
wks.addCell(new Label(2, 20, currency.format((new
ModuleCosting()).calcGrassRootTotal(col))));

//Profit
wks.addCell(new Label(1, 22, "The yearly profit based on production"));
wks.addCell(new Label(2, 23, currency.format(col.calcProfit())));
wks.addCell(new Label(3, 23, "/year"));

//Profit based on one year PBP
wks.addCell(new Label(1, 25, "The total profit based on a one year payback
period"));
wks.addCell(new Label(2, 26, currency.format(col.calcProfit() - (new
ModuleCosting()).calcGrassRootTotal(col))));
//-----------------------------------------------------------------------

deswkb.write();
deswkb.close();
srcwkb.close();
} catch (FileNotFoundException e) {
deswkb = Workbook.createWorkbook(new File(srcpath));
wks = deswkb.createSheet("Output Results",0);
wks.addCell(new Label(1, 1, "Results"));

//-----------------------------------------------------------------------
//number of trays
wks.addCell(new Label(1, 2, "Number of Trays"));
wks.addCell(new Label(2, 3, idf.format(col.getNumberOfTrays())));

//distillate flow rate
```

```
                        wks.addCell(new Label(1, 4, "Distillate Molar Flow Rate"));
                        wks.addCell(new Label(2, 5, df.format(col.calcFlowRates()[1])));
                        wks.addCell(new Label(3, 5, "kmol/h"));

                        //bottoms flow rate
                        wks.addCell(new Label(1, 6, "Bottoms Molar Flow Rate"));
                        wks.addCell(new Label(2, 7, df.format(col.calcFlowRates()[2])));
                        wks.addCell(new Label(3, 7, "kmol/h"));

                        //Purchase cost of column and trays
                        wks.addCell(new Label(1, 9, "Purchase Cost"));
                        wks.addCell(new Label(1, 10, "Column:"));
                        wks.addCell(new Label(2, 10, currency.format((new
                        ModuleCosting()).calcCapitalPurchase(col))));
                        wks.addCell(new Label(1, 11, "Trays"));
                        wks.addCell(new Label(2, 11, currency.format((new
                        ModuleCosting()).calcCapitalPurchase(col.getTrays()))));

                        //Bare module cost of column and trays
                        wks.addCell(new Label(1, 12, "Bare Module Cost"));
                        wks.addCell(new Label(1, 13, "Column"));
                        wks.addCell(new Label(2, 13, currency.format((new
                        ModuleCosting()).calcBareModule(col)[1])));
                        wks.addCell(new Label(1, 14, "Trays"));
                        wks.addCell(new Label(2, 14, currency.format((new
                        ModuleCosting()).calcBareModule(col.getTrays())[1])));

                        //Total module cost
                        wks.addCell(new Label(1, 16, "Total Module Cost"));
                        wks.addCell(new Label(2, 17, currency.format(1.18*((new
                        ModuleCosting()).calcBareModule(col)[1]+ (new
                ModuleCosting()).calcBareModule(col.getTrays())[1])));

                        //Grassroots
                        wks.addCell(new Label(1, 19, "Grassroots Cost"));
                        wks.addCell(new Label(2, 20, currency.format((new
                        ModuleCosting()).calcGrassRootTotal(col))));

                        //Profit
                        wks.addCell(new Label(1, 22, "The yearly profit based on production"));
                        wks.addCell(new Label(2, 23, currency.format(col.calcProfit())));
                        wks.addCell(new Label(3, 23, "/year"));

                        //Profit based on one year PBP
                        wks.addCell(new Label(1, 25, "The total profit based on a one year payback
                        period"));
                        wks.addCell(new Label(2, 26, currency.format(col.calcProfit() - (new
                        ModuleCosting()).calcGrassRootTotal(col))));
                        //--------------------------------------------------------------------------

                        deswkb.write();
                        deswkb.close();
                }



        }
}
```

B.11 numerical.analysis : Function.java

```
package numerical.analysis;

/** This class aims to design the data structure
 *  behind the concept of a Function as an interface
 *  simply because it only requires two methods and
 *  has no instance variables.
 */
public interface Function {
        double calcY(double x);
```

```
	double[] calcY(double[] x);
}
```

B.12 numerical.analysis : LinearFunction.java

```
package numerical.analysis;

import validation.Validate;

/** This class aims to implement the Function interface
 *  methods as well as incorporate various instance
 *  variables which are akin to linear functions
 *  (such as slope and intercept) and implement additional
 *  methods such as calculating the intersection point of
 *  two LinearFunction objects.
 */
public class LinearFunction implements Function {

	//-------------------------------------------------
	//	INSTANCE VARIABLE(S)
	//-------------------------------------------------
	private double slope;		// every LinearFunction class must have a slope IV
	private double intercept;	// every LinearFunction class must have an intercept IV

	//-------------------------------------------------
	//	CONSTRUCTOR(S)
	//-------------------------------------------------
	// empty constructor
	public LinearFunction() {
		this.setSlope(0.);
		this.setIntercept(0.);
	}
	// standard constructor
	public LinearFunction(double slope, double intercept) {
		this.setSlope(slope);
		this.setIntercept(intercept);
	}
	// copy constructor
	public LinearFunction(LinearFunction lf) {
		this(lf.slope, lf.intercept); // uses standard constructor to assign IVs
	}

	//-------------------------------------------------
	//	ACCESSOR(S) AND MUTATOR(S) METHOD(S)
	//-------------------------------------------------
	// accessor and mutator methods for slope IV
	public double getSlope() {
		Validate.checkNotNull(this.slope);
		return this.slope;
	}
	public void setSlope(double slope) {
		Validate.checkNotNull(slope);
		this.slope = slope;
	}
	// accessor and mutator methods for intercept IV
	public double getIntercept() {
		Validate.checkNotNull(this.intercept);
		return this.intercept;
	}
	public void setIntercept(double intercept) {
		Validate.checkNotNull(intercept);
		this.intercept = intercept;
	}

	//-------------------------------------------------
	//	OTHER METHOD(S)
	//-------------------------------------------------
	// implemented calcY methods for double x and double[] x arguments
	public double calcY(double x) {
```

```
                Validate.checkNotNull(x);
                return this.getSlope()*x + this.getIntercept();
        }
        public double[] calcY(double[] x) {
                Validate.checkNotNull(x);
                double[] y = new double[x.length];
                for(int i = 0; i < x.length; i++)
                        y[i] = this.calcY(x[i]);
                return y;
        }

        /* uniquely defined calcX method to solve for the x value of the
         * LinearFunction given a double y and double[] y
         */
        public double calcX(double y) {
                Validate.checkNotNull(y);
                if (this.getSlope() == 0.0)
                        throw new ArithmeticException("Division by zero error! Cannot have slope
                value of zero.");
                return (y - this.getIntercept())/this.getSlope();
        }
        public double[] calcX(double[] y) {
                Validate.checkNotNull(y);
                double[] x = new double[y.length];
                for(int i = 0; i < y.length; i++)
                        x[i] = this.calcX(y[i]);
                return x;
        }

        /* static method to calculate the intersection of two LinearFunction
         * instances and returning a vector [x, y] of this intersection point
         */
        public static double[] calcIntersection(LinearFunction lf1, LinearFunction lf2) {
                Validate.checkNotNull(lf1);
                Validate.checkNotNull(lf2);
                double[] point = new double[2];
                if ((lf2.getSlope() - lf1.getSlope()) == 0)
                                throw new ArithmeticException("Division by zero error! Cannot have
                        denominator value of zero when calculating intersection point!.");
                point[0] = (lf1.getIntercept() - lf2.getIntercept())/(lf2.getSlope() -
lf1.getSlope());
                point[1] = lf2.getSlope()*point[0] + lf2.getIntercept();
                return point;
        }

        //-----------------------------------------------
        //      CLONE METHOD
        //-----------------------------------------------
        @Override
        public LinearFunction clone() {
                return new LinearFunction(this);
        }
}
```

B.13 numerical.analysis : Maths.java

```
package numerical.analysis;

import validation.Validate;

/** This class implements various static methods useful
 *  for the various numerical package classes
 */
public class Maths
{
        //-----------------------------------------------
        //      OTHER METHOD(S)
        //-----------------------------------------------
```

```
public static void toString(double[] x){
	String s = "[";
	for (int i = 0; i < x.length; i++) {
		if		(i == 0)
			s += String.format("%.2f", x[i]);
		else
			s += String.format(", %.2f", x[i]);
	}
	s += "]";
	System.out.println(s);
}

// sqrt(a^2 + b^2) without under/overflow
public static double hypot(double a, double b) {
	double r;
	if (Math.abs(a) > Math.abs(b)) {
		r = b/a;
		r = Math.abs(a)*Math.sqrt(1+r*r);
	} else if (b != 0) {
		r = a/b;
		r = Math.abs(b)*Math.sqrt(1+r*r);
	} else {
		r = 0.0;
	}
	return r;
}

/* method to calculate the difference between adjacent
 * elements within an array
 */
public static double[] diff(double[] x) {
	Validate.checkNotNull(x);
	Validate.checkNotEmpty(x);
	double[] y = new double[x.length];
	for (int i = 0; i < x.length - 1; i++){
		y[i] = x[i + 1] - x[i];
	}
	return y;
}

/* method to calculate the difference between adjacent
 * elements within each column (difference between
 * elements in the first dimension, i.e. between rows)
 */
public static double[][] diff(double[][] x) { // throws raggedArrayException{
	Validate.checkNotNull(x);
	int rows = x.length;
	int cols = x[0].length;
	double[][] y = new double[rows - 1][cols];
	for (int j = 0; j < cols; j++) {
		Validate.checkNotEmpty(x[j]);
		for (int i = 0; i < rows - 1; i++) {
			y[i][j] = x[i + 1][j] - x[i][j];
		}
	}
	return y;
}

/* method to calculate the product of each element
 * within an array
 */
public static double prod(double[] x) { // throws arrayTooSmallException
	Validate.checkNotNull(x);
	Validate.checkNotEmpty(x);
	double y = x[0];
	for (int i = 1; i < x.length; i++) {
		y = y*x[i];
	}
	return y;
}
```

```
        /* method to calculate the product of each element
         * along both dimensions of the array
         */
        public static double[] prod(double[][] x) { // throws raggedArrayException
                Validate.checkNotNull(x);
                int rows = x.length;
                int cols = x[0].length;
                double[] y = new double[cols];

                for (int j = 0; j < cols; j++) {
                        Validate.checkNotEmpty(x[j]);
                        y[j] = x[0][j];
                }
                for (int j = 0; j < cols; j++) {
                        for (int i = 1; i < rows; i++) {
                                y[j] = y[j]*x[i][j];
                        }
                }
                return y;
        }
}
```

B.14 numerical.analysis : PolynomialFunction.java

```
package numerical.analysis;

import validation.EmptyArrayArgumentException;
import validation.*;

/** This class aims to extend the concept of a function
 *  through the Function interface to polynomial
 *  functions along with additional instance variables
 *  and methods.
 */
public class PolynomialFunction implements Function {

        //-------------------------------------------------
        //      INSTANCE VARIABLE(S)
        //-------------------------------------------------
        private final double coefficients[];

        //-------------------------------------------------
        //      CONSTRUCTOR(S)
        //-------------------------------------------------
        // empty constructor
        public PolynomialFunction() {
                this.coefficients = null;
        }
        // standard constructor
        public PolynomialFunction(double c[]) throws EmptyArrayArgumentException,
        NullArgumentException {
                Validate.checkNotNull(c);
                Validate.checkNotEmpty(c);
                int n = c.length;
                while ((n > 1) && (c[n - 1] == 0)) {
                        --n;
                }
                this.coefficients = new double[n];
                System.arraycopy(c, 0, this.coefficients, 0, n);
        }
        // copy constructor
        public PolynomialFunction(PolynomialFunction pf) {
                this(pf.coefficients);
        }

        //-------------------------------------------------
        //      ACCESSOR(S) AND MUTATOR(S) METHOD(S)
        //-------------------------------------------------
        public int getDegree() {
                return coefficients.length - 1;
```

```
}

public double[] getCoefficients() {
	return coefficients.clone();
}

//-------------------------------------------------
//	OTHER METHOD(S)
//-------------------------------------------------
public double calcY(double x) {
	Validate.checkNotNull(x);
	return evaluate(coefficients, x);
}
public double[] calcY(double[] x) {
	Validate.checkNotNull(x);
	Validate.checkNotEmpty(x);
	double[] copy = new double[x.length];
	for (int i = 0; i < x.length; i++)
		copy[i] = calcY(x[i]);
	return copy;
}

private static double evaluate(double[] coefficients, double argument) throws
	EmptyArrayArgumentException, NullArgumentException {
	Validate.checkNotNull(coefficients);
	Validate.checkNotEmpty(coefficients);
	int n = coefficients.length;
	double result = coefficients[n - 1];
	for (int j = n - 2; j >= 0; j--) {
		result = argument*result + coefficients[j];
	}
	return result;
}

@Override
public String toString() {
	String s = "";
	boolean first = true;
	double[] coefficients = this.getCoefficients();

	for (int j = 0; j < coefficients.length; j++) {
		if (coefficients[j] == 0.0)
			;
		else {
			if		(coefficients[j] < 0.0)
				s += "- " + String.format("%.2f",
			Math.abs(coefficients[j]));
			else if (coefficients[j] > 0.0 && !first)
				s += "+ " + String.format("%.2f",
			Math.abs(coefficients[j]));
			else if (coefficients[j] > 0.0 && first)
				s += String.format("%.2f", Math.abs(coefficients[j]));

			if (j != 0 && j != 1)
				s += String.format(" x^%d ", j);
			else if (j == 1)
				s += " x ";
			else
				s += " ";
			first = false;
		}
	}
	return s;
}

//-------------------------------------------------
//	CLONE METHOD
//-------------------------------------------------
@Override
public PolynomialFunction clone() {
	return new PolynomialFunction(this);
```

```
	}
}
```

B.15 numerical.analysis : PolynomialSplineFunction.java

```
package numerical.analysis;

import java.util.*;

import validation.InvalidArgumentException;
import validation.NullArgumentException;
import validation.OutOfRangeArgumentException;
import validation.Validate;

/** This class aims to extend the concept of a function
 *  through the Function interface to polynomial spline
 *  functions along with additional instance variables
 *  and methods. This class also uses composition by
 *  having an array of class type PolynomialFunction.
 */
public class PolynomialSplineFunction extends PolynomialFunction {

	//------------------------------------------------
	//	INSTANCE VARIABLE(S)
	//------------------------------------------------
	/* Spline segment interval delimiters (knots).
	 * Size is n + 1 for n segments.
	 */
	private final double knots[];
	/* The polynomial functions that make up the spline.  The first element
	 * determines the value of the spline over the first subinterval, the
	 * second over the second, etc. Spline function values are determined by
	 * evaluating these functions at (x - knot[i]) where i is the
	 * knot segment to which x belongs.
	 */
	private final PolynomialFunction polynomials[];
	/* Number of spline segments. It is equal to the number of polynomials and
	 * to the number of partition points - 1.
	 */
	private final int n;

	//------------------------------------------------
	//	CONSTRUCTOR(S)
	//------------------------------------------------
	/* Construct a polynomial spline function with the given segment delimiters
	 * and interpolating polynomials.
	 * The constructor copies both arrays and assigns the copies to the knots
	 * and polynomials properties, respectively.
	 */
	public PolynomialSplineFunction() {
		this.n = 0;
		this.knots = null;
		this.polynomials = new PolynomialFunction[1];
	}
	public PolynomialSplineFunction(double knots[], PolynomialFunction polynomials[])
		throws NullArgumentException, InvalidArgumentException {
		if (knots == null || polynomials == null) {
			throw new NullArgumentException();
		}
		if (knots.length < 2) {
			throw new InvalidArgumentException("Number of knots too small! You
		entered: " + knots.length + ". Required: >= 2");
		}
		if (knots.length - 1 != polynomials.length) {
			throw new InvalidArgumentException("Dimension mismatch! Number of knots -
		1 should be equal to "
					+ "number of polynomials. You entered number of knots-1: "
		+ (knots.length - 1) + "and number of polynomials:" + polynomials.length);
		}
```

```
        this.n = knots.length - 1;
        this.knots = new double[n + 1];
        System.arraycopy(knots, 0, this.knots, 0, n + 1);
        this.polynomials = new PolynomialFunction[n];
        System.arraycopy(polynomials, 0, this.polynomials, 0, n);
}
public PolynomialSplineFunction(PolynomialSplineFunction pf) {
        this(pf.knots, pf.polynomials);
}

//-----------------------------------------------
//      ACCESSOR(S) AND MUTATOR(S) METHOD(S)
//-----------------------------------------------
/* Get the number of spline segments.
 * It is also the number of polynomials and the number of knot points - 1.
 */
public int getN() {
        return n;
}

/* Get a copy of the interpolating polynomials array.
 * It returns a fresh copy of the array. Changes made to the copy will
 * not affect the polynomials property.
 */
public PolynomialFunction[] getPolynomials() {
        PolynomialFunction p[] = new PolynomialFunction[n];
        System.arraycopy(polynomials, 0, p, 0, n);
        return p;
}

/* Get an array copy of the knot points.
 * It returns a fresh copy of the array. Changes made to the copy
 * will not affect the knots property.
 */
public double[] getKnots() {
        double out[] = new double[n + 1];
        System.arraycopy(knots, 0, out, 0, n + 1);
        return out;
}

//-----------------------------------------------
//      OTHER METHOD(S)
//-----------------------------------------------
/* Implement the calcY methods enforced from the Function interface
 * for both double x and double[] x argument types
 */
public double calcY(double x) {
        Validate.checkNotNull(x);
        if (x < knots[0] || x > knots[n]) {
                throw new OutOfRangeArgumentException("calcY argument", x, knots[0],
        knots[n]);
        }
        int i = Arrays.binarySearch(knots, x);
        if (i < 0) {
                i = -i - 2;
        }
        // This will handle the case where x is the last knot value
        // There are only n-1 polynomials, so if x is the last knot
        // then we will use the last polynomial to calculate the value.
        if ( i >= this.polynomials.length ) {
                i--;
        }
        return this.polynomials[i].calcY(x - knots[i]);
}
public double[] calcY(double[] x) {
        Validate.checkNotNull(x);
        Validate.checkNotEmpty(x);
        double[] v = new double[x.length];
        for (int i = 0; i < v.length; i++)
                v[i] = calcY(x[i]);
        return v;
```

```
	}

	// Indicates whether a point is within the interpolation range.
	public boolean isValidPoint(double x) {
		Validate.checkNotNull(x);
		if (x < knots[0] || x > knots[n])
			return false;
		else
			return true;
	}

	/* Implement overridden method toString to handle the use of
	 * System.out.prinln() when argument is of this class type
	 */
	@Override
	public String toString() {
		String s = "";
		PolynomialFunction[] polynomials = this.getPolynomials();
		for (int i = 0; i < polynomials.length; i++) {
			s = s + "\n";
			s = s + polynomials[i].toString();
		}
		return s;
	}
	// Alternative toString method which selectively converts only selected polynomial
	public String toString(int iPolynomial) {
		PolynomialFunction[] polynomials = this.getPolynomials();
		return polynomials[iPolynomial].toString();
	}

	//-------------------------------------------------
	//	CLONE METHOD
	//-------------------------------------------------
	@Override
	public PolynomialSplineFunction clone() {
		return new PolynomialSplineFunction(this);
	}
}
```

B.16 numerical.interpolation : Interpolator.java

```
package numerical.interpolation;

import numerical.analysis.*;

/** This interface aims to define the signature required
 *  for each interpolator classes which will implement their
 *  own interpolation algorithm through the interpolate
 *  method (SplineInterpolator and PchipInterpolator).
 */
public interface Interpolator
{
	//-------------------------------------------------
	//	OTHER METHOD(S)
	//-------------------------------------------------
	/* Method that uses the x and y array vectors and implements the appropriate
	 * interpolation algorithms in the implemented classes.
	 */
	Function interpolate(double xval[], double yval[]);
	//throws MathIllegalArgumentException, DimensionMismatchException;
}
```

B.17 numerical.interpolation : PchipInterpolator.java

```
package numerical.interpolation;

import numerical.analysis.*;
import validation.EmptyArrayArgumentException;
import validation.InvalidArgumentException;
import validation.NullArgumentException;
```

```
import validation.Validate;

/** This class aims to implement the shape-preserving cubic
 *  Hermite interpolant heavily inspired by the MATLAB
 *  implemented function 'pchip' (for more information
 *  refer to: https://www.mathworks.com/moler/interp.pdf)
 *  enforced by the Interpolator  interface (through the
 *  interpolate method) which will then return a
 *  PolynomialSplineFunction class type.
 */
public class PchipInterpolator implements Interpolator
{
	//-------------------------------------------------
	//	CONSTRUCTOR(S)
	//-------------------------------------------------
	// empty constructor
	public PchipInterpolator() {
	}
	// copy constructor
	public PchipInterpolator(PchipInterpolator pci) {
		this();
	}

	//-------------------------------------------------
	//	OTHER METHOD(S)
	//-------------------------------------------------
	// implementation of the interpolate method from Interpolator interface
	public PolynomialSplineFunction interpolate(double x[], double y[]) throws
		EmptyArrayArgumentException, NullArgumentException, InvalidArgumentException {
		Validate.checkNotNull(x);
		Validate.checkNotNull(y);
		Validate.checkNotEmpty(x);
		Validate.checkNotEmpty(y);

		if (x.length != y.length) {
			throw new InvalidArgumentException("Dimension mismatch. Both data arrays x
			and y must be the same length!");
		}

		if (x.length < 3) {
			throw new InvalidArgumentException("Minimum number of data points: 3. You
			entered: " + x.length);
		}
		// number of data points
		final int n = x.length;

		// number of intervals. The number of data points is n + 1
		final int numberOfIntervals = n - 1;

		// differences between knot points of x[] and y[]
		double[] h = new double[numberOfIntervals];
		double[] hy = new double[numberOfIntervals];
		h = Maths.diff(x);
		hy = Maths.diff(y);

		// element-by-element division of diff(x) and diff(y)
		final double[] del = new double[numberOfIntervals];
		for (int i = 0; i < numberOfIntervals; i++) {
			del[i] = hy[i]/h[i];
		}

		// compute slopes
		double[] slopes = pchipslopes(x, y, h, del);

		// compute piecewise Hermite interpolant to those values and slopes
		final double[] dzzdx = new double[numberOfIntervals];
		final double[] dzdxdx = new double[numberOfIntervals];
		for (int i = 0; i < numberOfIntervals; i++) {
			dzzdx[i] = (del[i] - slopes[i])/h[i];
		}
		for (int i = 0; i < numberOfIntervals; i++) {
```

```
                dzdxdx[i] = (slopes[i + 1] - del[i])/h[i];
        }

        //
        final PolynomialFunction polynomials[] = new
        PolynomialFunction[numberOfIntervals];
        final double coefficients[] = new double[4];
        for (int i = 0; i < numberOfIntervals; i++) {
                coefficients[0] = y[i];
                coefficients[1] = slopes[i];
                coefficients[2] = 2*dzzdx[i] - dzdxdx[i];
                coefficients[3] = (dzdxdx[i] - dzzdx[i])/h[i];
                polynomials[i] = new PolynomialFunction(coefficients);
        }

        return new PolynomialSplineFunction(x, polynomials);
}

// internal private method that is used exclusively in the interpolate method
private double[] pchipslopes(double[] x, double[] y, double[] h, double[] del) {

        // start slope algorithm portion
        int n = x.length;
        int numberOfIntervals = n - 1;

        final double[] slopes = new double[n];

        // special case n = 2, use linear interpolation
        if (n == 2) {
                for (int i = 0; i < numberOfIntervals; i++)
                        slopes[i] = del[0];
                return slopes;
        }

        // find indexes that have del(k-1) and del(k) when they have the same sign
        int[] k = new int[numberOfIntervals - 1];
        int kindex = 0;
        for (int i = 0; i < k.length; i++) {
                if (del[i]*del[i+1] > 0) {
                        k[kindex] = i;
                        if (i != k.length - 1)
                                kindex++;
                }
        }

        // redim k[] to length of kindex + 1
        int[] temp = new int[kindex + 1];
        System.arraycopy(k, 0, temp, 0, temp.length);
        k = temp;

        // compute slopes at interior points
        double[] hs = new double[k.length];
        double[] w1 = new double[k.length];
        double[] w2 = new double[k.length];
        double[] dmax = new double[k.length];
        double[] dmin = new double[k.length];
        for (int i : k) {
                hs[i] = h[i] + h[i + 1];
                w1[i] = (h[i] + hs[i])/(3*hs[i]);
                w2[i] = (h[i + 1] + hs[i])/(3*hs[i]);
                dmax[i] = Math.max(Math.abs(del[i]), Math.abs(del[i + 1]));
                dmin[i] = Math.min(Math.abs(del[i]), Math.abs(del[i + 1]));
                slopes[i + 1] = dmin[i]/(w1[i]*del[i]/dmax[i] + w2[i]*del[i + 1]/dmax[i]);
        }

        // compute slopes at end points
        slopes[0] = ((2*h[0]+h[1])*del[0] - h[0]*del[1])/(h[0]+h[1]);
        if (Math.signum(slopes[0]) != Math.signum(del[0]))
                slopes[0] = 0.0;
        else if ((Math.signum(del[0]) != Math.signum(del[1])) && (Math.abs(slopes[0]) >
        Math.abs(3*del[0])))
```

```
                        slopes[0] = 3*del[0];

                slopes[n - 1] = ((2*h[n - 2] + h[n - 3])*del[n - 2] - h[n - 2]*del[n - 3])/(h[n -
                2]+h[n - 3]);
                if ((Math.signum(slopes[n - 1]) != Math.signum(del[n - 2])))
                        slopes[n - 1] = 0;
                else if ((Math.signum(del[n - 2]) != Math.signum(del[n - 3])) &&
                (Math.abs(slopes[n - 1]) > Math.abs(3*del[n - 2])))
                        slopes[n - 1] = 3*del[n - 2];

                return slopes;
        }

        //-------------------------------------------------
        //      CLONE METHOD
        //-------------------------------------------------
        @Override
        public PchipInterpolator clone() {
                return new PchipInterpolator(this);
        }
}
```

B.18 process.flow : BottomsStream.java

```
package process.flow;

import excelextractor.*;

public class BottomsStream extends Stream
{
        //-------------------------------------------------
        //      CONSTRUCTOR(S)
        //-------------------------------------------------
        public BottomsStream() {
                super();
        }
        public BottomsStream(double flowRate, double[] compFraction, double temperature, double
        latentHeat, double[] compBP, double[] compCP, double unitCost) {
                super(flowRate, compFraction, temperature, latentHeat, compBP, compCP, unitCost);
        } // end of BottomsStream empty constructor

        public BottomsStream(ExcelExtractor ee) {
                super(0.0, ee.getBottomsFraction(), ee.getTemperature(), ee.getLatentHeat(),
        ee.getNormalBP(), ee.getHeatCapacity(), ee.getSaleBottoms());
        } //end of BottomsStream constructor

        public BottomsStream(BottomsStream bs) {
                this(bs.getFlowRate(), bs.getCompFraction(), bs.getTemperature(),
        bs.getLatentHeat(), bs.getCompBP(), bs.getCompCP(), bs.getUnitCost());
        } // end of BottomsStream copy constructor

        //-------------------------------------------------
        //      OTHER METHOD(S)
        //-------------------------------------------------
        public double calcQ() {
                return -9999.0; // return dummy value, q is not required for bottoms stream
        }

        //-------------------------------------------------
        //      CLONE METHOD
        //-------------------------------------------------
        public BottomsStream clone() {
                return new BottomsStream(this);
        }
}
```

B.19 process.flow : DistillateStream.java

```
package process.flow;
```

```
import excelextractor.*;

public class DistillateStream extends Stream {
	//-------------------------------------------------
	//	CONSTRUCTOR(S)
	//-------------------------------------------------
	public DistillateStream() {
		super();
	}
	public DistillateStream(double flowRate, double[] compFraction, double temperature,
	double latentHeat, double[] compBP, double[] compCP, double unitCost) {
		super(flowRate, compFraction, temperature, latentHeat, compBP, compCP, unitCost);
	} // end of Distillate constructor

	public DistillateStream(ExcelExtractor ee) {
		super(0.0, ee.getDistillateFraction(), ee.getTemperature(), ee.getLatentHeat(),
	ee.getNormalBP(), ee.getHeatCapacity(), ee.getSaleDistillate());
	} //end of FeedStream constructor

	public DistillateStream(DistillateStream ds) {
		this(ds.getFlowRate(), ds.getCompFraction(), ds.getTemperature(),
	ds.getLatentHeat(), ds.getCompBP(), ds.getCompCP(), ds.getUnitCost());
	} // end of DistillateStream copy constructor

	//-------------------------------------------------
	//	OTHER METHOD(S)
	//-------------------------------------------------
	public double calcQ() {
		return -9999; // return dummy value, no q is needed for distillate stream
	}

	//-------------------------------------------------
	//	CLONE METHOD
	//-------------------------------------------------

	public DistillateStream clone() {
		return new DistillateStream(this);
	}
}
```

B.20 process.flow : FeedStream.java

```
package process.flow;

import excelextractor.*;

public class FeedStream extends Stream
{
	//-------------------------------------------------
	//	CONSTRUCTOR(S)
	//-------------------------------------------------
	public FeedStream() {
		super();
	} //end of empty constructor

	public FeedStream(double flowRate, double[] compFraction, double temperature, double
	latentHeat, double[] compBP, double[] compCP, double unitCost) {
		super(flowRate, compFraction, temperature, latentHeat, compBP, compCP, unitCost);
	}

	public FeedStream(ExcelExtractor ee) {
		super(ee.getFeedFlowRate(), ee.getFeedFraction(), ee.getTemperature(),
	ee.getLatentHeat(), ee.getNormalBP(), ee.getHeatCapacity(), ee.getCostRaw());
	} //end of default constructor

	public FeedStream(FeedStream fs) {
		this(fs.getFlowRate(), fs.getCompFraction(), fs.getTemperature(),
	fs.getLatentHeat(), fs.getCompBP(), fs.getCompCP(), fs.getUnitCost());
	} // end of copy constructor
```

```
        //-------------------------------------------------
        //      CALC METHOD(S)
        //-------------------------------------------------
        public double calcQ() {
                return (getLatentHeat()+calcStreamCP()*(calcStreamBP()-
                        getTemperature()))/getLatentHeat();
        }

        //-------------------------------------------------
        //      CLONE METHOD
        //-------------------------------------------------
        public FeedStream clone() {
                return new FeedStream(this);
        } //end of clone method
} // end
 of FeedStream class
```

B.21 process.flow : Stream.java

```
package process.flow;

import validation.*;

public abstract class Stream {
        //-------------------------------------------------
        //      INSTANCE VARIABLE(S)
        //-------------------------------------------------
        protected double flowRate;
        protected double[] compFraction;
        protected double[] compBP;
        protected double[] compCP;
        protected double temperature;
        protected double latentHeat;
        protected double unitCost;

        //-------------------------------------------------
        //      CONSTRUCTOR(S)
        //-------------------------------------------------
        public Stream() {
                this.setFlowRate(0.0);
                this.setCompFraction(null);
                this.setCompCP(null);
                this.setCompBP(null);
                this.setTemperature(0.0);
                this.setLatentHeat(0.0);
                this.setUnitCost(0.0);
        } // end of empty constructor

        public Stream(double flowRate, double[] compFraction, double temperature, double
        latentHeat, double[] compBP, double[] compCP, double unitCost) {
                this.setFlowRate(flowRate);
                this.setCompFraction(compFraction);
                this.setCompBP(compBP);
                this.setCompCP(compCP);
                this.setTemperature(temperature);
                this.setLatentHeat(latentHeat);
                this.setUnitCost(unitCost);
        } // end of default constructor

        public Stream(Stream stream) {
                this(stream.flowRate, stream.compFraction, stream.temperature, stream.latentHeat,
                stream.compBP, stream.compCP, stream.unitCost);
        } //end of copy constructor

        //-------------------------------------------------
        //      ACCESSOR AND MUTATOR METHOD(S)
        //-------------------------------------------------
        public double getFlowRate() {
                return this.flowRate;
        } //end of default getter
```

```
public void setFlowRate(double flowRate) throws InvalidArgumentException {
	if (Validate.isNonNegative(flowRate))
		this.flowRate = flowRate;
	else
		throw new InvalidArgumentException();
} //end of default setter

public double [] getCompFraction() {
	double[] copyCompFraction = new double[this.compFraction.length];
	for (int i=0; i<this.compFraction.length; i++)
		copyCompFraction[i] = this.compFraction[i];
	return copyCompFraction;
} //end of default getter
public void setCompFraction(double[] compFraction) {
	if (Validate.isNonNegative(compFraction) && Validate.isLessThan1(compFraction)) {
		this.compFraction = new double[compFraction.length];
		for (int i = 0; i < compFraction.length; i++)
			this.compFraction[i] = compFraction[i];
	} else throw new InvalidArgumentException();
} //end of default setter

public double getTemperature() {
	return this.temperature;
} //end of default getter
public void setTemperature(double temperature) {
	if (Validate.isNonNegative(temperature))
		this.temperature = temperature;
	else
		throw new InvalidArgumentException();
} //end of default setter

public double getLatentHeat() {
	return this.latentHeat;
} //end of default getter
public void setLatentHeat(double latentHeat) {
	if (Validate.isNonNegative(latentHeat))

	this.latentHeat = latentHeat;
	else
		throw new InvalidArgumentException();
} //end of default setter

public double[] getCompBP() {
	double[] copy = new double[this.compBP.length];
	for (int i=0; i<this.compBP.length; i++)
		copy[i] = this.compBP[i];
	return copy;
} //end of default getter
public void setCompBP(double[] compBP) {
	if (Validate.isNonNegative(compBP)) {
		this.compBP = new double[compBP.length];
		for (int i = 0; i < compBP.length; i++)
			this.compBP[i] = compBP[i];
	} else throw new InvalidArgumentException();
} //end of default setter

public double[] getCompCP() {
	double[] copy = new double[this.compCP.length];
	for (int i=0; i<this.compCP.length; i++) {
		copy[i] = this.compCP[i];
	}
	return copy;

} //end of default getter
public void setCompCP(double[] compCP) {
	if (Validate.isNonNegative(compCP)) {
		this.compCP = new double[compCP.length];
		for (int i = 0; i < compCP.length; i++)
			this.compCP[i] = compCP[i];
	} else throw new InvalidArgumentException();
} //end of default setter
```

```
        public double getUnitCost() {
                return this.unitCost;
        } //end of default getter
        public void setUnitCost(double unitCost) {
                if (Validate.isNonNegative(unitCost))
                        this.unitCost = unitCost;
                else
                        throw new InvalidArgumentException();
        } //end of default setter

        //-------------------------------------------------
        //      OTHER METHODS
        //-------------------------------------------------
        public double calcStreamBP() {
                double sum = 0.;
                for (int i = 0; i< this.compFraction.length; i++)
                        sum = sum + this.compFraction[i]*this.compBP[i];
                return sum;
        }
        public double calcStreamCP() {
                double sum = 0.;
                for (int i = 0; i< this.compFraction.length; i++)
                        sum = sum + this.compFraction[i]*this.compCP[i];
                return sum;
        }
        public abstract double calcQ();

        //-------------------------------------------------
        //      CLONE METHOD
        //-------------------------------------------------
        public abstract Stream clone(); //end of abstract clone signature

} // end of abstract stream parent class
```

B.22 process.unitoperation : Column.java

```
package process.unitoperation;

import process.flow.*;
import validation.*;

public class Column {

        //-------------------------------------------------
        //      INSTANCE VARIABLES(S)
        //-------------------------------------------------
        private double diameter;
        private double length;
        private double gaugePressure;
        private String material;
        private Stream[] streams;

        //-------------------------------------------------
        //      CONSTRUCTOR(S)
        //-------------------------------------------------
        public Column() {
                this.diameter = 0.0;
                this.length = 0.0;
                this.gaugePressure = 0.0;
                this.material = "";
                this.streams = null;
        } //end of empty constructor

        public Column(double diameter, double length, double gaugePressure, String material,
        Stream[] streams) {
                this.setDiameter(diameter);
                this.setLength(length);
                this.setGaugePressure(gaugePressure);
                this.setMaterial(material);
```

```
                this.setStreams(streams);

        } //end of overloaded constructor

        public Column(Column column) {
                this(column.diameter, column.length, column.gaugePressure, column.material,
        column.streams);
        } //end of copy constructor

        //-------------------------------------------------
        //      ACCESSOR(S) and/or MUTATOR(S)
        //-------------------------------------------------
        // diameter accessor and mutator
        public double getDiameter()   { return this.diameter; }// end of diameter getter
        public void setDiameter(double diameter) {
                if(Validate.isNonNegative(diameter)) {
                        this.diameter = diameter;
                } else throw new InvalidArgumentException();
        }// end of diameter setter

        // length accessor and mutator
        public double getLength(){ return this.length; }// end of length getter
        public void setLength(double length) {
                if(Validate.isNonNegative(length)) {
                        this.length = length;
                } else throw new InvalidArgumentException();
        }// end of length setter

        public double getGaugePressure() { return this.gaugePressure; } //end of getter
        public void setGaugePressure(double gaugePressure){
                if(Validate.isNonNegative(gaugePressure)) {
                        this.gaugePressure = gaugePressure;
                } else throw new InvalidArgumentException();
        }// end of setter

        // material accessor and mutator
        public String getMaterial()   { return this.material; } //end of material getter
        public void setMaterial(String material)
{ this.material = material; } //end of material setter

        // Stream array accessor and mutator
        public Stream[] getStreams() {
                Stream[] copy = new Stream[this.streams.length];
                for (int i = 0; i < this.streams.length; i++)
                        copy[i] = this.streams[i].clone();
                return copy;
        } //end of streams getter
        public void setStreams(Stream[] streams) {
                this.streams = new Stream[streams.length];
                for (int i = 0; i < this.streams.length; i++)
                        this.streams[i] = streams[i].clone();
        } //end of streams setter

        //-------------------------------------------------
        //      OTHER METHOD(S)
        //-------------------------------------------------
        // this method calculates the flow rate of the a specific stream in kg/hr
        public double[] calcFlowRates() {
                this.streams[2].setFlowRate(this.streams[0].getFlowRate()*
                (this.streams[0].getCompFraction()[0]
                        - this.streams[1].getCompFraction()[0])/
                (this.streams[2].getCompFraction()[0]
                        - this.streams[1].getCompFraction()[0]));
                this.streams[1].setFlowRate(this.streams[0].getFlowRate()
                - this.streams[2].getFlowRate());
                return new double[]{this.streams[0].getFlowRate(), this.streams[1].getFlowRate(),
                        this.streams[2].getFlowRate()};
        } // end of calcFlowRates method

        // this method calculates the profit of the distillation column strictly based on feed
// cost, and distillate and bottoms price
```

```
        public double calcProfit() {
                return (this.streams[1].getUnitCost()*this.calcFlowRates()[1]
                                + this.streams[2].getUnitCost()*this.calcFlowRates()[2]
                        - this.streams[0].getUnitCost()*this.streams[0].getFlowRate())*24*365;
        } // end of calcProft method

        //-------------------------------------------------
        //      CLONE METHOD
        //-------------------------------------------------
        public Column clone() {
                return new Column(this);
        } // end of clone method

} //end of Column class
```

B.23 process.unitoperation : DistillationColumn.java

```
package process.unitoperation;

import java.util.Arrays;

import excelextractor.ExcelExtractor;
import numerical.analysis.*;
import numerical.interpolation.*;
import process.flow.*;
import validation.*;

public class DistillationColumn extends Column implements McCabeThiele{

        //-------------------------------------------------
        //      INSTANCE VARIABLES
        //-------------------------------------------------
        private double refluxRatio;
        private LinearFunction[] lines;
        private Tray[] trays;
        private int numberOfTrays;

        //-------------------------------------------------
        //      CONSTRUCTOR(S)
        //-------------------------------------------------
        public DistillationColumn() {
                super();
                this.refluxRatio = 0.0;
                this.numberOfTrays = 0;
                this.trays = null;
                this.lines = null;
        } //end of empty constructor

        public DistillationColumn(double diameter, double length, double gaugePressure, String
material, Stream[] streams, double refluxRatio, LinearFunction[] lines, Tray[] trays, int
numberOfTrays) {
                super(diameter, length, gaugePressure, material, streams);
                this.setRefluxRatio(refluxRatio);
                this.setTrays(trays);
                this.setLines(lines);
                this.setNumberOfTrays(numberOfTrays);
        } //end of default constructor

        public DistillationColumn(ExcelExtractor ee) {
                super(ee.getDiameter(), ee.getLength(), ee.getGaugePressure(), ee.getMocColumn(),
                new Stream[]{new FeedStream(ee),
        new DistillateStream(ee), new BottomsStream(ee)});
                this.setRefluxRatio(ee.getRefluxRatio());
                this.setLines(new LinearFunction[]{new EnrichingLine(this),
                        new QLine(this), new StrippingLine()});
                //cannot simultaneously call setLines and StrippingLine(this) initially since
                //lines[0] and lines[0] are null and hence StrippingLine.calcSlope will crash
                this.setLines(new LinearFunction[]{new EnrichingLine(this), new QLine(this),
                new StrippingLine(this)});
```

```
        if (Arrays.asList("sieve tray","sieve","sieve
        trays").contains(ee.getTrayType().toLowerCase())) {
                this.trays = new SieveTray[this.calcNumberOfTrays(new
                PchipInterpolator().interpolate(ee.getEqX(),ee.getEqY()))];
                for (int i = 0; i < this.trays.length; i++)
                        this.trays[i] = new SieveTray(ee);
        }
        this.setNumberOfTrays(this.calcNumberOfTrays(new
        PchipInterpolator().interpolate(ee.getEqX(),ee.getEqY())));
} //end of overloaded constructor

public DistillationColumn(DistillationColumn column) {
        this(column.getDiameter(), column.getLength(), column.getGaugePressure(),
        column.getMaterial(),
                        column.getStreams(), column.getRefluxRatio(), column.getLines(),
                column.getTrays(), column.getNumberOfTrays());
} //end of copy constructor

//-------------------------------------------------
//      ACCESSORS AND MUTATORS
//-------------------------------------------------
public double getRefluxRatio()
{ return this.refluxRatio; } //end of default getter
public void setRefluxRatio(double refluxRatio) {
        if (Validate.isNonNegative(refluxRatio))
                this.refluxRatio = refluxRatio;
        else throw new InvalidArgumentException();
        } //end of default setter

public int getNumberOfTrays()
{ return this.numberOfTrays; } //end of default getter
public void setNumberOfTrays(int numberOfTrays)      {
        if (Validate.isNonNegative(numberOfTrays))
                this.numberOfTrays = numberOfTrays;
        else throw new InvalidArgumentException();
} //end of default setter

public Tray[] getTrays() {
        Tray[] copy = new Tray[this.trays.length];
        for (int i = 0; i < this.trays.length; i++)
                copy[i] = this.trays[i];
        return copy;
} //end of default getter
public void setTrays(Tray[] trays) {
        this.trays = new Tray[trays.length];
        for (int i = 0; i < this.trays.length; i++)
                this.trays[i] = trays[i].clone();
} //end of default setter

public LinearFunction[] getLines() {
        LinearFunction[] copy = new LinearFunction[this.lines.length];
        for (int i = 0; i < this.lines.length; i++)
                copy[i] = this.lines[i].clone();
        return copy;
} //end of default getter
public void setLines(LinearFunction[] lines) {
        this.lines = new LinearFunction[lines.length];
        for (int i = 0; i < this.lines.length; i++)
                this.lines[i] = lines[i].clone();
} //end of default setter

//-------------------------------------------------
//      OTHER METHOD(S)
//-------------------------------------------------
/** This method calculates the number of trays by using the McCabe-Thiele
 * method by stepping between the binary equilibrium curve calculated
 * using the spline function and the stripping and enriching lines.
 */
public int calcNumberOfTrays(PolynomialSplineFunction equilibrium) {
        double x = getStreams()[2].getCompFraction()[0];
        double y = x;
```

```
                double yIntersection =
                LinearFunction.calcIntersection(this.getLines()[0],this.getLines()[1])[1];

                int nTrays = 0;

                do {
                        y = equilibrium.calcY(x);

                        if (y >= this.getStreams()[1].getCompFraction()[0]) {x = y;}
                        else if (y >= yIntersection)
                        {x = lines[0].calcX(y);} //EnrichingLine
                        else
                {x = lines[2].calcX(y);} //StrippingLine

                        nTrays++;
                } while (x <= this.getStreams()[1].getCompFraction()[0]);

                return nTrays;
        }

        //-------------------------------------------------
        //      CLONE METHOD
        //-------------------------------------------------
        public DistillationColumn clone() {
                return new DistillationColumn(this);
        } //end of clone method

} //end of DistillationColumn class
```

B.24 process.unitoperation : EnrichingLine.java

```
package process.unitoperation;

import numerical.analysis.LinearFunction;
import process.flow.Stream;

/**EnrichingLine as a child of LinearFunction and implements the interface OperatingLine
 *
 */
public class EnrichingLine extends LinearFunction implements OperatingLine {

        //-------------------------------------------------
        //      CONSTRUCTOR(S)
        //-------------------------------------------------
        public EnrichingLine() {
                super();
        } //end of empty

        public EnrichingLine(EnrichingLine ol) {
                super(ol);
        } //end default constructor

        public EnrichingLine(DistillationColumn column) {
                /**Check that column indeed has assigned streams
             * for calcSlope and calcIntercept else NULLERROR!
                 */
                super();
                setSlope(calcSlope(column));
                setIntercept(calcIntercept(column));
        } //end of overloaded constructor

        public EnrichingLine(Stream[] streams, double refluxRatio) {
                super();
                setSlope(calcSlope(refluxRatio));
                setIntercept(calcIntercept(streams, refluxRatio));
        } //end of overloaded constructor
        //-------------------------------------------------
        //      OTHER METHOD(S)
        //-------------------------------------------------
        // method for slope calculation
```

```
        public double calcSlope(DistillationColumn column) {
                return (column.getRefluxRatio()/(column.getRefluxRatio() + 1));
        }
        // method for y-intercept calculation
        public double calcIntercept(DistillationColumn column) {
                return column.getStreams()[1].getCompFraction()[0]/(column.getRefluxRatio() + 1);
        }
        // method for slope calculation
        public double calcSlope(double refluxRatio) {
                return (refluxRatio/(refluxRatio + 1));
        }
        // method for y-intercept calculation
        public double calcIntercept(Stream[] streams, double refluxRatio) {
                return streams[1].getCompFraction()[0]/(refluxRatio + 1);
        }

        //-------------------------------------------------
        //      CLONE METHOD
        //-------------------------------------------------
        public EnrichingLine clone() {
                return new EnrichingLine(this);
        }
}
```

B.25 process.unitoperation : StrippingLine.java

```
package process.unitoperation;

import numerical.analysis.*;
import process.flow.Stream;

/**Child class of LinearFunction and implements OperatingLine interface
 *
 */
public class StrippingLine extends LinearFunction implements OperatingLine {

        //-------------------------------------------------
        //      CONSTRUCTOR(S)
        //-------------------------------------------------
        public StrippingLine() {
                super();
        }
        public StrippingLine(StrippingLine ol) {
                super(ol);
        }
        public StrippingLine(DistillationColumn column) {
                super();
                setSlope(calcSlope(column));
                setIntercept(calcIntercept(column));
        }
        public StrippingLine(Stream[] streams, LinearFunction[] lines) {
                super();
                setSlope(calcSlope(streams, lines));
                setIntercept(calcIntercept(streams, lines));
        }
        //-------------------------------------------------
        //      OTHER METHOD(S)
        //-------------------------------------------------
        //column.getLines()[] = {EnrichingLine, QLine, StrippingLine}
        public double calcSlope(DistillationColumn column) {
                return (calcIntersection(column.getLines()[0],column.getLines()[1])[1] -
                        column.getStreams()[2].getCompFraction()[0])
                                /(calcIntersection(column.getLines()[0],column.getLines()[1])[0] -
                        column.getStreams()[2].getCompFraction()[0]);
        }
        public double calcIntercept(DistillationColumn column) {
                return calcIntersection(column.getLines()[0],column.getLines()[1])[1] -

        this.getSlope()*calcIntersection(column.getLines()[0],column.getLines()[1])[0];
        }
```

```
        public double calcSlope(Stream[] streams, LinearFunction[] lines) {
                return (calcIntersection(lines[0],lines[1])[1] - streams[2].getCompFraction()[0])
                                /(calcIntersection(lines[0],lines[1])[0] -
                        streams[2].getCompFraction()[0]);
        }
        public double calcIntercept(Stream[] streams, LinearFunction[] lines) {
                return calcIntersection(lines[0],lines[1])[1] -
                                this.getSlope()*calcIntersection(lines[0],lines[1])[0];
        }

        //-------------------------------------------------
        //      CLONE METHOD
        //-------------------------------------------------
        public StrippingLine clone() {
                return new StrippingLine(this);
        }
}
```

B.26 process.unitoperation : QLine.java

```
package process.unitoperation;
import process.flow.Stream;

import numerical.analysis.LinearFunction;

// child class of LinearFunction and implements OperatingLine interface
public class QLine extends LinearFunction implements OperatingLine {

        //-------------------------------------------------
        //      CONSTRUCTOR(S)
        //-------------------------------------------------
        public QLine(DistillationColumn column) {
                super();
                setSlope(calcSlope(column));
                setIntercept(calcIntercept(column));
        }
        public QLine() {
                super();
        }
        public QLine(Stream[] streams) {
                super();

        }

        public QLine(QLine ol) {
                super(ol);
        }

        //-------------------------------------------------
        //      OTHER METHOD(S)
        //-------------------------------------------------
        // calculate slope method overriding calcSlope in OperatingLine
        public double calcSlope(DistillationColumn column) {
                return column.getStreams()[0].calcQ()/(column.getStreams()[0].calcQ() - 1);
        }
        // calculate intercept method overriding calcIntercept in OperatingLine
        public double calcIntercept(DistillationColumn column) {
                return -
column.getStreams()[0].getCompFraction()[0]/(column.getStreams()[0].calcQ() - 1);
        }
        // calculate slope method overriding calcSlope in OperatingLine
        public double calcSlope(Stream[] streams) {
                return streams[0].calcQ()/(streams[0].calcQ() - 1);
        }
        // calculate intercept method overriding calcIntercept in OperatingLine
        public double calcIntercept(Stream[] streams) {
                return -streams[0].getCompFraction()[0]/(streams[0].calcQ() - 1);
        }

        //-------------------------------------------------
```

```
        //      CLONE METHOD
        //-------------------------------------------------
        public QLine clone() {
                return new QLine(this);
        }
}
```

B.27 process.unitoperation : OperatingLine.java

```
package process.unitoperation;

// interface implemented in EnrichingLine, QLine and StrippingLine
interface OperatingLine {

        //-------------------------------------------------
        //      OTHER METHOD(S)
        //-------------------------------------------------
        // method for slope and y-intercept calculation to be implemented
        double calcSlope(DistillationColumn column);
        double calcIntercept(DistillationColumn column);
}
```

B.28 process.unitoperation : McCabeThiele.java

```
package process.unitoperation;

import numerical.analysis.*;

public interface McCabeThiele {
        int calcNumberOfTrays(PolynomialSplineFunction function);
}
```

B.29 process.unitoperation : Tray.java

```
package process.unitoperation;

import excelextractor.*;
import validation.*;

public class Tray {
        //-------------------------------------------------
        //      INSTANCE VARIABLES
        //-------------------------------------------------
        private String material;
        private double diameter;

        //-------------------------------------------------
        //      CONSTRUCTOR(S)
        //-------------------------------------------------
        public Tray() {}// end of empty constructor

        public Tray(String material) {
                this.setMaterial(material);
        }// end of overloaded constructor

        public Tray(String material, double diameter) {
                this.material = material;
                this.diameter = diameter;
        }// end of second overloaded constructor

        public Tray(ExcelExtractor ee) {
                this.setMaterial(ee);
                this.setDiameter(ee);
        }// end of ExcelExtractor overloaded constructor

        public Tray(Tray tray) {
                this(tray.material, tray.diameter);
        }// end of copy constructor
```

```
        //-------------------------------------------------
        //      ACCESSORS AND MUTATORS
        //-------------------------------------------------
        // material accessor and mutator
        public String getMaterial()                         {return this.material;}
        public void setMaterial(String material)     {this.material = material;}
        public void setMaterial(ExcelExtractor ee)   {this.material = ee.getMocTrays();}

        // diameter accessor and mutator
        public double getDiameter()                         {return this.diameter;}
        public void setDiameter(double diameter) {
                if (Validate.isNonNegative(diameter))
                        this.diameter = diameter;
                else throw new InvalidArgumentException();
        }
        public void setDiameter(ExcelExtractor ee)   {this.diameter = ee.getDiameter();}

        //-------------------------------------------------
        //      CLONE METHOD
        //-------------------------------------------------
        //clone method
        public Tray clone() {
                return new Tray(this);
        }
}// end of Tray class
```

B.30 process.unitoperation : SieveTray.java

```
package process.unitoperation;

import excelextractor.*;

public class SieveTray extends Tray {

        //-------------------------------------------------
        //      CONSTRUCTOR(S)
        //-------------------------------------------------
        public SieveTray() { }// end of empty constructor

        public SieveTray(ExcelExtractor ee) {
                super(ee);
        }// end of overloaded constructor

}// end of SieveTray class
```

B.31 validation: Validate.java

```
package validation;

public class Validate {

        public static void checkNotNull(Object value) {
                if (!isNotNull(value))
                        throw new NullArgumentException();
        }

        public static void checkNotEmpty(double[] c) {
                if (c.length == 0)
                        throw new EmptyArrayArgumentException();
        }

        public static boolean isNotNull(Object value) {
                if (value != null)
                        return true;
                else return false;
        }
        public static boolean isType(String str, String type) {
                try {
                        if (type.equalsIgnoreCase("float"))
                                Float.parseFloat(str);
```

```
                        else if (type.equalsIgnoreCase("int"))
                                Integer.parseInt(str);
                        else if (type.equalsIgnoreCase("double"))
                                Double.parseDouble(str);
                return true;
                } catch(Exception e) {
                        return false;
                }
        }

        public static boolean isNonNegative(double value) {
                if (value < 0)
                        return false;
                else return true;
        }

        public static boolean isNonNegative(double[] value) {
                for (int i = 0; i < value.length; i++) {
                        if (!isNonNegative(value[i]))
                                return false;
                }
                return true;
        }

        public static boolean isLessThan1(double value) {
                if (value > 1)
                        return false;
                else return true;
        }

        public static boolean isLessThan1(double[] value) {
                for (int i = 0; i < value.length; i++) {
                        if (!isLessThan1(value[i]))
                                return false;
                }
                return true;
        }
}
```

B.32 validation: EmptyArrayArgumentException.java

```
package validation;

@SuppressWarnings("serial")
public class EmptyArrayArgumentException extends InvalidArgumentException {
        public EmptyArrayArgumentException() {
                super();
                this.printStackTrace();
        }
        public EmptyArrayArgumentException(String msg) {
                super(msg);
                this.printStackTrace();
        }
}
```

B.33 validation: InvalidArgumentException.java

```
package validation;

@SuppressWarnings("serial")
public class InvalidArgumentException extends IllegalArgumentException {
        public InvalidArgumentException() {
                super();
        }
        public InvalidArgumentException(String msg) {
                super(msg);
        }
        public InvalidArgumentException(String variable, double wrong, String correct) {
        super("Wrong value of " + variable + " entered!\n" + "Entered value: " +
                        wrong + ".\nPermitted: " + correct);
```

```
	}
	public InvalidArgumentException(String variable, String wrong, String correct) {
	super("Wrong value of " + variable + " entered!\n" + "Entered value: " + wrong +
		".\nPermitted: " + correct);
	}
}
```

B.34 validation: NullArgumentException.java

```
package validation;

@SuppressWarnings("serial")
public class NullArgumentException extends InvalidArgumentException {
	public NullArgumentException() {
		super();
		this.printStackTrace();
	}
	public NullArgumentException(String msg) {
		super(msg);
		this.printStackTrace();
	}
}
```

B.35 validation: OutOfRangeArgumentException.java

```
package validation;

@SuppressWarnings("serial")
public class OutOfRangeArgumentException extends InvalidArgumentException {
	public OutOfRangeArgumentException() {
		super();
		this.printStackTrace();
	}
	public OutOfRangeArgumentException(String msg) {
		super(msg);
		this.printStackTrace();
	}
	public OutOfRangeArgumentException(String variable, double wrong, double low, double
high) {
		super("Wrong value of " + variable + " entered!\n" + "Entered value: " + wrong +
".\nPermitted range: " + low + " - " + high + " ");
		this.printStackTrace();
	}
	public OutOfRangeArgumentException(String variable, String wrong, double low, double
high) {
		super("Wrong value of " + variable + " entered!\n" + "Entered value: " + wrong +
".\nPermitted range: " + low + " - " + high + " ");
		this.printStackTrace();
	}
}
```

C MATLAB Code

C.1 Function: `main`

```
function main
    j = java.text.NumberFormat.getCurrencyInstance();
    hAx = subplothandles(3,3,14);
    col1 = calcVentureResults(1);
    col2 = calcVentureResults(2);
    col3 = calcVentureResults(3);
    col1.PlotStages(hAx(1)); title(hAx(1),['Venture: 1/#Trays: ' ...
        num2str(col1.NumberOfTrays)  '/$PBP1y: ' ...
        char(j.format(col1.ProfitPBP1))]);
    col2.PlotStages(hAx(2)); title(hAx(2),['Venture: 2/#Trays: ' ...
        num2str(col2.NumberOfTrays)  '/$PBP1y: ' ...
        char(j.format(col2.ProfitPBP1))]);
    col3.PlotStages(hAx(3)); title(hAx(3),['Venture: 3/#Trays: ' ...
        num2str(col3.NumberOfTrays)  '/$PBP1y: ' ...
        char(j.format(col3.ProfitPBP1))]);
end
```

C.2 Function: `calc_venture_results`

```
function column = calc_venture_esults(vnum)
    column = Column; % Instantiate Column object
    % Venture 1
    if vnum == 1
        Components = {'Fancy Scarves','Fancy Ties'};
        column.Diameter         = 2;
        column.Length           = 10;
        column.PressureGauge    = 16.4;
        column.RefluxRatio      = 3.5;
        column.Material         = 'Titanium';
        column.Tray.Material    = 'Nickel alloy';
        column.Tray.Diameter    = column.Diameter;
        column.Feed.Temperature             = 421.8;            % K
        column.Feed.Flow                    = 100;              % kmol/h
        column.Feed.ComponentName           = Components;
        column.Feed.ComponentFrac           = [0.5 0.5];        % mol/mol
        column.Feed.Cost                    = 25;               % CDN$/kmol

        column.Distillate.ComponentName     = Components;
        column.Distillate.ComponentFrac     = [0.95 0.05];      % mol/mol
        column.Distillate.Cost              = 100;              % CDN$/kmol

        column.Bottoms.ComponentName        = Components;
        column.Bottoms.ComponentFrac        = [0.1 0.9];        % mol/mol
        column.Bottoms.Cost                 = 30;               % CDN$/kmol
    % Venture 2
    elseif vnum == 2
        Components = {'Cocoa','Sugar'};
        column.Diameter         = 2.4;
        column.Length           = 22;
        column.PressureGauge    = 6.5;
        column.RefluxRatio      = 10.6;
        column.Material         = 'Nickel alloy clad';
```

```
    column.Tray.Material    = 'Nickel alloy';
    column.Tray.Diameter    = column.Diameter;

    column.Feed.Temperature             = 399;              % K
    column.Feed.Flow                    = 100;              % kmol/h
    column.Feed.ComponentName           = Components;
    column.Feed.ComponentFrac           = [0.25 0.75];      % mol/mol
    column.Feed.Cost                    = 42;               % CDN$/kmol

    column.Distillate.ComponentName     = Components;
    column.Distillate.ComponentFrac     = [0.72 0.28];      % mol/mol
    column.Distillate.Cost              = 200;              % CDN$/kmol

    column.Bottoms.ComponentName        = Components;
    column.Bottoms.ComponentFrac        = [0.15 0.85];      % mol/mol
    column.Bottoms.Cost                 = 47;               % CDN$/kmol
% Venture 3
elseif vnum == 3
    Components = {'Watches','iPads'};
    column.Diameter         = 1.5;
    column.Length           = 32;
    column.PressureGauge    = 2.1;
    column.RefluxRatio      = 3.8;
    column.Material         = 'Stainless steel clad';
    column.Tray.Material    = 'Stainless steel';
    column.Tray.Diameter    = column.Diameter;

    column.Feed.Temperature             = 509;              % K
    column.Feed.Flow                    = 100;              % kmol/h
    column.Feed.ComponentName           = Components;
    column.Feed.ComponentFrac           = [0.45 0.55];      % mol/mol
    column.Feed.Cost                    = 206;              % CDN$/kmol

    column.Distillate.ComponentName     = Components;
    column.Distillate.ComponentFrac     = [0.85 0.15];      % mol/mol
    column.Distillate.Cost              = 375;              % CDN$/kmol

    column.Bottoms.ComponentName        = Components;
    column.Bottoms.ComponentFrac        = [0.11 0.89];      % mol/mol
    column.Bottoms.Cost                 = 50;               % CDN$/kmol
end

column.Bottoms.Flow = column.Feed.Flow/(1 ...
    + (column.Bottoms.ComponentFrac(1) ...
    - column.Feed.ComponentFrac(1)) ...
    /(column.Feed.ComponentFrac(1) ...
    - column.Distillate.ComponentFrac(1))); % kmol/h
column.Distillate.Flow = column.Feed.Flow ...
    - column.Feed.Flow/(1 + (column.Bottoms.ComponentFrac(1) ...
    - column.Feed.ComponentFrac(1))...
    /(column.Feed.ComponentFrac(1) ...
    - column.Distillate.ComponentFrac(1))); % kmol/h;

numberoftrays = column.NumberOfTrays;
column.Tray.NumberOfTrays = numberoftrays;

disp(' ');
```

```
    disp('----------------------------------------');
    disp('                RESULTS');
    disp('----------------------------------------');
    disp(['           Venture: ' num2str(vnum)]);
    disp(['   Number of Trays: ' num2str(column.NumberOfTrays)]);

    disp(' ');
    disp('********** Column Flow Rates ***********');
    disp(['         Feed Flow: ' num2str(column.Feed.Flow)]);
    disp(['   Distillate Flow: ' num2str(column.Distillate.Flow)]);
    disp(['      Bottoms Flow: ' num2str(column.Bottoms.Flow)]);

    disp(' ');
    disp('*********** Column Costing *************');
    disp(['      Capital Cost: ' num2str(column.Cp)]);
    disp(['  Bare Module Cost: ' num2str(column.Cbm)]);
    disp([' Total Module Cost: ' num2str(column.Ctm)]);
    disp(['  Grass Roots Cost: ' num2str(column.Cgr)]);

    disp(' ');
    disp('************* Tray Costing ************');
    disp(['      Capital Cost: ' num2str(column.Tray.Cp)]);
    disp(['  Bare Module Cost: ' num2str(column.Tray.Cbm)]);
    disp([' Total Module Cost: ' num2str(column.Tray.Ctm)]);
    disp(['  Grass Roots Cost: ' num2str(column.Tray.Cgr)]);

    disp(' ');
    disp('**** Both Column and Tray Costing ****');
    disp(['      Capital Cost: ' num2str(column.Cp +column.Tray.Cp)]);
    disp(['  Bare Module Cost: ' num2str(column.Cbm+column.Tray.Cbm)]);
    disp([' Total Module Cost: ' num2str(column.Ctm+column.Tray.Ctm)]);
    disp(['  Grass Roots Cost: ' num2str(column.Cgr+column.Tray.Cgr)]);

    disp(' ');
    disp('****** Overall Revenues/Profit *******');
    disp(['Production Revenue: ' num2str(column.StreamProfit)]);
    disp([' 1 Year PBP Profit: ' num2str(column.ProfitPBP1)]);
end
```

C.3 Class: Column

```
classdef Column
    %---------------------------------------
    % PROPERTIES
    %---------------------------------------
    properties (Constant)
        CEPCI_2014 = 579.7;
        CEPCI_1996 = 382;
    end
    properties
        Diameter            % m
        Length              % m
        PressureGauge       % bar gauge
        RefluxRatio
        Material
        Tray
```

```
    Feed
    Distillate
    Bottoms
end
properties (Dependent)
    FeedQuality
    QLine
    EnrichLine
    StripLine
    XIntersection
    YIntersection
    NumberOfTrays
    Cp
    Cbm0
    Cbm
    Ctm
    Cgr
    TotalCost
    StreamProfit
    ProfitPBP1
end
%---------------------------------------
% METHODS
%---------------------------------------
methods
    function obj = Column
        obj.Feed          = Stream;
        obj.Distillate    = Stream;
        obj.Bottoms       = Stream;
        obj.Tray          = SieveTray;
    end
    %---------------------------------------
    function value = get.FeedQuality(obj)
        value = (obj.Feed.HeatCapacity*(obj.Feed.NormalBP ...
            - obj.Feed.Temperature) + obj.Feed.LatentHeat)/ ...
            obj.Feed.LatentHeat;
    end
    %---------------------------------------
    function value = get.QLine(obj)
        value = Line;
        value.Slope = obj.FeedQuality/(obj.FeedQuality - 1);
        value.Intercept = -obj.Feed.ComponentFrac(1)/ ...
            (obj.FeedQuality - 1);
    end
    %---------------------------------------
    function value = get.EnrichLine(obj)
        value = Line;
        value.Slope = obj.RefluxRatio/(obj.RefluxRatio + 1);
        value.Intercept = obj.Distillate.ComponentFrac(1)/ ...
            (obj.RefluxRatio + 1);
    end
    %---------------------------------------
    function value = get.StripLine(obj)
        value = Line;
        value.Slope = (obj.YIntersection - ...
            obj.Bottoms.ComponentFrac(1))/(obj.XIntersection ...
            - obj.Bottoms.ComponentFrac(1));
```

```
    value.Intercept = obj.YIntersection ...
        - value.Slope*obj.XIntersection;
end
%---------------------------------------
function value = get.XIntersection(obj)
    value = (obj.EnrichLine.Intercept - obj.QLine.Intercept) ...
        /(obj.QLine.Slope - obj.EnrichLine.Slope);
end
%---------------------------------------
function value = get.YIntersection(obj)
    value = obj.QLine.Slope*obj.XIntersection ...
        + obj.QLine.Intercept;
end
%---------------------------------------
function value = get.NumberOfTrays(obj)
    fXStrip = @(Y)(Y - obj.StripLine.Intercept) ...
        /obj.StripLine.Slope;
    fXEnrich = @(Y)(Y - obj.EnrichLine.Intercept)/ ...
        obj.EnrichLine.Slope;

    BEDobj     = BED(obj.Feed.ComponentName{1});
    fitPoly    = fit(BEDobj.X',BEDobj.Y','pchipinterp');
    % equlibrium stages
    lastX = obj.Bottoms.ComponentFrac(1);
    lastY = lastX;
    i = 1;
    while(lastX < obj.Distillate.ComponentFrac(1))
        ystage(i) = fitPoly(lastX);
        if ystage(i) >= obj.Distillate.ComponentFrac(1)
            xstage(i) = ystage(i);
        elseif ystage(i) >= obj.YIntersection
            xstage(i) = fXEnrich(ystage(i));
        else
            xstage(i) = fXStrip(ystage(i));
        end
        lastX = xstage(i);
        lastY = ystage(i);
        i = i + 1;
    end
    value = i - 1;
end
%---------------------------------------
function PlotStages(obj,hAx)
    fXStrip = @(Y)(Y - obj.StripLine.Intercept)/ ...
        obj.StripLine.Slope;
    fXEnrich = @(Y)(Y - obj.EnrichLine.Intercept)/ ...
        obj.EnrichLine.Slope;

    BEDobj     = BED(obj.Feed.ComponentName{1});
    fitPoly    = fit(BEDobj.X',BEDobj.Y','pchipinterp');

    % fitted polynomial and equilibrium data
    plot(hAx,BEDobj.X',BEDobj.Y','k*');
    hold on;
    plot(hAx,linspace(0,1,50),fitPoly(linspace(0,1,50)),'k');

    % 45 degree line
```

```
    plot(hAx,[0 1],[0 1],'k');

    % StripLine
    plot(hAx,[obj.Bottoms.ComponentFrac(1),obj.XIntersection], ...
        [obj.Bottoms.ComponentFrac(1),obj.YIntersection],'k');

    % EnrichLine
    plot(hAx,[obj.Distillate.ComponentFrac(1), ...
        obj.XIntersection],[obj.Distillate.ComponentFrac(1), ...
        obj.YIntersection],'k');

    % equlibrium stages
    lastX = obj.Bottoms.ComponentFrac(1);
    lastY = lastX;
    i = 1;
    while(lastX < obj.Distillate.ComponentFrac(1))
        ystage(i) = fitPoly(lastX);
        if ystage(i) >= obj.Distillate.ComponentFrac(1)
            xstage(i) = ystage(i);
        elseif ystage(i) >= obj.YIntersection
            xstage(i) = fXEnrich(ystage(i));
        else
            xstage(i) = fXStrip(ystage(i));
        end
        plot(hAx,[lastX,lastX],[lastY,ystage(i)],'k');
        plot(hAx,[lastX,xstage(i)],[ystage(i),ystage(i)],'k');
        lastX = xstage(i);
        lastY = ystage(i);
        i = i + 1;
    end
    %numberoftrays = i - 1;
    xlabel(hAx,'x (mole frac.)');
    ylabel(hAx,'y (mole frac.)');
    legend(hAx,{'pchip fit','data'},'location','southeast');
end
%---------------------------------------
function value = get.Cp(obj)
    % Cp (Capital Cost)
    if      (obj.Diameter() <= 0.3)
        k1 = 3.3392; k2 =-0.5538; k3 = 0.2851;
    elseif (obj.Diameter() <= 0.5 && obj.Diameter() >  0.3)
        k1 = 3.4746; k2 = 0.5893; k3 = 0.2053;
    elseif (obj.Diameter() <= 1.0 && obj.Diameter() >  0.5)
        k1 = 3.6237; k2 = 0.5262; k3 = 0.2146;
    elseif (obj.Diameter() <= 1.5 && obj.Diameter() >  1.0)
        k1 = 3.7559; k2 = 0.6361; k3 = 0.1069;
    elseif (obj.Diameter() <= 2.0 && obj.Diameter() >  1.5)
        k1 = 3.9484; k2 = 0.4623; k3 = 0.1717;
    elseif (obj.Diameter() <= 2.5 && obj.Diameter() >  2.0)
        k1 = 4.0547; k2 = 0.4620; k3 = 0.1558;
    elseif (obj.Diameter() <= 3.0 && obj.Diameter() >  2.5)
        k1 = 4.1110; k2 = 0.6094; k3 = 0.0490;
    elseif (obj.Diameter() <= 4.0 && obj.Diameter() >  3.0)
        k1 = 4.3919; k2 = 0.2859; k3 = 0.1842;
    end
    value = 10^(k1 + k2*log10(obj.Length) ...
        + k3*(log10(obj.Length))^2);
```

```
end
%---------------------------------------
function [Fm,Fp] = calcFactors(obj)
    % Fp (Pressure Factor)
    if(obj.PressureGauge >= 3.7 && obj.PressureGauge < 400)
        Fp = 0.5146 + 0.6838*log10(obj.PressureGauge) ...
            + 0.2970*(log10(obj.PressureGauge))^2 ...
            + 0.0235*(log10(obj.PressureGauge))^6 ...
            + 0.0020*(log10(obj.PressureGauge))^8;
    elseif(obj.PressureGauge >= -0.5 && obj.PressureGauge < 3.7)
        Fp = 1;
    elseif(obj.PressureGauge < -0.5)
        Fp = 1.25;
    end
    % Fm (Material Factor)
    switch(obj.Material)
        case 'Carbon Steel'
            Fm = 1.0;
        case 'Stainless steel clad'
            Fm = 2.5;
        case 'Stainless steel'
            Fm = 4.0;
        case 'Nickel alloy clad'
            Fm = 4.5;
        case 'Nickel alloy'
            Fm = 9.8;
        case 'Titanium clad'
            Fm = 4.9;
        case 'Titanium'
            Fm = 10.6;
    end
end
%---------------------------------------
function value = get.Cbm(obj)
    % Cbm (Bare Module Cost)
    B1 = 2.50;
    B2 = 1.72;
    [Fm,Fp] = obj.calcFactors;
    value = obj.Cp*(B1 + B2*Fm*Fp)*(obj.CEPCI_2014/obj.CEPCI_1996);
end
%---------------------------------------
function value = get.Cbm0(obj)
    B1 = 2.50;
    B2 = 1.72;
    value= obj.Cp*(B1 +B2*1.0*1.0)*(obj.CEPCI_2014/obj.CEPCI_1996);
end
%---------------------------------------
function value = get.Ctm(obj)
    % Ctm (Total Module Cost)
    value = 1.18*obj.Cbm;
end
%---------------------------------------
function value = get.Cgr(obj)
    % Cgr (Grass root Cost)
    value = obj.Ctm + 0.5*obj.Cbm0;
end
%---------------------------------------
```

```
        function value = get.TotalCost(obj)
            value = obj.Cgr + obj.Tray.Cgr;
        end
        %---------------------------------------
        function value = get.StreamProfit(obj)
            value = (obj.Distillate.Flow*obj.Distillate.Cost ...
                + obj.Bottoms.Flow*obj.Bottoms.Cost ...
                - obj.Feed.Flow*obj.Feed.Cost)*24*365;
        end
        %---------------------------------------
        function value = get.ProfitPBP1(obj)
            value = obj.StreamProfit ...
                - obj.TotalCost;
        end
    end
end
```

C.4 Class: `Stream`

```
classdef Stream
    %---------------------------------------
    % PROPERTIES
    %---------------------------------------
    properties
        ComponentName
        ComponentFrac       % mol/mol
        Temperature
        Flow                % kmol/h
        Cost                % CDN$/kmol
    end
    properties (Dependent)
        ComponentHeatCapacity
        HeatCapacity
        ComponentNormalBP
        NormalBP
        ComponentLatentHeat
        LatentHeat
    end
    %---------------------------------------
    % METHODS
    %---------------------------------------
    methods
        %---------------------------------------
        function obj = set.Flow(obj,val)
            obj.Flow = val;
        end
        function obj = set.ComponentName(obj,val)
            for i = 1:length(val)
                obj.ComponentName{i} = val{i};
            end
        end
        %---------------------------------------
        function value = get.ComponentHeatCapacity(obj)
            value = zeros(1,length(obj.ComponentName));
            for i = 1:length(obj.ComponentName)
                species = Species(obj.ComponentName{i});
```

```
            value(i) = species.HeatCapacity;
        end
    end
    %---------------------------------------
    function value = get.ComponentNormalBP(obj)
        value = zeros(1,length(obj.ComponentName));
        for i = 1:length(obj.ComponentName)
            species = Species(obj.ComponentName{i});
            value(i) = species.NormalBP;
        end
    end
    %---------------------------------------
    function value = get.ComponentLatentHeat(obj)
        value = zeros(1,length(obj.ComponentName));
        for i = 1:length(obj.ComponentName)
            species = Species(obj.ComponentName{i});
            value(i) = species.LatentHeat;
        end
    end
    %---------------------------------------
    function value = get.HeatCapacity(obj)
        value = sum(obj.ComponentFrac.*obj.ComponentHeatCapacity);
    end
    %---------------------------------------
    function value = get.NormalBP(obj)
        value = sum(obj.ComponentFrac.*obj.ComponentNormalBP);
    end
    %---------------------------------------
    function value = get.LatentHeat(obj)
        value = sum(obj.ComponentFrac.*obj.ComponentLatentHeat);
    end
  end
end
```

C.5 Class: `Line`

```
classdef Line
    properties
        Slope
        Intercept
    end
    %% METHODS
    methods
        function obj = Line(varargin)
            if isempty(varargin)
                obj.Slope = 0;
                obj.Intercept = 0;
            else
                obj.Slope        = varargin{1};
                obj.Intercept    = varargin{2};
            end
        end
        function value = calcX(obj, y)
            value = (y - obj.Intercept)./obj.Slope;
        end
        function value = calcY(obj, x)
            value = obj.Slope.*x + obj.Intercept;
```

```
        end
    end


    methods (Static)
        function pt = calcIntersection(lf1,lf2)
            x = (lf1.Intercept - lf2.Intercept)/(lf2.Slope - lf1.Slope);
            y = lf2.Slope*x + lf2.Intercept;
            pt.x = x;
            pt.y = y;
        end
    end
end
```

C.6 Class: `SieveTray`

```
classdef SieveTray
    %---------------------------------------
    % PROPERTIES
    %---------------------------------------
    properties (Constant)
        CEPCI_2014 = 579.7;
        CEPCI_1996 = 382;
    end
    properties
        Material
        Diameter
        NumberOfTrays
    end
    properties (Dependent)
        Fq
        Fbm
        Cbm
        Cbm0
        Cp
        Ctm
        Cgr
    end
    %---------------------------------------
    % METHODS
    %---------------------------------------
    methods
        function value = get.Cp(obj)
            % Cp (Capital Cost)
            value = 235 + 19.80*obj.Diameter + 75.07*obj.Diameter^2;
        end
        %---------------------------------------
        function value = get.Fq(obj)
            n = obj.NumberOfTrays;
            if      (n >= 1           && n < (4  + 1 )/2), value = 3.0;
            elseif  (n >= (4  + 1)/2  && n < (7  + 4 )/2), value = 2.5;
            elseif  (n >= (7  + 4)/2  && n < (10 + 7 )/2), value = 2.0;
            elseif  (n >= (10 + 7)/2  && n < 20         ), value = 1.5;
            elseif  (n >= 20), value = 1;
            end
        end
```

```
        %---------------------------------------
        function value = get.Fbm(obj)
            % Fbm (Bare Module Factor)
            switch(obj.Material)
                case 'Carbon steel'
                    value = 1.2;
                case 'Stainless steel'
                    value = 2.0;
                case 'Nickel alloy'
                    value = 5.0;
            end
        end
        %---------------------------------------
        function value = get.Cbm(obj)
            % Cbm (Bare Module Cost)
            value = obj.Cp*obj.NumberOfTrays*obj.Fbm*obj.Fq
                                            *(obj.CEPCI_2014/obj.CEPCI_1996);
        end
        %---------------------------------------
        function value = get.Cbm0(obj)
            % Cbm (Bare Module Cost - Base Conditions)
            Fbm0 = 1.2;
            value = obj.Cp*obj.NumberOfTrays*Fbm0*obj.Fq
                                            *(obj.CEPCI_2014/obj.CEPCI_1996);
        end
        %---------------------------------------
        function value = get.Ctm(obj)
            value = 1.18*obj.Cbm;
        end
        %---------------------------------------
        function value = get.Cgr(obj)
            value = obj.Ctm + 0.5*obj.Cbm0;
        end
    end
end
```

D Unit Testing (Validations)

D.1 `numerical.analysis` Package

Table D-1. Validation of the `calcX`, `calcY`, and `calcIntersection` methods of the `LinearFunction` class.

Class	Method	Input Parameter(s)	Java Output	MATLAB Solution	Comment
LinearFunction	calcX (double[]) : double[]	({1.00, 2.00, 3.00})	[-1.65, -1.35, -1.06]	[-1.65 -1.35 -1.06]	
		({-1.00, -2000.00, 4000000.00})	[-2.24, -590.18, 1176468.65]	[-2.24 -590.18 1176468.65]	
		null	validation.NullArgumentException at validation.Validate.checkNotNull(Validate.java:7)	*N/A*	NullArgumentException thrown when method's logic finds null input.
	calcY (double[]) : double[]	({1.00, 2.00, 3.00})	[45.40, 90.40, 135.40]	[45.4 90.4 135.4]	
		({-1.00, -2000.00, 4000000.00})	[-44.60, -89999.60, 180000000.40]	[-44.6 -89999.6 180000000.4]	
		null	validation.NullArgumentException at validation.Validate.checkNotNull(Validate.java:7)	*N/A*	NullArgumentException thrown when method's logic finds null input.
	calcIntersection (LinearFunction, LinearFunction) : double[]	(LinearFunction(3.4,6.6), LinearFunction(45.,0.4))	[0.15, 7.11]	[0.14904 7.1067]	
		(LinearFunction(3.4,6.6), LinearFunction(3.4,6.6))	validation.InvalidArgumentException: Division by zero error! Cannot have denominator value of zero when calculating intersection point!. at numerical.analysis.LinearFunction.calcIntersection(LinearFunction.java:102)	[NaN NaN]	InvalidArgumentException thrown when method's logic finds potential division by zero and outputs custom message. MATLAB allows arithmetics with non-numeric values such as NaN (not a number), Inf (infinity), etc.
		(null, null)	validation.NullArgumentException at validation.Validate.checkNotNull(Validate.java:7) at numerical.analysis.LinearFunction.calcIntersection(LinearFunction.java:97)	*N/A*	NullArgumentException thrown when method's logic finds null input.

Table D-2. Validation of the `diff` method of the `Maths` class.

Class	Method	Input Parameter(s)	Java Output	MATLAB Solution	Comment
Maths	diff (double[]) : double[]	({1.00, 2.00, 3.00})	[1.00, 1.00, 0.00]	[1 1]	The MATLAB code, from which this Java implementation relied on, resizes its array easily and dynamically (staple of MATLAB), but for the intended purposes here, this was not required since the dropped dimension (last element) was never used subsequently (i.e. this was a design intent).
		({-3234232.34, 3, 325355.33333})	[3234235.34, 325352.33, 0.00]	[3234235.34 325352.33333]	Same as above.
		null	validation.NullArgumentException at validation.Validate.checkNotNull(Validate.java:7) at numerical.analysis.Maths.diff(Maths.java:44) at numerical.Main.main(Main.java:47)	*N/A*	NullArgumentException thrown when method's logic finds null input.
	diff (double[][]) : double[][]	({{1.0, 2.0, 3.0}, {31.0, 43.0, 33.0}, {6.0, 1.0, 32.0}})	{[30.0, 41.0, 30.0], [-25.0, -42.0, -1.0]}	[30 41 30;-25 -42 -1]	
		({{-23241, 24343, 33}, {0.000031, 2.43, 5.3}, {6, 0.00001, 3223424}})	{[23241.000031, -24340.57, -27.7], [5.999969, -2.42999, 3223418.7]}	[23241.000031 -24340.57 -27.7; 5.999969 -2.42999 3223418.7]	
		null	validation.NullArgumentException at validation.Validate.checkNotNull(Validate.java:7) at numerical.analysis.Maths.diff(Maths.java:58)	*N/A*	NullArgumentException thrown when method's logic finds null input.

Table D-3. Validation of the `hypot` and prod methods of the `Maths` class.

Class	Method	Input Parameter(s)	Java Output	MATLAB Solution	Comment
Maths	hypot(double, double) : double	(2.30, 3.40)	4.1	4.1049	
		(-1.00, 2.00)	2.24	2.24	
		(-100000, 450.343243432)	100001.01	100001.014	
	prod (double[]) : double	(new double[] {1.00, 2.00, 3.00})	6	6	
		(new double[] {-3234232.34, 3, 325355.33333})	-3.15682E+12	-3.15682E+12	
		null	validation.NullArgumentException at validation.Validate.checkNotNull(Validate.java:7) at numerical.analysis.Maths.prod(Maths.java:75)	*N/A*	NullArgumentException thrown when method's logic finds null input.
	prod (double[][]) : double[]	(new double[][] {{1.0, 2.0, 3.0}, {1.0, 2.0, 3.0}, {1.0, 2.0, 3.0}})	[1.00, 8.00, 27.00]	[1 8 27]	
		(new double[][] {{-23241, 24343, 33}, {0.000031, 2.43, 5.3}, {6, 0.00001, 3223424}})	[-4.32, 0.59, 563776857.60]	[-4.322826 0.5915349 563776857.6]	
		null	validation.NullArgumentException at validation.Validate.checkNotNull(Validate.java:7) at numerical.analysis.Maths.prod(Maths.java:88)	*N/A*	NullArgumentException thrown when method's logic finds null input.

Table D-4. Validation of the `calcY` method of the `PolynomialFunction` class.

Class	Method	Input Parameter(s)	Java Output	MATLAB Solution	Comment
PolynomialFunction	calcY (double[]) : double[]	({1.00, 2.00, 3.00, 4.00, 5.00})	[9.00, 24.00, 47.00, 78.00, 117.00]	[9 24 47 78 117]	
		({20.00, 2.00, -3.00, 4.00, 5000.00})	[1662.00, 24.00, 29.00, 78.00, 100015002.00]	[1662 24 29 78 100015002]	
		null	validation.NullArgumentException at validation.Validate.checkNotNull(Validate.java:7) at numerical.analysis.PolynomialFunction.calcY(PolynomialFunction.java:60)	*N/A*	NullArgumentException thrown when method's logic finds null input.

Table D-5. Validation of the `calcY` and `isValidPoint` methods of the `PolynomialSplineFunction` class.

Class	Method	Input Parameter(s)	Java Output	MATLAB Solution	Comment
PolynomialSplineFunction	calcY (double[]) : double[]	({0.11, 0.22, 0.33, 0.44, 0.55, 0.66, 0.77, 0.88, 0.99})	[0.11, 0.22, 0.33, 0.44, 0.55, 0.66, 0.77, 0.88, 0.99]	[0.38 0.56 0.66 0.74 0.79 0.84 0.89 0.94 0.99]	
		({0.11, -200.00, 0.33, 4.00, 0.55, 0.66, 23.00, 0.88, 0.99})	validation.OutOfRangeArgumentException: Wrong value of calcY argument entered! Entered value: -200.0. Permitted range: 0.0 - 1.0 at numerical.analysis.PolynomialSplineFunction.calcY(PolynomialSplineFunction.java:111)	[0.38 93568121.57 0.66 -2183.13 0.79 0.84 -860338.91 0.94 0.99]	MATLAB permits the user to enter values outside range of knots of splines and uses the coefficients at whichever end is closest to do calculate value. The design intent here was to forbid such extrapolations outside the range of supplied interpolation knots. The OutOfRangeArgumentException class apprehends this and displays useful details about validity range of calculable interpolated value.
		null	validation.NullArgumentException at validation.Validate.checkNotNull(Validate.java:7) at numerical.analysis.PolynomialSplineFunction.calcY(PolynomialSplineFunction.java:128)	*N/A*	NullArgumentException thrown when method's logic finds null input.
	isValidPoint (double) : boolean	(0)	true	true	
		(-200)	false	false	
		(-1.001)	false	false	

D.2 `numerical.interpolation` Package

Table D-6. Validation of the `interpolate` method of the `PchipInterpolator` class.

Class	Method	Input Parameter(s)	Java Output	MATLAB Solution	Comment
PchipInterpolator	interpolate (double[], double[]) : PolynomialSplineFunction	({0, 0.013, 0.025, 0.052, 0.09, 0.16, 0.25, 0.37, 0.52, 0.7, 0.85, 0.96, 1}, {0, 0.07, 0.13, 0.22, 0.33, 0.47, 0.59, 0.69, 0.78, 0.86, 0.93, 0.97, 1})	[0.00, 5.58, -15.23, -11.77] [0.07, 5.18, 44.12, -4945.22] [0.13, 4.11, -48.94, 753.84] [0.22, 3.11, 1.40, -186.59] [0.33, 2.41, -6.17, 4.60] [0.47, 1.61, -2.93, -2.00] [0.59, 1.04, -2.30, 5.02] [0.69, 0.70, -0.78, 0.66] [0.78, 0.51, -0.82, 2.46] [0.86, 0.46, 0.55, -3.18] [0.93, 0.41, -2.17, 16.25] [0.97, 0.52, 9.04, -80.77]	[0.00 5.58 -15.23 -11.77] [0.07 5.18 44.12 -4945.22] [0.13 4.11 -48.94 753.84] [0.22 3.11 1.40 -186.59] [0.33 2.41 -6.17 4.60] [0.47 1.61 -2.93 -2.00] [0.59 1.04 -2.30 5.02] [0.69 0.70 -0.78 0.66] [0.78 0.51 -0.82 2.46] [0.86 0.46 0.55 -3.18] [0.93 0.41 -2.17 16.25] [0.97 0.52 9.04 -80.77]	
		({-30.00, -10.00, 0.00, 10.00, 20.00, 30.00, 40.00, 60.00, 200.00}, {40.00, 0.07, 40.00, 0.22, 50.00, 0.47, 200.00, 0.69, 30.00})	[40.00, -5.99, -0.30, 0.02] [0.07, 11.98, -1.20, 0.04] [40.00, 0.00, -1.19, 0.08] [0.22, 0.00, 1.49, -0.10] [50.00, 0.00, -1.49, 0.10] [0.47, 0.00, 5.99, -0.40] [200.00, 0.00, -1.49, 0.05] [0.69, 0.00, -0.00, 0.00]	[40.00 -5.99 0.30 -0.00] [0.07 0.00 1.20 -0.08] [40.00 0.00 -1.19 0.08] [0.22 0.00 1.49 -0.10] [50.00 0.00 -1.49 0.10] [0.47 0.00 5.99 -0.40] [200.00 0.00 -1.49 0.05] [0.69 0.00 -0.00 0.00]	
		({1.2, 2, 4}, {4.2, 2})	Exception in thread "main" validation.InvalidArgumentException: Dimension mismatch. Both data arrays x and y must be the same length! at numerical.interpolation.PchipInterpolator.interpolate(PchipInterpolator.java:41)	Error using fit>iFit (line 135) X and Y must have the same number of rows.	InvalidArgumentException thrown indicating Dimension mismatch.
		({1.2, 2}, {4.2, 2})	Exception in thread "main" validation.InvalidArgumentException: Minimum number of data points: 3. You entered: 2 at numerical.interpolation.PchipInterpolator.interpolate(PchipInterpolator.java:45)	Error using fit>iFit (line 477) Insufficient data. You need at least 3 data points to fit this model.	InvalidArgumentException thrown indicating Minimum data points not met.
		(null, null)	validation.NullArgumentException at validation.Validate.checkNotNull(Validate.java:7) at numerical.interpolation. PchipInterpolator.interpolate(PchipInterpolator.java:35) at numerical.Main.main(Main.java:121)	*N/A*	NullArgumentException thrown when method's logic finds null input.

D.3 `process.unitoperation` Package

Table D-7. Validation of `calcFlowRates` method in `Column` class.

Package	Class	Method Name	Input Parameter(s)	Java Output	Hand Solution	Comment
process .unitoperation	Column	calcFlowRates	Feed CompFraction = 0.5 Feed FlowRate = 100 Distillate CompFraction = 0.95 Bottoms CompFraction = 0.1	[100.00, 47.06, 52.94]	[100, 47.06, 52.94]	Java is specified to 2 decimal places
		calcFlowRates	Feed CompFraction = 0.5 Feed FlowRate = 100000 Distillate CompFraction = 0.95 Bottoms CompFraction = 0.1	[100000.00, 47058.82, 52941.18]	[100000, 47058.82, 52941.18]	Java is specified to 2 decimal places
		calcFlowRates	Feed CompFraction = 0.5 Feed FlowRate = -100 Distillate CompFraction = 0.95 Bottoms CompFraction = 0.1	"Wrong value of feed flow rate entered! Entered value: -100 Permitted: non negative"	[-100, -47.06, -52.94]	Exception catches unrealistic value in Java
		calcFlowRates	Feed CompFraction = 0.5 Feed FlowRate = f Distillate CompFraction = 0.95 Bottoms CompFraction = 0.1	"Invalid data type for feed flow rate Required: numeric"	[f, N/A, N/A]	Exception catches unrealistic value in Java
		calcFlowRates	Feed CompFraction = 1.5 Feed FlowRate = 100 Distillate CompFraction = 0.95 Bottoms CompFraction = 0.1	"Wrong value of feed fraction entered! Entered value: [1.5, 0.5] Permitted range: 0.0 - 1.0"	[100, 164.71, -64.71]	Exception catches unrealistic value in Java
		calcFlowRates	Feed CompFraction = -0.5 Feed FlowRate = 100 Distillate CompFraction = 0.95 Bottoms CompFraction = 0.1	"Wrong value of feed fraction entered! Entered value: [-0.5, 0.5] Permitted range: 0.0 - 1.0"	[100, -70.59, 170.59]	Exception catches unrealistic value in Java
		calcFlowRates	Feed CompFraction = f Feed FlowRate = 100 Distillate CompFraction = 0.95 Bottoms CompFraction = 0.1	"Invalid data type for component feed fraction Required: numeric"	[100, N/A, N/A]	Exception catches unrealistic value in Java
		calcFlowRates	Feed CompFraction = 0.5 Feed FlowRate = 100 Distillate CompFraction = 0.1 Bottoms CompFraction = 0.95	[100.00, 52.94, 47.06]	[100, 52.94, 47.06]	Java is specified to 2 decimal places

Table D-8. Validation of `calcProfit` method in `Column` class.

Class	Method Name	Input Parameter(s)	Java Output	Hand Solution	Comment
Column	calcProfit	Feed FlowRate = 100 Distillate FlowRate = 47.06 Bottoms FlowRate = 52.94	33236470.59	33236470.59	Same
	calcProfit	Feed FlowRate = 100000 Distillate FlowRate = 47058.82 Bottoms FlowRate = 52941.18	33236470588.24	33236470588.24	Same
	calcProfit	Feed FlowRate = -100 Distillate FlowRate = -47.06 Bottoms FlowRate = -52.94	"Wrong value of feed flow rate entered! Entered value: -100 Permitted: non negative"	-33236470.59	Exception catches unrealistic value in Java
	calcProfit	Feed FlowRate = f Distillate FlowRate = 47.06 Bottoms FlowRate = 52.94	"Invalid data type for feed flow rate Required: numeric"	N/A	Exception catches unrealistic value in Java

Table D-9. Validation of `calcNumberOfTrays` method in `DistillationColumn` class – part 1.

Package	Class	Method Name	Input Parameter(s)		Java Output	Hand Solution	Comment
process .unitoperation	DistillationColumn	calcNumberOfTrays	Equilibrium Data		6	6	Same
			liquid mol frac.	vapour mol frac.			
			0	0			
			0.013	0.07			
			0.025	0.13			
			0.052	0.22			
			0.09	0.33			
			0.16	0.47			
			0.25	0.59			
			0.37	0.69			
			0.52	0.78			
			0.7	0.86			
			0.85	0.93			
			0.96	0.97			
			1	1			
			Bottoms CompFraction = 0.1 Distillate CompFraction = 0.95				
		calcNumberOfTrays	Equilibrium Data		Wrong value of Liquid equilibrium fraction entered! Entered value: [0.0,0.013,0.025,0.052, 0.09,5.0,0.25,0.37, 0.52, 0.7,0.85,0.96,1.0] Permitted range: 0.0 - 1.0	6	Exception catchs out of range value in Java
			liquid mol frac.	vapour mol frac.			
			0	0			
			0.013	0.07			
			0.025	0.13			
			0.052	0.22			
			0.09	0.33			
			5	0.47			
			0.25	0.59			
			0.37	0.69			
			0.52	0.78			
			0.7	0.86			
			0.85	0.93			
			0.96	0.97			
			1	1			
			Bottoms CompFraction = 0.1 Distillate CompFraction = 0.95				

Table D-10. Validation of `calcNumberOfTrays` method in `DistillationColumn` class – part 2

Package	Class	Method Name	Input Parameter(s)		Java Output	Hand Solution	Comment
		calcNumberOfTrays	Equilibrium Data		Wrong value of Vapor equilibrium fraction entered! Entered value: [0.0,0.07,0.13,0.22, 0.33, 5.0,0.59,0.69,0.78, 0.86, 0.93,0.97,1.0] Permitted range: 0.0 - 1.0	1	Exception catchs out of range value in Java
			liquid mol frac.	vapour mol frac.			
			0	0			
			0.013	0.07			
			0.025	0.13			
			0.052	0.22			
			0.09	0.33			
			0.16	5			
			0.25	0.59			
			0.37	0.69			
			0.52	0.78			
			0.7	0.86			
			0.85	0.93			
			0.96	0.97			
			1	1			
			Bottoms CompFraction = 0.1 Distillate CompFraction = 0.95				
		calcNumberOfTrays	Equilibrium Data		Invalid data type for liquid equilibrium mole fraction	infinite	Exception catchs out of range value in Java
			liquid mol frac.	vapour mol frac.			
			0	0			
			0.013	0.07			
			0.025	0.13			
			0.052	0.22			
			0.09	0.33			
			f	0.47			
			0.25	0.59			
			0.37	0.69			
			0.52	0.78			
			0.7	0.86			
			0.85	0.93			
			0.96	0.97			
			1	1			
			Bottoms CompFraction = 0.1 Distillate CompFraction = 0.95				

Table D-11. Validation of `calcNumberOfTrays` method in `DistillationColumn` class – part 3

Package	Class	Method Name	Input Parameter(s)		Java Output	Hand Solution	Comment
		calcNumberOfTrays	Equilibrium Data		Invalid data type for vapour equilibrium mole fraction	8	Exception catchs out of range value in Java
			liquid mol frac.	vapour mol frac.			
			0	0			
			0.013	0.07			
			0.025	0.13			
			0.052	0.22			
			0.09	0.33			
			0.16	f			
			0.25	0.59			
			0.37	0.69			
			0.52	0.78			
			0.7	0.86			
			0.85	0.93			
			0.96	0.97			
			1	1			
			Bottoms CompFraction = 0.1 Distillate CompFraction = 0.95				
		calcNumberOfTrays	Equilibrium Data		1	1	No difference
			liquid mol frac.	vapour mol frac.			
			0	0			
			0.013	0.07			
			0.025	0.13			
			0.052	0.22			
			0.09	0.33			
			0.16	0.47			
			0.25	0.59			
			0.37	0.69			
			0.52	0.78			
			0.7	0.86			
			0.85	0.93			
			0.96	0.97			
			1	1			
			Bottoms CompFraction = 0.95 Distillate CompFraction = 0.1				

D.4 `process.flow` Package

Table D-12. Validation of the `calcStreamBP` method of the `FeedStream` class.

Class	Method	Input Parameter(s)	Java Output	Hand Solution	Comment
FeedStream	calcStreamBP	compFraction = [0.5, 0.5] compBP = [401, 489]	445.0	445	Java prints exactly one decimal position
		compFraction = [test, 0.5] compBP = [401, 489]	Invalid data type for component feed fraction Required: numeric	N/A	InvalidExcelCellException thrown in setter method
		compFraction = [0.5, test] compBP = [401, 489]	Invalid data type for component feed fraction Required: numeric	N/A	InvalidExcelCellException thrown in setter
		compFraction = [-99.9, 0.5] compBP = [401, 489]	Wrong value of feed fraction entered! Entered value: [-99.0, 0.5] Permitted range: 0.0 - 1.0	9201	OutOfRangeArgumentException thrown in setter method
		compFraction = [0.5, -99.9] compBP = [401, 489]	Wrong value of feed fraction entered! Entered value: [0.5, -99.9] Permitted range: 0.0 - 1.0	-48650.6	OutOfRangeArgumentException thrown in setter
		compFraction = [0.5, 0.5] compBP = [test, 489]	Invalid data type for component normal boiling point Required: numeric	N/A	InvalidExcelCellException thrown in setter method
		compFraction = [0.5, 0.5] compBP = [401, test]	Invalid data type for component normal boiling point Required: numeric	N/A	InvalidExcelCellException thrown in setter method
		compFraction = [0.5, 0.5] compBP = [-99.9, 489]	Wrong value of component normal boiling point entered! Entered value: [-99.0, 489.0] Permitted: non negative	199.55	OutOfRangeArgumentException thrown in setter
		compFraction = [0.5, 0.5] compBP = [401, -99.9]	Wrong value of component normal boiling point entered! Entered value: [401.0, -99.0] Permitted: non negative	150.55	OutOfRangeArgumentException thrown in setter

Table D-13. Validation of the `calcStreamCP` method of the `FeedStream` class.

Class	Method	Input Parameter(s)	Java Output	Hand Solution	Comment
FeedStream	calcStreamCP	compFraction = [0.5, 0.5] compCP = [142, 160]	151.0	151	Java prints exactly one decimal position
		compFraction = [test, 0.5] compCP = [142, 160]	Invalid data type for component feed fraction Required: numeric	N/A	InvalidExcelCellException thrown in setter method
		compFraction = [0.5, test] compCP = [142, 160]	Invalid data type for component feed fraction Required: numeric	N/A	InvalidExcelCellException thrown in setter
		compFraction = [-99.9, 0.5] compCP = [142, 160]	Wrong value of feed fraction entered! Entered value: [-99.0, 0.5] Permitted range: 0.0 - 1.0	-14105.8	OutOfRangeArgumentException thrown in setter method
		compFraction = [0.5, -99.9] compCP = [142, 160]	Wrong value of feed fraction entered! Entered value: [0.5, -99.0] Permitted range: 0.0 - 1.0	-15913	OutOfRangeArgumentException thrown in setter
		compFraction = [0.5, 0.5] compCP = [test, 160]	Invalid data type for component heat capacity Required: numeric	N/A	InvalidExcelCellException thrown in setter method
		compFraction = [0.5, 0.5] compCP = [142, test]	Invalid data type for component heat capacity Required: numeric	N/A	InvalidExcelCellException thrown in setter method
		compFraction = [0.5, 0.5] compCP = [-99.9, 160]	Wrong value of component heat capacity entered! Entered value: [-99.0, 489.0] Permitted: non negative	30.05	OutOfRangeArgumentException thrown in setter
		compFraction = [0.5, 0.5] compCP = [142, -99.9]	Wrong value of component heat capacity entered! Entered value: [401.0, -99.0] Permitted: non negative	21.05	OutOfRangeArgumentException thrown in setter

Table D-14. Validation of the `calcQ` method of the `FeedStream` class.

Class	Method	Input Parameter(s)	Java Output	Hand Solution	Comment
FeedStream	calcQ	LatentHeat = 17039 StreamCP = 151 StreamBP = 445	1.20559892012442	1	Java prints double precision, q is only used internally, so no number formatting is applied
		LatentHeat = test StreamCP = 151 StreamBP = 445	Invalid data type for latent heat Required: numeric	N/A	InvalidExcelCellException in setter
		LatentHeat = -99.9 StreamCP = 151 StreamBP = 445	Wrong value of latent heat entered! Entered value: -99.9 Permitted: non-negative	N/A	InvalidArgumentException

Table D-15. Validation of the `calcStreamBP` method of the `DistillateStream` class.

Class	Method	Input Parameter(s)	Java Output	Hand Solution	Comment
DistillateStream	calcStreamBP	compFraction = [0.95, 0.05] compBP = [401, 489]	405.4	405	
		compFraction = [test, 0.05] compBP = [401, 489]	Invalid data type for component distillate fraction Required: numeric	N/A	InvalidExcelCellException thrown in setter method
		compFraction = [0.95, test] compBP = [401, 489]	Invalid data type for component distillate fraction Required: numeric	N/A	InvalidExcelCellException thrown in setter
		compFraction = [-99.9, 0.05] compBP = [401, 489]	Wrong value of distillate fraction entered! Entered value: [-99.0, 0.05] Permitted range: 0.0 - 1.0	-39674.6	OutOfRangeArgumentException thrown in setter method
		compFraction = [0.95, -99.9] compBP = [401, 489]	Wrong value of distillate fraction entered! Entered value: [0.95, -99.0] Permitted range: 0.0 - 1.0	-48470	OutOfRangeArgumentException thrown in setter
		compFraction = [0.95, 0.05] compBP = [test, 489]	Invalid data type for component normal boiling point Required: numeric	N/A	InvalidExcelCellException thrown in setter method
		compFraction = [0.95, 0.05] compBP = [401, test]	Invalid data type for component normal boiling point Required: numeric	N/A	InvalidExcelCellException thrown in setter method
		compFraction = [0.95, 0.05] compBP = [-99.9, 489]	Wrong value of component normal boiling point entered! Entered value: [-99.0, 489.0] Permitted: non negative	-70.4	OutOfRangeArgumentException thrown in setter
		compFraction = [0.95, 0.05] compBP = [401, -99.9]	Wrong value of component normal boiling point entered! Entered value: [401.0, -99.0] Permitted: non negative	376	OutOfRangeArgumentException thrown in setter

Table D-16. Validation of the `calcStreamCP` method of the `DistillateStream` class.

Class	Method	Input Parameter(s)	Java Output	Hand Solution	Comment
DistillateStream	calcStreamCP	compFraction = [0.95, 0.05] compCP = [142, 160]	142.9	142.9	
		compFraction = [test, 0.05] compCP = [142, 160]	Invalid data type for component distillate fraction Required: numeric	N/A	InvalidExcelCellException thrown in setter method
		compFraction = [0.95, test] compCP = [142, 160]	Invalid data type for component distillate fraction Required: numeric	N/A	InvalidExcelCellException thrown in setter
		compFraction = [-99.9, 0.05] compCP = [142, 160]	Wrong value of distillate fraction entered! Entered value: [-99.0, 0.05] Permitted range: 0.0 - 1.0	-14177.8	OutOfRangeArgumentException thrown in setter method
		compFraction = [0.95, -99.9] compCP = [142, 160]	Wrong value of distillate fraction entered! Entered value: [0.95, -99.9] Permitted range: 0.0 - 1.0	-15849.1	OutOfRangeArgumentException thrown in setter
		compFraction = [0.95, 0.05] compCP = [test, 160]	Invalid data type for component heat capacity Required: numeric	N/A	InvalidExcelCellException thrown in setter method
		compFraction = [0.95, 0.05] compCP = [142, test]	Invalid data type for component heat capacity Required: numeric	N/A	InvalidExcelCellException thrown in setter method
		compFraction = [0.95, 0.05] compCP = [-99.9, 160]	Wrong value of component heat capacity entered! Entered value: [-99.0, 489.0] Permitted: non negative	-86.9	OutOfRangeArgumentException thrown in setter
		compFraction = [0.95, 0.05] compCP = [142, -99.9]	Wrong value of component heat capacity entered! Entered value: [401.0, -99.0] Permitted: non negative	129.9	OutOfRangeArgumentException thrown in setter

Table D-17. Validation of the `calcStreamBP` and `calcQ` method of the `BottomsStream` class.

Class	Method	Input Parameter(s)	Java Output	Hand Solution	Comment
DistillateStream	calcQ		-9999.0	N/A	Java prints a dummy value because q is not required for distillate stream for this project
BottomsStream	calcStreamBP	compFraction = [0.1, 9] compBP = [401, 489]	480.2	480	
		compFraction = [test, 0.9] compBP = [401, 489]	Invalid data type for component bottoms fraction Required: numeric	N/A	InvalidExcelCellException thrown in setter method
		compFraction = [0.1, test] compBP = [401, 489]	Invalid data type for component bottoms fraction Required: numeric	N/A	InvalidExcelCellException thrown in setter
		compFraction = [-99.9, 0.9] compBP = [401, 489]	Wrong value of bottoms fraction entered! Entered value: [-99.0, 0.9] Permitted range: 0.0 - 1.0	-39619.8	OutOfRangeArgumentException thrown in setter method
		compFraction = [0.1, -99.9] compBP = [401, 489]	Wrong value of bottoms fraction entered! Entered value: [0.1, -99.0] Permitted range: 0.0 - 1.0	-48370.9	OutOfRangeArgumentException thrown in setter
		compFraction = [0.1, 0.9] compBP = [test, 489]	Invalid data type for component normal boiling point Required: numeric	N/A	InvalidExcelCellException thrown in setter method
		compFraction = [0.1, 0.9] compBP = [401, test]	Invalid data type for component normal boiling point Required: numeric	N/A	InvalidExcelCellException thrown in setter method
		compFraction = [0.1, 0.9] compBP = [-99.9, 489]	Wrong value of component normal boiling point entered! Entered value: [-99.0, 489.0] Permitted: non negative	430.1	OutOfRangeArgumentException thrown in setter
		compFraction = [0.1, 0.9] compBP = [401, -99.9]	Wrong value of component normal boiling point entered! Entered value: [401.0, -99.0] Permitted: non negative	-49.8	OutOfRangeArgumentException thrown in setter

Table D-18. Validation of the `calcStreamCP` and `calcQ` method of the `BottomsStream` class.

Class	Method	Input Parameter(s)	Java Output	Hand Solution	Comment
BottomsStream	calcStreamCP	compFraction = [0.1, 0.9] compCP = [142, 160]	158.2	158.2	
		compFraction = [test, 0.9] compCP = [142, 160]	Invalid data type for component bottoms fraction Required: numeric	N/A	InvalidExcelCellException thrown in setter method
		compFraction = [0.1, test] compCP = [142, 160]	Invalid data type for component bottoms fraction Required: numeric	N/A	InvalidExcelCellException thrown in setter
		compFraction = [-99.9, 0.9] compCP = [142, 160]	Wrong value of bottoms fraction entered! Entered value: [-99.0, 0.9] Permitted range: 0.0 - 1.0	-14041.8	OutOfRangeArgumentException thrown in setter method
		compFraction = [0.1, -99.9] compCP = [142, 160]	Wrong value of bottoms fraction entered! Entered value: [0.1, -99.9] Permitted range: 0.0 - 1.0	-15969.8	OutOfRangeArgumentException thrown in setter
		compFraction = [0.1, 0.9] compCP = [test, 160]	Invalid data type for component heat capacity Required: numeric	N/A	InvalidExcelCellException thrown in setter method
		compFraction = [0.1, 0.9] compCP = [142, test]	Invalid data type for component heat capacity Required: numeric	N/A	InvalidExcelCellException thrown in setter method
		compFraction = [0.1, 0.9] compCP = [-99.9, 160]	Wrong value of component heat capacity entered! Entered value: [-99.0, 489.0] Permitted: non negative	134	OutOfRangeArgumentException thrown in setter
		compFraction = [0.1, 0.9] compCP = [142, -99.9]	Wrong value of component heat capacity entered! Entered value: [401.0, -99.0] Permitted: non negative	-75.7	OutOfRangeArgumentException thrown in setter
BottomsStream	calcQ		-9999.0	N/A	Java prints a dummy value because q is not required for distillate stream for this project

Table D-19. Validation of `calcSlope` and `calcIntercept` in the `Lines` class – part 1.

Class	Method	Input Parameter(s)	Java Output	Hand Solution	Comment
	calc Slope	RefluxRatio = 3.5	0.78	0.78	
		RefluxRatio = -2	validation.InvalidArgumentException: Wrong value of relfux ratio entered! Entered value: -2.0 Permitted: non negative	2	excel does not have exception handling, thus will use the unreasonable values to calculate slope
		RefluxRatio = 1000	1	1	
		RefluxRatio = reflux ratio	excelextractor.InvalidExcelCellException: Invalid data type for reflux ratio Required: numeric	#VALUE!	excel does not have exception handling, however as this is not a number input this would throw an error in excel
	calc Intercept	feed compFraction = 0.5 RefluxRatio = 3.5	0.21	0.21	
		feed compFraction = 10 RefluxRatio = 3.5	validation.OutOfRangeArgumentException: Wrong value of feed fraction entered! Entered value: [10.0, 0.5]. Permitted range: 0.0 - 1.0	0.21	the y values of the enriching line in excel is obtained using reflux ratio and distillate fraction only, thus changing the comp fraction would not affect the calculation
		feed compFraction = -0.5 RefluxRatio = 3.5	validation.OutOfRangeArgumentException: Wrong value of feed fraction entered! Entered value: [-0.5, 0.5]. Permitted range: 0.0 - 1.0	0.21	the y values of the enriching line in excel is obtained using reflux ratio and distillate fraction only, thus changing the comp fraction would not affect the calculation
		feed compFraction = feed RefluxRatio = 3.5	validation.OutOfRangeArgumentException: Wrong value of feed fraction entered! Entered value: [-0.5, 0.5]. Permitted range: 0.0 - 1.0	0.21	the y values of the enriching line in excel is obtained using reflux ratio and distillate fraction only, thus changing the comp fraction would not affect the calculation
		feed compFraction = 0.5 RefluxRatio = -2	Exception in thread "main" validation.InvalidArgumentException: Wrong value of reflux ratio entered! Entered value: -2.0. Permitted: non negative	-0.95	excel does not have exception handling, thus will use the unreasonable values to calculate intercept

Table D-20. Validation of `calcSlope` and `calcIntercept` in the `Lines` class – part 2.

Class	Method	Input Parameter(s)	Java Output	Hand Solution	Comment
Qline	calc Slope	feed compFraction = 0.5 calcQ value = (get from column)	5.86	5.86	
		feed compFraction = 10 calcQ value = (get from column)	validation.OutOfRangeArgumentException: Wrong value of feed fraction entered! Entered value: [10.0, 0.5]. Permitted range: 0.0 - 1.0	2.05	excel does not have exception handling, thus will use the unreasonable values to calculate slope
		feed compFraction = feed calcQ value = (get from column)	Exception in thread "main" excelextractor.InvalidExcelCellException: Invalid data type for component feed fraction Required: numeric	#VALUE!	excel does not have exception handling, however as this is not a number input this would throw an error in excel
	calc Intercept	feed compFraction = 0.5 calcQ value = (get from column)	-2.43	-2.43	
		feed compFraction = 10 calcQ value = (get from column)	validation.OutOfRangeArgumentException: Wrong value of feed fraction entered! Entered value: [10.0, 0.5]. Permitted range: 0.0 - 1.0	-10.48	excel does not have exception handling, thus will use the unreasonable values to calculate intercept
		feed compFraction = feed calcQ value = (get from column)	Exception in thread "main" excelextractor.InvalidExcelCellException: Invalid data type for component feed fraction Required: numeric	#VALUE!	excel does not have exception handling, however as this is not a number input this would throw an error in excel

Table D-21. Validation of `calcSlope` and `calcIntercept` in the `Lines` class – part 3.

Class	Method	Input Parameter(s)	Java Output	Hand Solution	Comment
Stripping Line	calc Slope	intersection cooridinate value = (get from column) feed compFraction = 0.5	1.23	1.23	
		intersection cooridinate value = (get from column) feed compFraction = -2	validation.OutOfRangeArgumentException: Wrong value of feed fraction entered! Entered value: [-2.0, 0.5]. Permitted range: 0.0 - 1.0	0.58	excel does not have exception handling, thus will use the unreasonable values to calculate slope
		intersection cooridinate value = (get from column) feed compFraction = feed	Exception in thread "main" excelextractor.InvalidExcelCellException: Invalid data type for component feed fraction Required: numeric	#VALUE!	excel does not have exception handling, however as this is not a number input this would throw an error in excel
	calc Intercept	slope value = (get from calcSlope) intersection value = (get from column)	-0.02	-0.02	
		slope value = (get from calcSlope) when feed comFraction = -2 intersection value = (get from column)	validation.OutOfRangeArgumentException: Wrong value of feed fraction entered! Entered value: [-2.0, 0.5]. Permitted range: 0.0 - 1.0	0.04	excel does not have exception handling, thus will use the unreasonable values to calculate intercept
		slope value = (get from calcSlope) when feed compFraction = feed intersection value = (get from column)	Exception in thread "main" excelextractor.InvalidExcelCellException: Invalid data type for component feed fraction Required: numeric	#VALUE!	excel does not have exception handling, however as this is not a number input this would throw an error in excel

D.5 `costing` Package

Table D-22. Validation of `calcCapitalPurchase` method for the column in `ModuleCosting` class.

Class	Method Name	Input Parameter(s)	Java Output	Hand Solution	Comment
ModuleCosting	calcCapitalPurchase(Column)	diameter = 2	$38,229	$38,229	java solution prints exact same value
	calcCapitalPurchase(Column)	diameter = 0	The column diameter is outside the applicable range. Range Required: 0m< diameter<=4m	N/A	exception thrown in to `CostingException`
	calcCapitalPurchase(Column)	diameter = 5	The column diameter is outside the applicable range. Range Required: 0m< diameter<=4m	N/A	exception thrown in to `CostingException`

Table D-23. Validation of `calcFp` method for the column in `ModuleCosting` class.

Class	Method Name	Input Parameter(s)	Java Output	Hand Solution	Comment
ModuleCosting	calcFp(DistillationColumn)	gaugePressure = 16.4	[1.0, 3.78]	[1.0, 3.78]	java solution prints exact same value
	calcFp(DistillationColumn)	gaugePressure = 2.0	[1.0, 1.0]	[1.0, 1.0]	java solution prints exact same value
	calcFp(DistillationColumn)	guagePressure = -1.0	[1.0, 1.25]	[1.0, 1.25]	java solution prints exact same value
	calcFp(DistillationColumn)	gaugePressure = 500	The column pressure is outside the applicable range. Range Required: pressure <=400psig	N/A	exception thrown in to `CostingException`

Table D-24. Validation of `calcFm` method for the column in `ModuleCosting` class.

Class	Method Name	Input Parameter(s)	Java Output	Hand Solution	Comment
ModuleCosting	calcFm(DistillationColumn)	material = titanium	[1, 10.6]	[1, 10.6]	java solution prints exact same value
	calcFm(DistillationColumn)	material = titanium clad	[1, 4.9]	[1, 4.9]	java solution prints exact same value
	calcFm(DistillationColumn)	material = carbon steel	[1, 1]	[1, 1]	java solution prints exact same value
	calcFm(DistillationColumn)	material = plastic	The selectted column material does not have any associated costing coefficients. Fix the Excel file and try again	N/A	exception thrown in to CostingException

Table D-25. Validation of `calcBareModule` method for the column in `ModuleCosting` class.

Class	Method Name	Input Parameter(s)	Java Output	Hand Solution	Comment
ModuleCosting	calcBareModule(DistillationColumn)	calcFm = (1,10.6) calcFp = (1,87)	[$2121580.07, $244823.10]	[$2121580.07, $244823.10]	java solution prints exact same value
	calcBareModule(DistillationColumn)	calcFm = (1,4.9) calcFp = (1,1)	[$633987.40, $244823.10]	[$633987.40, $244823.10]	java solution prints exact same value
	calcBareModule(DistillationColumn)	calcFm = (1,1) calcFp = (1,1.25)	[$522227.39, $244823.10]	[$522227.39, $244823.10]	java solution prints exact same value

Table D-26. Validation of `calcBareModule` method for the trays in `ModuleCosting` class.

Class	Method Name	Input Parameter(s)	Java Output	Hand Solution	Comment
ModuleCosting	calcBareModule(Tray)	tray number = 6 material = nickel alloy	[$269769.53, $244823.10]	[$269769.53, $244823.10]	java solution prints exact same value
	calcBareModule(Tray)	tray number = 19 material = carbon steel	[$9421.95, $15828.88.10]	[$9421.95, $15828.88.10]	java solution prints exact same value
	calcBareModule(Tray)	tray number = -1 material = carbon steel	Error while determining Fq value for tray costing	N/A	Java shows error message, but hand solution has nothing to show
	calcBareModule(Tray)	tray number = 2 material = glass	The selected tray material does not have any associated costing coefficients. Fix the Excel file and try again	N/A	Java shows error message, but hand solution has nothing to show

Table D-27. Validation of `calcGrassRoot` method for the column in `ModuleCosting` class.

Class	Method Name	Input Parameter(s)	Java Output	Hand Solution	Comment
ModuleCosting	calcGrassRoot(DistillationColumn)	calcBareModule = [2121580.07, 244823.10]	$2,625,876.04	$2,625,876.05	java solution prints exact same value
	calcGrassRoot(DistillationColumn)	calcBareModule = [4143251.51, 244823.10]	$870,516.68	$870,516.69	java solution prints exact same value
	calcGrassRoot(DistillationColumn)	calcBareModule = [522227.39,244823.10]	$738,639.87	$738,639.88	java solution prints exact same value

Table D-28. Validation of `calcGrassRoot` method for the trays in `ModuleCosting` class.

Class	Method Name	Input Parameter(s)	Java Output	Hand Solution	Comment
ModuleCosting	calcGrassRoot(Tray)	calcBareModule = [269769.53,244823.10]	$68,047	$68,047	no difference
	calcGrassRoot(Tray)	calcBareModule = [9421.95,15828.88.10]	$15,828	$15,828	no difference
	calcGrassRoot(Tray)	calcBareModule = N/A	N/A	0	java occurs system error
	calcGrassRoot(Tray)	calcBareModule = N/A	N/A	0	java occurs system error

Table D-29. Validation of `calcGrassRootTotal` method in `ModuleCosting` class.

Class	Method Name	Input Parameter(s)	Java Output	Hand Solution	Comment
ModuleCosting	calcGrassRootTotal(DistillationColumn)	calcGrassRoot(DistillationColumn) = 2625876.04 calcGrassRoot(Tray) = 68048	$2,693,923.47	$2,693,923.48	no difference
	calcGrassRootTotal(DistillationColumn)	calcGrassRoot(DistillationColumn) = 870516.68 calcGrassRoot(Tray) = 15829	$886,345.56	$886,346.56	no difference
	calcGrassRootTotal(DistillationColumn)	calcGrassRoot(DistillationColumn) = 738639 calcGrassRoot(Tray) = N/A	N/A	$738,639	java occurs system error

D.6 `excelextractor` Package

Table D-30. Validation of the `extractEq` method of the `ExcelExtractor` class.

Class	Method	Input Parameter(s)	Java Output	Hand Solution	Comment
ExcelExtractor	extractEq Case 1 Expected values	Equilibrium liq frac (excel)	EqX[]	N/A This method extracts the equilibrium data from excel, it does not calculate anything. The purpose of this validation is to ensure the correct values are imported from the venture datasheet, before assignment to double arrays eqX[] and eqY[]	Java prints an extra decimal place at 0 and 1 Note: method input parameters are a String that is the name of the file, and an int that is the worksheet number The mole fractions are being changed in the excel file, not in the method parameter list
		0	0.0		
		0.013	0.013		
		0.025	0.025		
		0.052	0.052		
		0.09	0.09		
		0.16	0.16		
		0.25	0.25		
		0.37	0.37		
		0.52	0.52		
		0.7	0.70		
		0.85	0.85		
		0.96	0.96		
		1	1.0		
		Equilibrium vap frac (excel)	EqY[]		
		0	0.0		
		0.07	0.07		
		0.13	0.13		
		0.22	0.22		
		0.33	0.33		
		0.47	0.47		
		0.59	0.59		
		0.69	0.69		
		0.78	0.78		
		0.86	0.86		
		0.93	0.93		
		0.97	0.97		
		1	1.0		

Table D-31. Validation of the `extractEq` method for out of range liquid mole fraction of the `ExcelExtractor` class.

Class	Method	Input Parameter(s)	Java Output	Hand Solution	Comment
ExcelExtractor	extractEq Case 2 Liquid mole fraction out of range Range: 0 <= x <=1	Equilibrium liq frac (excel)	EqX[], EqY[] not assigned		Java returned OutOfRangeArgumentException because a liquid mole fraction of 5 is not permitted The range of equilibrium is checked first before being assigned to instance variables Note: method input parameters are a String that is the name of the file, and an int that is the worksheet number The mole fractions are being changed in the excel file, not in the method parameter list
		0	validation.OutofRangeArgumentException: Wrong value of liquid equilibrium entered! Entered value: [0.0, 0.013, 0.025, 0.052, 5.0, 0.16, 0.25, 0.37, 0.52, 0.7, 0.85, 0.96, 1.0]. Permitted range: 0.0 - 1.0		
		0.013			
		0.025			
		0.052			
		5			
		0.16			
		0.25			
		0.37			
		0.52			
		0.7			
		0.85			
		0.96			
		1			
		Equilibrium vap frac (excel)			
		0			
		0.07			
		0.13			
		0.22			
		0.33			
		0.47			
		0.59			
		0.69			
		0.78			
		0.86			
		0.93			
		0.97			
		1			

Table D-32. Validation of the `extractEq` method for out of range vapor mole fraction of the `ExcelExtractor` class.

Class	Method	Input Parameter(s)	Java Output	Hand Solution	Comment
ExcelExtractor	extractEq Case 2 Vapor mole fraction out of range Range: 0 <= x <=1	Equilibrium liq frac (excel)	EqX[], EqY[] not assigned		Java returned OutOfRangeArgumentException because a vapor mole fraction of 9999 is not permitted The range of equilibrium is checked first before being assigned to instance variables Note: method input parameters are a String that is the name of the file, and an int that is the worksheet number The mole fractions are being changed in the excel file, not in the method parameter list
		0	Wrong value of Vapor equilibrium entered! Entered value: [0.0, 0.07, 0.13, 0.22, 0.33, 0.47, 0.59, 0.69, 9999.0, 0.86, 0.93, 0.97, 1.0]. Permitted range: 0.0 - 1.0		
		0.013			
		0.025			
		0.052			
		0.09			
		0.16			
		0.25			
		0.37			
		0.52			
		0.7			
		0.85			
		0.96			
		1			
		Equilibrium vap frac (excel)			
		0			
		0.07			
		0.13			
		0.22			
		0.33			
		0.47			
		0.59			
		0.69			
		9999			
		0.86			
		0.93			
		0.97			
		1			

Table D-33. Validation of the `extractEq` method for wrong data type input for the liquid mole fraction of the `ExcelExtractor` class.

Package	Class	Method	Input Parameter(s)	Java Output	Hand Solution	Comment
excelextractor	ExcelExtractor	extractEq Case 3 Liquid mole fraction wrong data type Required datatype: any number type	Equilibrium liq frac (excel)	EqX[], EqY[] not assigned		Java returned InvalidExcelCellException because a liquid mole fraction of TEST is not permitted The datatype of equilibrium is checked first before being assigned to instance variables Note: method input parameters are a String that is the name of the file, and an int that is the worksheet number The mole fractions are being changed in the excel file, not in the method parameter list
			0	Invalid data type for liquid equilibrium mole fraction Required: numeric		
			0.013			
			0.025			
			0.052			
			0.09			
			0.16			
			0.25			
			0.37			
			0.52			
			TEST			
			0.85			
			0.96			
			1			
			Equilibrium vap frac (excel)			
			0			
			0.07			
			0.13			
			0.22			
			0.33			
			0.47			
			0.59			
			0.69			
			0.78			
			0.86			
			0.93			
			0.97			
			1			

Table D-34. Validation of the `extractEq` method for wrong data type input for the vapor mole fraction of the `ExcelExtractor` class.

Package	Class	Method	Input Parameter(s)	Java Output	Hand Solution	Comment
excelextractor	ExcelExtractor	extractEq Case 3 Vapour mole fraction wrong data type Required datatype: any number type	Equilibrium liq frac (excel)	EqX[], EqY[] not assigned		Java returned InvalidExcelCellException because a liquid mole fraction of TEST is not permitted The datatype of vapor equilibrium is checked first before being assigned to instance variables Note: method input parameters are a String that is the name of the file, and an int that is the worksheet number The mole fractions are being changed in the excel file, not in the method parameter list
			0	Invalid data type for vapour equilibrium mole fraction Required: numeric		
			0.013			
			0.025			
			0.052			
			0.09			
			0.16			
			0.25			
			0.37			
			0.52			
			0.7			
			0.85			
			0.96			
			1			
			Equilibrium vap frac (excel)			
			0			
			0.07			
			TEST			
			0.22			
			0.33			
			0.47			
			0.59			
			0.69			
			0.78			
			0.86			
			0.93			
			0.97			
			1			

Table D-35. Validation of the `prompt` method of the `ExcelExtractor` class.

Package	Class	Method	Input Parameter(s)	Java Output	Hand Solution	Comment
excelextractor	ExcelExtractor	prompt	Scanner input filepath = Venture.xls sheetNum = 1	The number of trays is 6 The total grassroots is $2,693,923.47 The yearly profit based on production is $33,236,470.59/year The total profit is $30,542,547.12 over a one year payback period. Would you like to analyze another venture? (Y)es or (N)o:		With a valid filepath and sheetNum, the program runs to completion
			Scanner input filepath = TEST sheetNum = 1	File does not exist!		InvalidExcelFileException thrown
			Scanner input filepath = Venture.xls sheetNum = 999	Sheet number is larger than the total sheets this file contains!		InvalidExcelSheetException thrown

Table D-36. Validation of the `extractVentureData` method of the `ExcelExtractor` class – part 1.

Package	**Class**	**Method Name**	Description: The extractVentureData method takes in as parameters a String which specifies a file name and an integer which specifies a worksheet number. Those parameters will be tested separately - i.e. to test if an invalid workbook and sheet number are entered. The following validation is for the excel file data. Each value is changed manually in excel, then the method is tested to see how it handles data that is out of range and of wrong data type.		
excelextractor	ExcelExtractor	extractVentureData			
Excel cell	**Cell Description**	**Expected**	**Input Parameter**	**Java Output**	**Comments**
B3	Feed Flow Rate	positive numeric	100	100.0	Java prints exactly one decimal position
			test	Invalid data type for feed flow rate Required: numeric	excelextractor.InvalidExcelCellException
			-99.9	Invalid data type for feed flow rate Required: numeric Wrong value of feed flow rate entered! Entered value: -99.0 Permitted: non negative	excelextractor.InvalidExcelCellException validation.InvalidArgumentException
B6	Feed Fraction	numeric, 0-1	0.5	0.5	Java prints exactly one decimal position
	component 1		test	Invalid data type for component feed fraction Required: numeric	excelextractor.InvalidExcelCellException
			-99.9	Invalid data type for component feed fraction Required: numeric Wrong value of feed fraction entered! Entered value: [-99.9, 0.5] Permitted range: 0.0 - 1.0	excelextractor.InvalidExcelCellException OutOfRangeArgumentException
C6	Feed Fraction	numeric, 0-1	0.5	0.5	
	component 2		test	Invalid data type for component feed fraction Required: numeric	excelextractor.InvalidExcelCellException
			-99.9	Invalid data type for component feed fraction Required: numeric Wrong value of feed fraction entered! Entered value: [0.5, -99.9] Permitted range: 0.0 - 1.0	excelextractor.InvalidExcelCellException OutOfRangeArgumentException
B7	Distillate Fraction	numeric, 0-1	0.95	0.95	
	component 1		test	Invalid data type for component distillate fraction Required: numeric	excelextractor.InvalidExcelCellException
				Invalid data type for component distillate fraction Required: numeric	excelextractor.InvalidExcelCellException
			-99.9	Wrong value of Distillate fraction entered! Entered value: [-99.9, 0.05] Permitted range: 0.0 - 1.0	OutOfRangeArgumentException
C7	Distillate Fraction	numeric, 0-1	0.05	0.05	
	component 2		test	Invalid data type for component distillate fraction Required: numeric	excelextractor.InvalidExcelCellException
			-99.9	Invalid data type for component distillate fraction Required: numeric Wrong value of Distillate fraction entered! Entered value: [0.05, -99.9] Permitted range: 0.0 - 1.0	excelextractor.InvalidExcelCellException OutOfRangeArgumentException

Table D-37. Validation of the `extractVentureData` method of the `ExcelExtractor` class – part 2.

Package	Class	Method Name	**Description:** The extractVentureData method takes in as parameters a String which specifies a file name and an integer which specifies a worksheet number. Those parameters will be tested separately - i.e. to test if an invalid workbook and sheet number are entered. The following validation is for the excel file data. Each value is changed manually in excel, then the method is tested to see how it handles data that is out of range and of wrong data type.			
excelextractor	ExcelExtractor	extractVentureData			
Excel cell	**Cell Description**	**Expected**	**Input**	**Java Output**	**Comments**
B8	Bottoms Fraction	numeric, 0-1	0.1	0.1	
	component 1		test	Invalid data type for component bottoms fraction Required: numeric	excelextractor.InvalidExcelCellException
			-99.9	Invalid data type for component bottoms fraction Required: numeric Wrong value of Bottoms fraction entered! Entered value: [-99.9, 0.9] Permitted range: 0.0 - 1.0	excelextractor.InvalidExcelCellException OutOfRangeArgumentException
C8	Bottoms Fraction	numeric, 0-1	0.9	0.9	
	component 2		test	Invalid data type for component distillate fraction Required: numeric	excelextractor.InvalidExcelCellException
			-99.9	Invalid data type for component distillate fraction Required: numeric Wrong value of Distillate fraction entered! Entered value: [0.1, -99.9] Permitted range: 0.0 - 1.0	excelextractor.InvalidExcelCellException OutOfRangeArgumentException
B11	Cp	positive numeric	142	142.0	java prints exactly one decimal position
	component 1		test	Invalid data type for component heat capacity Required: numeric	excelextractor.InvalidExcelCellException
				Invalid data type for component heat capacity Required: numeric	excelextractor.InvalidExcelCellException
			-99.9	Wrong value of Distillate fraction entered! Entered value: [-99.9, 160.0] Permitted: non negative	OutOfRangeArgumentException
C11	Cp	positive numeric	160	160.0	java prints exactly one decimal position
	component 2		test	Invalid data type for component heat capacity Required: numeric	excelextractor.InvalidExcelCellException
			-99.9	Invalid data type for component heat capacity Required: numeric Wrong value of normal BP entered! Entered value: [142.0, -99.9] Permitted: non negative	excelextractor.InvalidExcelCellException OutOfRangeArgumentException
B12	Normal BP	positive numeric	401	401.0	java prints exactly one decimal position
	component 1		test	Invalid data type for component normal boiling point Required: numeric	excelextractor.InvalidExcelCellException
			-99.9	Invalid data type for component normal boiling point Required: numeric Wrong value of normal BP entered! Entered value: [-99.9, 489.0] Permitted: non negative	excelextractor.InvalidExcelCellException OutOfRangeArgumentException

Table D-38. Validation of the `extractVentureData` method of the `ExcelExtractor` class – part 3.

Package	Class	Method Name	**Description:** The extractVentureData method takes in as parameters a String which specifies a file name and an integer which specifies a worksheet number. Those parameters will be tested separately - i.e. to test if an invalid workbook and sheet number are entered. The following validation is for the excel file data. Each value is changed manually in excel, then the method is tested to see how it handles data that is out of range and of wrong data type.		
excelextractor	ExcelExtractor	extractVentureData			
Excel cell	**Cell Description**	**Expected**	**Input Parameter**	**Java Output**	**Comments**
C12	Normal BP	positive numeric	489	489.0	java prints exactly one decimal position
	component 2		test	Invalid data type for component normal boiling point Required: numeric	excelextractor.InvalidExcelCellException
			-99.9	Invalid data type for component normal boiling point Required: numeric Wrong value of Distillate fraction entered! Entered value: [401.0, -99.9] Permitted: non negative	excelextractor.InvalidExcelCellException OutOfRangeArgumentException
B13	Latent heat	positive numeric	17039	17039.0	java prints exactly one decimal position
			test	Invalid data type for latent heat Required: numeric	excelextractor.InvalidExcelCellException
				Invalid data type for latent heat Required: numeric	excelextractor.InvalidExcelCellException
			-99.9	Wrong value of latent heat entered! Entered value: -99.9 Permitted: non negative	OutOfRangeArgumentException
B15	Diameter	positive numeric	2	2.0	java prints exactly one decimal position
			test	Invalid data type for diameter Required: numeric	excelextractor.InvalidExcelCellException
				Invalid data type for diameter Required: numeric	excelextractor.InvalidExcelCellException
			-99.9	Wrong value of diameter entered! Entered value: -99.9 Permitted: non negative	OutOfRangeArgumentException:
B16	Length	positive numeric	10	10.0	java prints exactly one decimal position
			test	Invalid data type for length Required: numeric	excelextractor.InvalidExcelCellException
			-99.9	Invalid data type for lenth Required: numeric Wrong value of length entered! Entered value: -99.9 Permitted: non negative	excelextractor.InvalidExcelCellException OutOfRangeArgumentException
B17	Gauge Pressure	positive numeric	16.4	16.4	
			test	Invalid data type for pressure Required: numeric	excelextractor.InvalidExcelCellException
			-99.9	Invalid data type for pressure Required: numeric Wrong value of pressure entered! Entered value: -99.9 Permitted: non negative	excelextractor.InvalidExcelCellException OutOfRangeArgumentException

Table D-39. Validation of the `extractVentureData` method of the `ExcelExtractor` class – part 4.

Package	Class	Method Name	**Description:** The extractVentureData method takes in as parameters a String which specifies a file name and an integer which specifies a worksheet number. Those parameters will be tested separately - i.e. to test if an invalid workbook and sheet number are entered. The following validation is for the excel file data. Each value is changed manually in excel, then the method is tested to see how it handles data that is out of range and of wrong data type.		
excelextractor	ExcelExtractor	extractVentureData			
Excel cell	**Cell Description**	**Expected**	**Input Parameter**	**Java Output**	**Comments**
B18	Reflux Ratio	positive numeric	3.5	3.5	
			test	Invalid data type for reflux ratio Required: numeric	excelextractor.InvalidExcelCellException
				Invalid data type for reflux ratio Required: numeric	excelextractor.InvalidExcelCellException
			-99.9	Wrong value of reflux ratio entered! Entered value: -99.9 Permitted: non negative	OutOfRangeArgumentException
B19	Inlet Temperature	positive numeric	421.8	421.8	
			test	Invalid data type for temperature Required: numeric	excelextractor.InvalidExcelCellException
				Invalid data type for temperature Required: numeric	excelextractor.InvalidExcelCellException
			-99.9	Wrong value of temperature entered! Entered value: -99.9 Permitted: non negative	OutOfRangeArgumentException
B20	MOC Column	String	Titanium	Titanium	
			test	test	Validation of material is done in costing
			-99.9	Invalid data type for column material of construction Required: String	excelextractor.InvalidExcelCellException
				Invalid data type for column material of construction Required: String	excelextractor.InvalidExcelCellException
B21	MOC Trays	String	Nickel Alloy	Titanium	
			test	test	Validation of material is done in costing
			-99.9	Invalid data type for tray material of construction Required: String	excelextractor.InvalidExcelCellException
				Invalid data type for tray material of construction Required: String	excelextractor.InvalidExcelCellException
B22	Tray Type	String	Sieve	Sieve	
			test	test	Validation of material is done in costing
			-99.9	Invalid data type for tray type Required: String	excelextractor.InvalidExcelCellException
				Invalid data type for tray typeRequired: String	excelextractor.InvalidExcelCellException

Table D-40. Validation of the `extractVentureData` method of the `ExcelExtractor` class – part 5.

Package	**Class**	**Method Name**	**Description:** The extractVentureData method takes in as parameters a String which specifies a file name and an integer which specifies a worksheet number. Those parameters will be tested separately - i.e. to test if an invalid workbook and sheet number are entered. The following validation is for the excel file data. Each value is changed manually in excel, then the method is tested to see how it handles data that is out of range and of wrong data type.		
excelextractor	ExcelExtractor	extractVentureData			
Excel cell	**Cell Description**	**Expected**	**Input Parameter** (from excel)	**Java Output** (exactly what is printed out to the console)	**Comments**
B24	Unit cost Raw Material	positive numeric	25	25.0	java prints exactly one decimal position
			test	Invalid data type for raw material cost Required: numeric	excelextractor.InvalidExcelCell Exception
			-99.9	Invalid data type for raw material cost Required: numeric Wrong value of raw material cost entered! Entered value: -99.9 Permitted: non negative	excelextractor.InvalidExcelCell Exception OutOfRangeArgumentExceptio n
B25	Unit Sale Price of Distillate	positive numeric	100	100.0	java prints exactly one decimal position
			test	Invalid data type for distillate sale price Required: numeric	excelextractor.InvalidExcelCell Exception
				Invalid data type for distillate sale price Required: numeric	excelextractor.InvalidExcelCell Exception
			-99.9	Wrong value of distillate sale price entered! Entered value: -99.9 Permitted: non negative	OutOfRangeArgumentExceptio n
B26	Unit Sale Price of Bottoms	positive numeric	30	30.0	java prints exactly one decimal position
			test	Invalid data type for bottoms sale price Required: numeric	excelextractor.InvalidExcelCell Exception
				Invalid data type for bottoms sale price Required: numeric	excelextractor.InvalidExcelCell Exception
			-99.9	Wrong value of bottoms sale price entered! Entered value: -99.9 Permitted: non negative	OutOfRangeArgumentExceptio n

E Task Allocation

Table E-1. Task allocation summary of Group 7.

Name	Task Allocated (package responsible: code and write-up)
Tatum Alenko	`numerical.analysis` `numerical.interpolation`
Sandra Kari	`jexcel.excelextractor` `process.flow`

Jordon Kay	`process.unitoperations`
	`.Column`
	`.DistillationColumn`
	`.McCabeThiele`
	`.Tray`
	`.SieveTray`

Janice Leung	`process.unitoperations` (half)
	`.OperatingLine`
	`.StrippingLine`
	`.EnrichingLine`
	`.QLine`

Yun Song	`costing`

References

Chapra, S., & Canale, R. (2009). *Numerical Methods for Engineers*. McGraw-Hill.

Fritsch, F., & Carlson, R. (1980). Monotone Piecewise Cubic Interpolation. *SIAM Journal on Numerical Analysis*(17), 238-246.

Khan, A. (2015). *SourceForce*. Retrieved from Java Excel API: http://jexcelapi.sourceforge.net

MathWorks. (2015). *Piecewise Cubic Hermite Interpolating Polynomial (PCHIP)*. Retrieved from MATLAB Documentation: http://www.mathworks.com/help/matlab/ref/pchip.html

Savitch, W. (2013). *Absolute Java*. New Jersey: Pearson Education, Inc.

Seider, J., Henley, E. J., & Roper, D. K. (2011). *Separation Process Principles*. John Wilet & Sons, Inc.