De Wet Reitz Simulation and Economic Analysis Model

by

Vickus Rudolph Botha

29129797

Final Document

Submitted in partial fulfilment of the requirements for the degree of

BACHELORS OF INDUSTRIAL ENGINEERING

in the

FACULTY OF ENGINEERING, BUILT ENVIRONMENT AND INFORMATION TECHNOLOGY

UNIVERSITY OF PRETORIA

17 October 2012

Executive Summary

De Wet Reitz requires a model to simulate the amount of property conveyance tasks that will be generated at ideal conditions, from the joint venture of a real estate company. The amount of conveyance tasks are determined by a “series” predetermined by De Wet Reitz. These conveyance tasks will generate income; thus resources will be needed to process the conveyance tasks because at present the resources are not optimal and also not sufficient.

The simulation of the amount of tasks will be performed by a Monte Carlo simulation that will also display different scenarios. The inputs of the Monte Carlo simulation will be able to be changed as to generate different scenarios.

This joint venture project requires investment from other capital sources; therefore a need exists for a model to estimate the break-even point and the capital needed before the break-even point is achieved. In addition there is a need for the optimal amount of resources to optimise the workload and maximise the income relative to the cost. This can be accomplished through a cash flow projection method and calculations completed from the projections. The projections is also required by De Wet Reitz.

The model will be built in Java. Java is able to accommodate these simulation needs together with the solutions, for building the simulation model. The model is completed in Java and will accommodate the average income per branch from De Wet Reitz.

The model will simulate, optimise and visualise the cash flow of De Wet Reitz and through this method De Wet Reitz will be able to determine whether the scenarios are feasible. The result of the simulation will be measured against De Wet Reitz expectations, if the models’ output shows less than expected management will be able to change inputs and scenarios until the desired effect is achieved, for verification, and what factors should be changed and managed to achieve the desired result.

This model may also be edited for future implementation into other De Wet Reitz projects. The model will ultimately simulate the profit and display it as a projection. De Wet Reitz will then be able to identify the number of resources that should be allocated at a specific time, the amount of capital needed for the project and if the project is feasible within a 60 month period.

Table of Contents

Chapter 4 - Results

Chapter 5 - Conclusion

List of Figures

List of Tables

Chapter 1

Introduction

1.1 De Wet Reitz

Located in Linden, Johannesburg, De Wet Reitz (DWR) is a company that specialises in property conveyance (transferring ownership from one owner to another). DWR does the property conveyance for corporate institutions and was recently approached by a new property agency that will guarantee a certain number of conveyance tasks. This joint venture creates possibilities for future business.

1.2 The Problem

The property agency will guarantee a certain number of conveyance tasks over a period of 60 months. These tasks are received from 15 property agency branches that are performing at a functional level. DWR cannot charge normal rates for these conveyance tasks as there is a tariff rate of R3 500 per conveyance task (determined from the agreement). This is where the guarantee is coupled in from the 15 branches of the 60 month period, where a certain amount of conveyance tasks will be given. Fees are also collected per branch depending on the purchase price of the house.

Thus a certain amount of tasks per property agency branch has to be completed to break-even or make a profit. As the property agency is expanding the tasks will increase adding pressure on current resources, which leads to the amount of resources needed to complete the amount of work. This also leads to the amount of capital needed to accommodate the amount of work that has been created from the expanding property agency.

Together with maximizing the profit for the future, the resources will have to be adequately allocated for DWR to be able to support the growing business. The objective is to make maximum profit with a reasonable amount of allocated resources not only to be able to support current business operations but also be able to adapt accordingly for the future.

1.3 The Objective

The project consists of a simulation model that will predict the amount of capital needed to finance the DWR project before the break-even point has been reached, also to predict the amount of income for the DWR project. The model will incorporate an algorithm to determine when and how many attorneys and paralegals have to be allocated at a specific time; the model will perform in a similar manner on the allocation of other resources (optimisation model).

The model will create a cash flow projection for the 60 month period to determine whether the project is feasible or not. The model will be created in Java. Java is a strong platform for creating the model as various simulation models are built using the Java platform.

Overall, the model will balance the income of the future with the resources that are needed to accommodate the amount of work. The amount of work that is generated will be spread amongst the resources (paralegals). Determining the future projections of the amount of conveyance tasks, will be completed by the model.

1.4 Scope

The scope of the project will cover the entire DWR business service. All cost and incomes will be taken into account; this includes salaries, overheads, subscription fees, membership fees and business costs. All income will be taken into account, not only the property agency's fees.

The cost structures of the DWR business will be able to be changed within the model. Costs structures like the amount of people to be allocated according to the amount of paralegals and amount of branches that are open at a current time. Finance personal and managers within DWR will be able to perform market research and incorporate the findings into the simulation model. This will be in terms of average purchase price per branch, growth within the business also incorporating tax rates.

1. 5 Detail of the Project

DWR uses its existing infrastructure to complete the work and will only need to hire more workers if the amount of work increases.

The guaranteed tasks over the next 60 months are represented in the following series:
{0, 0, 2, 5, 7, 10, 11, 12, 13, 15, 17, 19, 19, 20, 22, 25, 25, 28, 28, 30, 30, 30, 30, 30, 30, 32, 32, 33, 33, 35, 35, 35, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37}

This indicates that in month 1 and 2 no tasks will be given, in the 3rd month 2 tasks will be given, 5 tasks in the 4th month will be given and so on.

Over the next 9 months on 3 month basis, a further 9 branches will open (this indicates that every 3 months 3 branches will be opened and the tasks that will be received from the new branches will follow the same pattern as given.

The professional work that is done is completed by highly qualified law paralegals; each paralegal can accommodate 50 active files. Their salaries range from R 22 000 to R 24 000 per month. Furthermore, a qualified attorney is at the disposal of the property agency to provide law advice. The attorneys' salaries range from R30 000 to R40 000 per month, where an attorney is allocated towards 3 branches.

1.6 The operations

The task of the paralegals is to process the "buy contract" and to gather the necessary personal information and law information. The paralegals also have to arrange for the payment of the buy price, the instalments for the current home loans and placing the property in the buyers' name.

The observation has to be made that not all tasks will proceed. Most contracts can only be approved when a home loan has been approved by the bank for the payment of the buying price. Only 42% of all home loans are approved; the home loan needs to be approved before

it is considered a conveyance task. Once it is accepted as a conveyance task, fees for the property transportation (conveyance) can be completed.
Depending on the complexity of a task, 10% of tasks will be handled within 3 months, 20% within 4 months, 30% within 5 months, 30% within 6 months and the rest will be completed after 7 months.

After the tasks have been completed within three months the deed of the property is sent to the buyers' attorney from the deeds office, the deed is also sent to the bank from the deeds office, then only is the task complete.

The salaries mentioned have to be generated from the fees of the buyers. It however takes a period of time for the salaries to be generated from the fees to actually pay the salaries.

1.7 Preparation

In preparation for completing the project for DWR, the first crucial step will be to learn Java, as this is a platform that can be easily used by DWR and changes can be made to accommodate changes of the DWR business structure. Java is a capable platform. However if problems appear there are strong support systems to find solutions to the creation of the model.

1.8 Project Plan

Tasks
The task for this project is to be able to write a simulation model in the Java platform. This requires the understanding of Java and how a simulation model can be built. This includes applying core programming principles so that a full function model can be built for DWR.

The next task will be to investigate the business systems, to verify that data that has been given is valid and that the method that was used to gather the data is valid; this will ensure that the model will be as accurate as possible, the data gathering is to confirm the business structure of DWR, which will in turn reflect the structure of the Java model. DWR keeps records of every conveyance task accepted and not accepted; this is where the data will be gathered to which the growth of the company can be observed from the data. The finance personal will be able to use this data in the model. Ultimately the data is used to have a detailed description of the business structure of DWR regarding costing, in terms of costs towards employees and the DWR building, for instance how maintenance is completed and how the costs will be allocated towards DWR. This gives further insight into DWR business practises.

If trends are identified the model will not necessarily incorporate these trends, however DWR wants to be informed and aware of these trends. Yet DWR requires the absolute ideal scenarios. Yet if there is market growth, DWR personal will be able to identify the growth or non-growth and will be able to incorporate this into the model.

Next, costs and income will be identified that should be incorporated into the model. There are also expansion plans that will have to be taken into consideration. This gives indication that DWR will grow, therefore the model will be built to accommodate the growth of DWR. The business will grow as more conveyance tasks will be allocated to DWR as more of the property agency branches will open. This indicates that growth is inevitable for DWR and the model will have to adjust accordingly.

The next task will be to create the model for the business through algorithms, then to program the model into Java. After programming the model in Java, the model will be tested. This may be completed through DWR expectations allowing for a margin of error. The model may be validated against the database data, in terms of costs and revenues and other relevant business data. When testing has been complete, debugging will be performed on the model followed by the completion of the model. After debugging final verification and validation of the model will be performed.

Once the model has been debugged in Java the model will be implemented at DWR, debugging will also come in as a factor when implemented at DWR. DWR will be able to modify the program in Java if specific changes need to be adjusted.

Tools

Monte Carlo simulation will be used to analyse the conveyance tasks of each branch and the income generated from those tasks. This will forecast the tasks to be generated for the next 60 months. The inputs will be changed through a Java interface that will be built, thus different scenarios will be accommodated.

A simple method for optimising the resources will be part of the solution; that will apply to the solution for De Wet Reitz. The optimisation model will work on a basic premise that the work will be divided equally. This optimisation method will be allocated only to paralegals since they handle the actual tasks that generate income. Other resources are allocated in terms of the amount of paralegals, thus optimising the paralegals, will optimise the amount of resources allocated to them. There are strict rules of how the rest of the resources are allocated in terms of the paralegals.

Together with the simulation methods, a break-even analysis will be built into the model as it is one of the deliverables and is needed to determine whether variables inserted will result in a feasible output. Together with the break-even analysis the amount of capital needed before the break-even point is also an output.

Java will be a tool for creating the model; a user interface can be created for entering variables into the model as inputs creating a new scenario. This is to determine if a certain risk should be taken within the business, like appointing a new receptionist, paralegal or attorney, which will add additional cost. The final output will be a table of the allocated resources per month for a single run of the simulation model. This will give perspective in terms of where DWR is each month and give reasonable certainty if a new paralegal should be allocated or if a new branch should be opened. An example is given in figure 1.

Resources	Month				
Paralegals	1	2	3	……	60
Transaction Coordinators					
Managing Attorneys					
Messengers					
Admin Staff					
Finance					
Deeds Office Attorneys					
Receptionists					
NPV					
Annual Growth					

Break-Even Point	
Capital before Break-Even Point	

Figure 1 – Final output table.

1.9 Chapter Summary

In accordance with the DWR series, a model is designed and built for the financial simulation of DWR. This model is then built in Java. The simulation model will return the outputs that DWR require, these outputs are consist of a projection for the simulated 60 months, a break-even point before the project turns to operate at positive profit, the amount of capital needed before the break-even point is reached and the amount of resources needed per month based on the entered variables. This will be achieved through Monte Carlo simulation and Economic Analysis techniques.

Chapter 2

Literature Review

To reach all the deliverables of the project, a simple optimising method and a simulation model will be introduced into the Java platform. The method will optimise the amount of resources needed to manage the forecasted work load. The workload data (amount of tasks) of DWR will be analysed to identify any patterns or trends. This is to accurately estimate the amount of work that each branch is generating and the amount of income generated as each task has an accompanied buying price.

An adapted Monte Carlo simulation model will be used to analyse the data and simulate the future as the nature of the business is based on volume. As the series for the amount of tasks is given, this will be the ideal operating procedures. The income will increase if the workload increases, the workload increases once the home loan of the buyer has been approved also once the seller's finances have been approved.

The method for optimising the workload will work on a simple basis. If the resources cannot cover the minimum workload intended, the resource will simply not be added by the model. This in turn is a need as DWR needs to be able to assign resources so that the workload can be accommodated in the future.

The simulation model together with the optimising method will be implemented into the Java platform. As a cash projection is needed, the forecasting and optimisation will be able to project this cash flow for the total 60 month period. Each month at a stipulated time, there will be a number of resources (attorneys and paralegals) allocated to the business, this information can be retrieved from the Java model, and this information will assist in the decision making of allocation resources.

In accordance with the different scenarios that can be portrayed depending on the DWR needs; the Java model will also display other scenarios that will be created by the user. This will also provide much more insight. The model will simulate the amount of tasks each month for each branch and from the simulation results of the amount of tasks, the total income can also be generated, "forecasting" the coming amount of tasks for the future months.

Growth is a prominent factor in the real estate industry, the simulation model will incorporate growth as a variable.

However if management has certain goals within the DWR strategy; to have a specified amount of resources at a specific stage; the feasibility will change; this will also be accommodated within the model.

All factors will be brought into account with regards to the model, which will not only forecast but optimise resources within the future of DWR. As the scenarios will be able to change the feasibility of the forecast will change.

2.1 Simulation

Simulation models have been built on the Java platform before with prominent accuracy. Java is one of the best platforms to build a user-interface simulation with. Because of the simplicity of the on-screen interactions of the user, these event include mouse interactions and retrieving user input data (Fenton & Cherry & Hastings & Evans, 2001).

Java also includes parameters to be changed with ease in real time with the addition of plotting the results in more than one dimension, which enables the results to be graphically displayed. However if the algorithms are not optimised the computational speed may decrease significantly (Fenton & Cherry & Hastings & Evans, 2001). For the project however the algorithms will have to be written in basic Java code, if the structure of DWR changes it will be able to be adapted with minimal knowledge.

Eclpss (Ecological Component Library for Parallel Spatial Simulation) is a framework of Java for ecologist to create grid ecosystems on different scales. This framework creates methods and objects for the simulation of eco-systems. With this complexity and the ability to create object interfaces that resemble different elements in an eco-system, Java is a perfect example of creating simulation (Wenderholm, 2004).

In previous work, where complex electrical charges, in the brain, were simulated by using Java, with the addition of begin and end user interfaces (Fenton & Cherry & Hastings & Evans, 2001). This clarifies that complex systems may be simulated in Java. Financial simulation will be simple as the costs structures may be complex, but with an experienced and knowledgeable user the model can be utilised fully, in terms of performing simulations and restructuring the simulation model if needed.

The basic structure within the model

It is assumed that the current 15 branches may be extended up to 20 real estate branches. These branches run on the same series as the newly opening branches. The simulation will be implemented on each separate branch as each branch generates different amounts of tasks and thus different amounts of income. This is because of the different areas where real estate is sold. In the areas where property prices are higher less property is sold but the buying price on each real estate property is high. The complete opposite is also true in the areas where the property is at a lower purchase price. In these areas more property is sold but at a lower price, thus the income differs from branch to branch as each branch is located in a different area.

Through investigation it has been observed that in the month of April and December the amount of tasks are low. In April, this is due to the amount of public holidays and in December, it is holiday season. Although DWR requirements are to simulate the ideal conditions the models' structure is easily modified to accommodate this aspect.

The series of tasks per given branch opening is as follows:

{0, 0, 2, 5, 7, 10, 11, 12, 13, 15, 17, 19, 19, 20, 22, 25, 25, 28, 28, 30, 30, 30, 30, 30, 30, 32, 32, 33, 33, 35, 35, 35, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37}

This series illustrates the ideal amount of tasks for the 60 month period.

The simulation will determine the amount of tasks that will be generated each month spanning over all the branches and the amount of resources necessary to accommodate the demand. This is where the optimising method will come in to optimise the amount of work with the minimum amount of resources needed to complete the work each month.

The simulation in its essence is completed by the model; it will run the Monte Carlo simulation and bring forth the amount of tasks that are generated, thus the income for every month.

The basic structure is to clarify that the DWR model will be easily programmable into Java which will be an optimal platform for the simulation to be completed in because of the support services for Java and the simplicity of creating user interfaces. Also with little programming knowledge the simulation structure, if needed, will be able to be changed.

2.1.1 Monte Carlo Simulation

Monte Carlo simulations are regularly taught from textbooks. It is a common practice in programming. Monte Carlo simulation is used in computational mathematics, physics, chemistry and finance. (Liang, 2011: 134).

A Monte Carlo simulation will be built into the Java platform taking the different buying prices of each branch into account. It is assumed that the existing 15 branches also generate the same series as for the new opening branches.

The Monte Carlo simulation was developed during the Manhattan project in the 1940's. Monte Carlo simulation involves inserting a large number of inputs into a model while recording the outputs. Normally the inputs are analogous to the roll of dice or such, but with DWR the inputs are deterministic from the series (the amount of tasks), DWR will utilise the model for future use where the series or other variables can be changed to display different scenarios in business. The income is deterministic as there is a specific fee charged for each conveyance task. The costs are more dynamic, based on averages, thus Monte Carlo simulation is used.

The Monte Carlo simulation will run for the different branches from the various areas as the incomes from each area differs. Some of the inputs are the following:

- Transfer fee
- Deeds Office fee
- Tax
- Salaries
- Inflation

For all the tasks that are generated from a branch, the result will be an average income per branch, probabilities with regards to the inputs will be implemented as to give a realistic approximation of the output. Probabilities of the success rate of a task that has been approved and the probability of the time needed to complete a task.

As there are more inputs, these inputs differ from branch to branch and property to property. If different property values were to be mixed in the beginning, an inaccurate average will be seen. Thus the Monte Carlo simulation will be run for different branches as they are located in the different areas.

The information needed for the inputs will be an analysis through statistical methods to ensure that the actual inputs will be accurate. These statistical methods will be implemented within the Java platform as the data will update automatically from the DWR database.

There are also other factors to consider as the amount of tasks that are completed with each month as there is a probability that some tasks may take more time to complete than other tasks.

Market conditions such as the growth will also be incorporated within the model. Market growth for the year will be implemented into the model as an input; the percentage of growth can easily be obtained from ABSA bank. ABSA have forecasting of the real estate market growth which is in the public domain. This information can be used by DWR for the simulation.

These factors will lead to an accurate model for forecasting. Thus an optimising method will be implemented to optimise the amount of work generated (forecasted).

2.2 Optimising Method

The optimising method is used on the amount of paralegals needed for the amount of tasks that have been accepted. Simply, the method will look at the cost of a paralegal; it will then determine the amount of tasks needed to cover the basic cost of allocating the new paralegal. If the cost of appointing a new paralegal is R24000 and the basic cost of covering a task R3500 and assuming the paralegal can maintain 50 tasks each month the amount of tasks to cover the basic cost would be 6.857; thus 7 tasks. If the amount of tasks for a month is 506, note that 10 paralegal are allocated ($500 \div 50 = 10$), a paralegal will not be appointed. If the amount of tasks for a month is 507 or above, a paralegal will be allocated as the basic cost will be covered, this will minimise the amount of cost in terms of paralegals per task and influences other employee allocation to be minimised.

2.3 Economic Analysis

Economic Analysis has been completed on simulation models before, however not by using Java but using other simulation models. HYSYS (Hypothetical System) simulation is used to simulate biodiesel production after the simulation was completed economic analysis has been completed on the simulation data and has given significant results (Lee & Posarac & Ellis, 2011).

The results of the HYSYS simulation model has shown which process represented the lowest capital invest and which processes were more feasible. Showing which processes were provided low manufacturing costs and resulting in a higher net present value. At the end of the simulation a sensitivity analysis was completed using indicating parameters to clarify which factors affect the net present value of the model the most (Lee & Posarac & Ellis, 2011).

Although the HYSYS simulation costs were determined by Aspen In-Plant Cost Estimator, DWR costs will be estimated by the user. The user will have to be a manager or a finance employee who will have performed data analysis to have cost estimates and has significant knowledge on the income and cost structures of DWR. For the income estimates data will have to be consolidated for the average price per branch. The costs and income are different from manufacturing processes as DWR performs a service (Lee & Posarac & Ellis, 2011).

In addition, simulation integration and economic analysis has also been performed, this process was a GTL (Gas–to-Liquid). The simulation has been created on ASPEN Plus simulation software. Afterwards economic analysis was performed, where the return on investment was a result and the break-even point was calculated from the simulation (Bao & El-Halwagi & Elbashir, 2010).

In the case of the GTL process, where the Fischer & Tropsch is used, the costs were still under uncertainty, this is to be expected in any process. To bound this uncertainty to an extent; assumptions were made to focus the simulation more and to give a rough yet accurate estimate for the costs of the process. Yet from the simulation results, economic analysis was still performed to give reasonably accurate end results (Bao & El-Halwagi & Elbashir,2010).

When the Monte Carlo Simulation has been completed for the branches and the optimising method has optimised the amount of resources allocated to the workload; the income will be added together with the cost to produce the profit. This profit will then be part of the cash flow projection. The monthly cash flows will be displayed on a cash flow chart generated by the Java model. This chart will be implemented in Java together with the Monte Carlo simulation and the optimising method.

Together with the projection chart, various economic analysis techniques will be completed on the cash flow chart.

Different analyses will be performed on the cash flow projection as to assess the worth and feasibility of the project. It will also assist in management decisions that need to be made.

The different analyses that will be performed:

- Present Worth Analysis
- Annual Cash Flow Analysis
- Break-even Analysis

From the research completed, although based on pure manufacturing industries and other simulation modelling software it is possible to complete economic analysis on simulation results and to be accurate within a certain margin. Complex systems can be simulated in Java, almost any form of simulation can be completed by using Java.

It is also noteworthy that DWR does perform a service, thus the structure of the business may differ from product simulations but accurate simulation is still possible within the Java platform. Thus, a combination will be created within Java. XBase was selected at first as a platform for the model but since there is not much support for XBase, Java was selected not only for its compatibility with almost all operating system, but mainly for the fact that only basic programming skills are necessary to edit the structure of the model (source code).

2.3.1 Present Worth Analysis (Net Present Value)

Present worth analysis is used to clarify what the business is worth at present, considering future circumstances. Once the present worth is known a decision can be made to continue the project and what value will bring to the business. The present value can be compared to the following year(s) cash flow to clarify if the projects worth will grow; stall or decline and from there a decision can be made.

As is the case with DWR, one of the outcomes is to build a model so that DWR can estimate if this project or perhaps future projects can be implemented in DWR to add value to the company. The Present worth analysis will only return a value and according to that value DWR will be able to make a more informed decision.

With the present worth analysis DWR management can make an informed decision on the lifespan of this project. Within the model the present worth will be calculated as to indicate the worth of the project, this leads to the fact that inflation as well as the interest rate will also be an input into the Monte Carlo simulation. The formulation of the present worth analysis will also be implemented into Java as DWR's systems are implemented with the Java platform.

Present worth analysis is applicable to almost any economic problem. DWR can use this model as the basis for other projects, where applicable. Present worth adds value particularly when the current worth of future costs and benefits of the project are needed. This extra information will assist in the decision making process of DWR. This will help determine if the project has future value for the company.

The present worth is computed in the following manner:

$$\text{PW (Present Worth)} = F\,(P \mid F, i, n)$$

F: The future value of each month, cash flow of each month, forecasted from the Monte Carlo simulation.

i: The interest rate, this can be obtained from the banks, the forecasting of the interest rate can be obtained from the banks.

n: The time period that the project spans over, which is 60 months.

The equation and adaptions of this equation will be implemented in the Java platform.

2.3.2 Break-even Analysis

The break-even analysis is completed to see how many tasks should be accepted and completed to cover all costs. This is in line with the objective of the model to determine the break-even point of DWR of the project. The analysis is also used to estimate the amount of capital needed to allow the project to start-up, as more investors are interested in this project.

There are different methods of estimating the break-even point. The equation method is one of two methods to be used in the model. The break-even point can be expressed in the following equation:

Profits = Sales – (Variable Expenses + Fixed Expenses)

This indicates to the amount of sales needed for the break-even point to be achieved, as follows:

Sales = Variable Expenses + Fixed Expenses + Profit

At the break-even point the profits are zero. Therefore the amount of sales (tasks) required to achieve the break-even point can be computed through the following example:

Sales = R3500Q

Variable expenses = R1700Q

Fixed expenses = R20 000

Profit = R0

Where Q is the amount of tasks needed to be completed to achieve the break-even point. The R3500 at sales is the rate for each task and the R1700 is the cost per task.

Thus the amount of tasks needed to break-even will be computed as follows:

$3500Q = 1700Q + 20000$

$Q = 11.11$ tasks to be completed for all the costs to be covered.

From here the capital can be calculated to finance the project.

2.3.3 Projection Method

As the equation method may work, but does not necessarily account for inflation and other factors that have a significant role in the sales and cost domains, it is also difficult to identify at what time the break-even point is reached. The projection method can be used and will be much more accurate in determining the break-even point for the whole project.

The model will identify all the points on the projection where losses were made and for the whole project; a point will be reached where there are no losses made and perhaps no income gained. At this point the model will calculate the amount of capital needed to finance the project. The time parameter where the break-even point is reached can also be identified.

The model will calculate the sum of the amount of losses before the break-even point is reached. The model will calculate the amount of capital needed for the start-up of the project or what the start-up should be. Through inputs this will be calculated through the present worth method, accounting for the inflation rate (Input) and interest rate (Input).

This method will give a much more accurate, “real world feel” to the projection and the amount of capital needed for the project, which is in line with the project objective.

2.4 Chapter Summary

Cleary shown, Java is a powerful simulation tool, incorporating user-interfaces for different scenarios. Complex simulation models are built in Java, with success. This gives reason that the DWR model can be built in a similar manner as the costing structures are complex, the difference is that the simulation model of DWR will be constructed in simple Java coding as changes may be needed for the future.

Chapter 3 - Research Methodology

3.1 Data Gathering

Data gathering has been conducted as to clarify the structure of costs and income. The data gathering process forced DWR, to clarify how their employees are allocated, thus clarifying how their costs are determined. Their cost per individual employee is determined by the finance personal and the costs of each employee is determined by their cost in terms of office space, used or needed, for each type of employee. The costs ranges from IT use, stationary use, printing cost, human resource costs, telephone usage, travelling and insurance. Other monthly costs includes the storage of the files (tasks that are completed), postage, utilities, advertising, entertainment, finance charges, maintenance costs, membership and subscription costs. The monthly cost only applies to a certain extent to the project. Thus a percentage or "split" value is assigned as to more accurately determine the costs that are generated from the project.

Throughout the data gathering process the costing structure was identified and the model was built on the cost assumption made by DWR management. DWR is an developing business and allocates staff as needed yet for the future change will be needed. Thus data gathering forced DWR to gather insight as to how their costing structures work and may work in the future, for instance how certain their personal will be allocated. If this changes, the model will have to be changed.

A crucial assumption of this model is that the model will simulate the ideal scenario. Through data gathering, assumptions were made on how the cost structures are determined, such as the amount of resources that are needed. For instance for every 10 branches there is one transaction coordinator needed, this assumption and similar assumptions have been discovered through data gathering. Also note that these assumptions will be able to be changed within the model.

3.2 The Java Model

The series of DWR {0, 0, 2, 5, 7, 10, 11, 12, 13, 15, 17, 19, 19, 20, 22, 25, 25, 28, 28, 30, 30, 30, 30, 30, 30, 32, 32, 33, 33, 35, 35, 35, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37} has been inserted into the Java platform as many different arrays. The first and second month has zero tasks that are generated, the third month has two tasks that are generated; thus an array is created with a size of two. This is the maximum of tasks that can be generated per month, as no more can be generated for the month. Within this created array are random generated probabilities, when the probability is less than 42% (Home loan acceptance figure) the task is accepted as a task that has been successfully generated by the branch, when the home loan is accepted. This variable of home loan acceptance can also be changed in the model interface. Furthermore, the same principle applies to subsequent months. For month four no more than five tasks can be generated, thus five probabilities are generated accepted and rejected. The model will generate the amount of tasks in this manner as it is the requirements from DWR, this is the assumption that DWR wants to perform its simulation on. This configuration is known a triangle array. This part of the model is created as static arrays. Within the verification phase it will be determined whether the maximum amount of tasks should be allowed to change.

The basis of the task generation is based on a triangle configuration where the amount of arrays are known but the size of those individual arrays are different from one another. The triangle array, Probabilities [j], is created from a single name and created separately, shown below (Note Java indexing starts with 0 not with 1).

$$BranchProbabilities\ [0] = size\ [1] = 0$$

(Due to the opening of the branch, also assumed no business actions will be performed)

$$BranchProbabilities\ \ [1] = size\ [1] = 0$$

(Due to the opening of the branch, also assumed no business actions will be performed)

$$BranchProbabilities\ [2] = size\ [2]$$

$$BranchProbabilities\ [3] = size\ [5]$$

$$BranchProbabilities\ [4] = size\ [7]$$

$$\dots$$

$$BranchProbabilities\ [60] = size\ [37]$$

On the left of the equation is the matrix name and its identifier, on the right of the equation is the size of the matrix thus for Probabilities [2], the size will be a 2x1 matrix.
For each index of these matrixes a random probability is produced, it is then checked against the acceptance level of the home loans (42%).

The amount of accepted (generated) tasks is stored in a different array as a total for the tasks accepted per month, per branch. This method works on a normal distribution, as it is also the ideal requirement from DWR and is applied to every branch. The amount of branches are inserted as an input in the interface which ranges from 15 to 20 branches, this range is a requirement of DWR. An input for the amount of iterations is also created for accuracy. The iterations cause the model to simulate a branch for the amount of iterations. This is stored in an array where the average is calculated per branch.

A growth rate is also added for the year; thus after a year a branch that is added after 12 month will produce more tasks than the previous branches. This growth rate is a variable that can be changed in the user interface.

The model will look at the amount of tasks per month generated and create an array with the same size as the amount of tasks for the month. The amount of tasks generated per month is an array that is generated from addition throughout all the branches per month. The model will then create random probabilities within this array. If these probabilities are less than 0.1, 0.1 – 0.3, 0.3 – 0.6, 0.6 -0.9 and larger than 0.9 for each task, the task will be added to an array that keeps track of the completed task for a specified month as it is needed to keep track of the total tasks per month that are completed. It is added into a specific index relative to the month of where the task was generated. The 0.1 probability will allocate it three months ahead of the current index as it is then completed. So for example, if the index of beginning month is [3], the model will allocate it to an array with index [6]. If the probability is between 0.1 and 0.3, the task will be allocated four indexes ahead; from index [3] to index [7]. When the probability is between 0.3 and 0.6 the task will be added to an index 5 indexes ahead; from index [3] for instance to index [8]. When the probability is between 0.6 and 0.9, the task will be added to an index 6 indexes ahead; from index [3] for instance to index [9]. When the probability is larger than 0.9, it will be allocated seven indexes ahead, from index [3] for instance to index [10]. The model will complete this method for every individual new task per month. The completed task array is allocated within the same time frame as the created tasks per month array, this allows for simple subtraction. The addition method of the total tasks per month is simple. For an index the tasks per month is added to the previous index of total tasks per month that are already registered, from there the amount of tasks that are completed will then be subtracted to give the total tasks that will have to be accommodated. It is completed in this manner as the total amount of tasks do pass over to the next month, the tasks can only be removed with a different array with the same time frame. Figure 2 will illustrate the method of subtraction and addition.

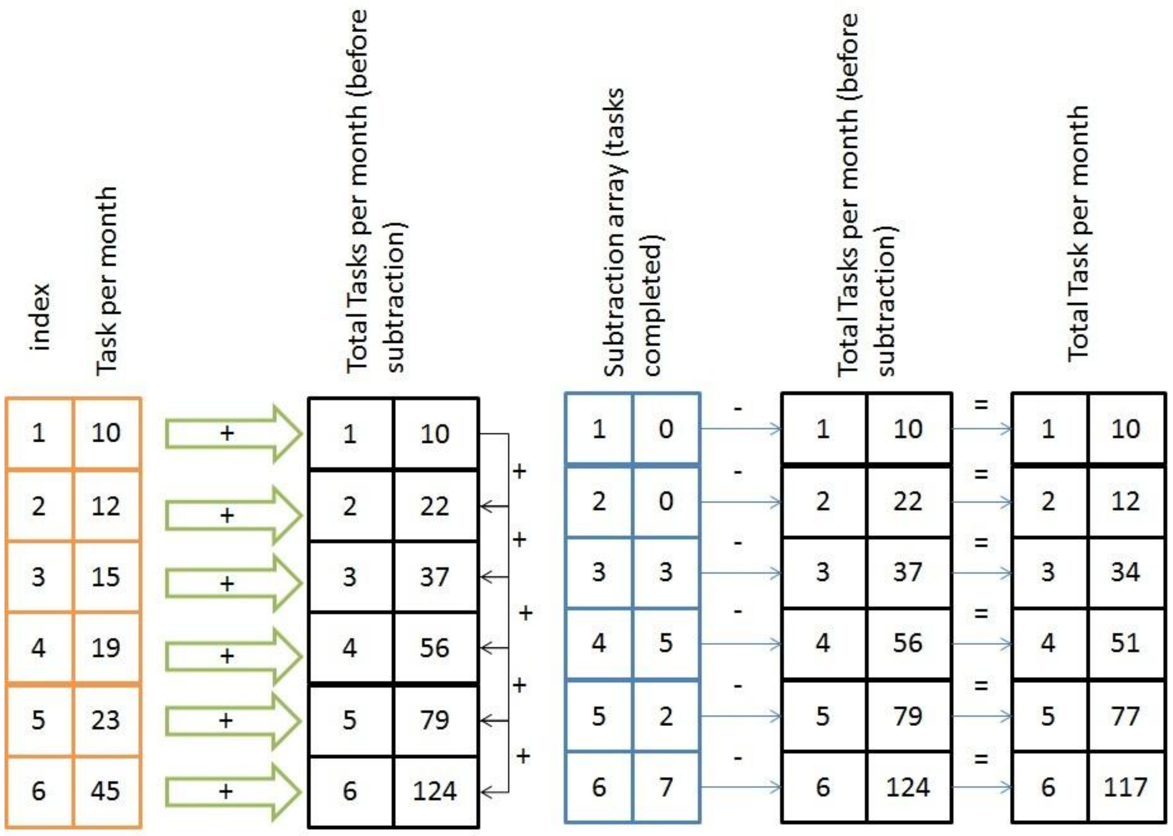


Figure 2 - Addition and subtraction method.

The average amount of tasks generated per branch will be the basis of the basic fee charge, which is R3500 per conveyance task that is accepted after home loan approval; together with this fee, DWR charges transport fees depending on the purchase price of the property. To accommodate this fee charge aspect, every branch is given a name depending on where the branches are situated. Branch name examples will be Midrand, Boksburg and Kempton Park etc. DWR will have the information of the average purchase price in the different areas; this will be the value for the variable that accommodates the transport fee charge and is linked to the specific branch.

There are certain limits to the resources. Resources include the paralegals, messengers, finance personal, operations managers, administration staff, deeds office attorneys, managing attorneys, transaction coordinators and receptionists. The limits of these resources are based on certain aspects of DWR. The limits of paralegals are based on the amount of tasks generated by the branches, thus one paralegal can only accommodate 50 tasks at any given time. For administration staff every 75 tasks that are generated, 1 administer will be allocated. The amount of deeds office attorneys is based on every 200 tasks that are completed (signed by the buyer and seller). A messenger is allocated based on the amount of paralegals; one messenger will be allocated per 10 paralegals. The same will also be true for financial personal and operations managers. Other resources are based on the amount of branches that are open so for every three branches that are open a transaction coordinator and a managing attorney is allocated. The amount of receptionists is based on the total amount of employees with in DWR. Through the user interface it will be possible to adjust the limits.

Other costs are also present during every month costs such as utilities, advertising, maintenance, membership and subscription costs, postage and storage costs. These costs will be total costs of DWR. This fact leads to a necessary separation of what costs are incurred by the project, this ratio or "split" is known by the user, this ratio may also increase over a year or so, this ratio will also be includes in the user interface. More cost variables include the salaries of each personal member; this is also based on an average value.

Within the model there is an income projection array, a cost array and profit array. The income array is for determining the amount income per month. An index is determined by the basic fee multiplied by the amount of tasks. Also the average price of a property for every branch is taken as an input; DWR will then take a percentage of that fee; thus the average price is multiplied by the percentage and the amount of tasks from the specific branch. This is completed by the model for every branch.

The cost array is determined by every cost variable per month. The model will identify the amount of resources per month and allocate the costs accordingly for each month; this will generate a cost projection for each month over the 60 month period. The cost per resource is an input from the user interface.

With the income and cost projection generated the profit projection is created with simple subtraction, where the cost projection is subtracted from the income projection for each month and thus creating the profit projection for each month over the 60 month projection.

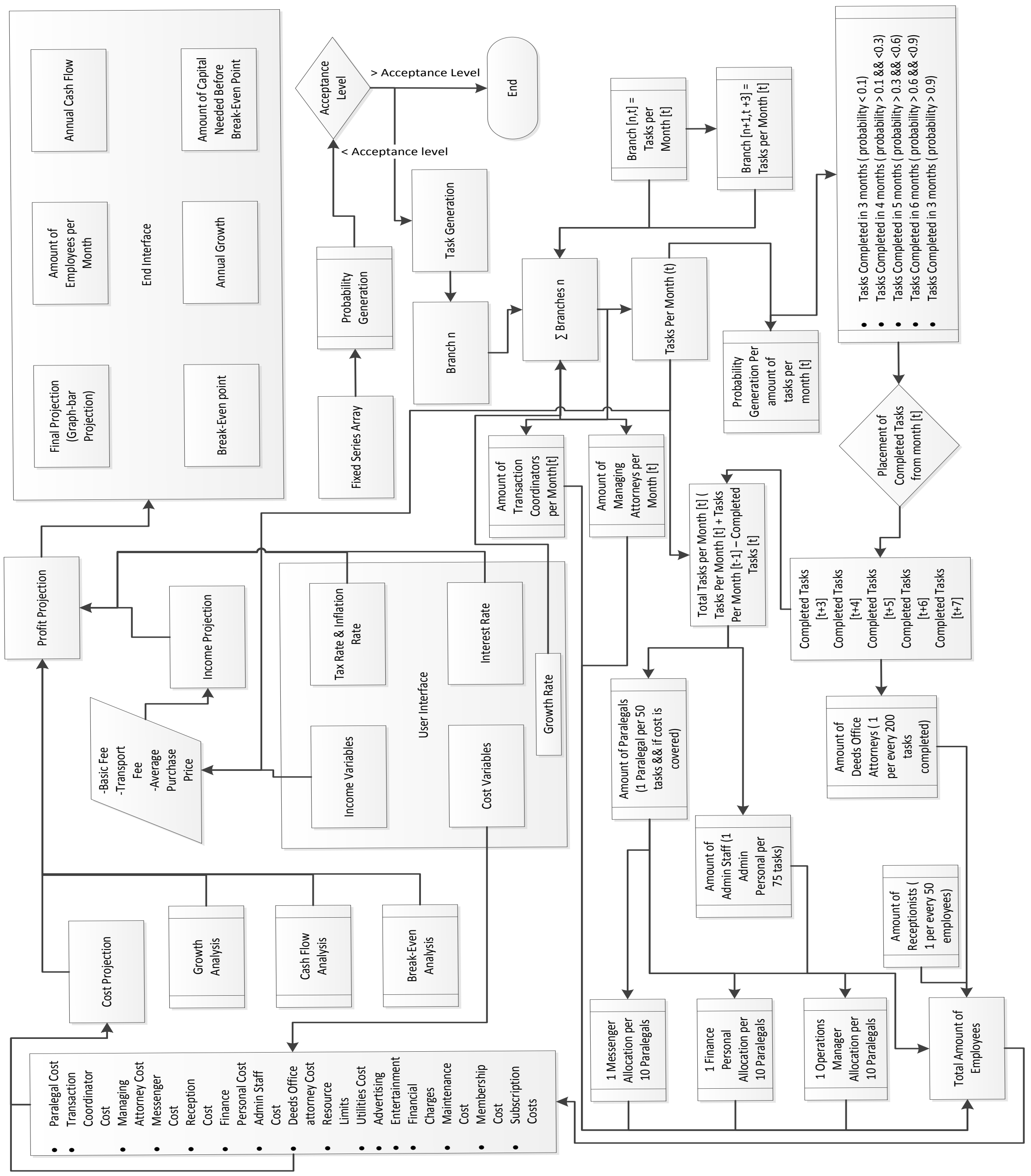
From the profit projection various economic analyses can be completed before and after tax; where the tax rate and interest rate is an user interface input. The economic analysis that is generated from the model is the present worth of the project before and after tax. The break-even point is also determined from this projection as it can be seen at an instance that the project profit is positive. It is expected that the simulation will give negative values for the first few months, indicating that losses are made, the summation of the losses will amount to the investment capital needed. This will give a invest amount before a true profit is turned. Generated out of the model is the annual cash flow and annual worth to give an indication of the performance of the project; this also indicates the projected worth growth of the project.

The end result of the model will be a final cash flow projection. Together with this cash flow projection will be the optimal amount resources needed for the values that were inserted in the user interface. The model produces simulation statistics based the different inputs from the user interface. This allows the user to generate different scenarios based on the change of the income, inflation, tax rate, cost, ratio, percentage, growth and limit variables. This will allow to compare scenarios for the user.

The user of the model should be thoroughly involved in the finance of DWR to ensure that the results generated are accurate. The model has also been written in very basic Java code, as the structure changes over time, so can the model. If the model needs to be changed it can be changed easily, a user needs only very basic Java code experience to change the model. Instructions are also imbedded with the model source code for simple changes.

3.3 Functional Model

The functional model gives a clearer description of how the model operates and what the structure looks, seen in Figure 3.



End Interface
Annual Cash Flow
Amount of Capital Needed Before Break-Even Point
Amount of Employees per Month
Annual Growth
Final Projection (Graph-bar Projection)
Break-Even point
Acceptance Level
> Acceptance Level
< Acceptance level
End
Task Generation
Probability Generation
Fixed Series Array
Branch n
∑ Branches n
Branch [n,t] = Tasks per Month [t]
Branch [n+1,t +3] = Tasks per Month [t]
Tasks Per Month (t)
Probability Generation Per amount of tasks per month [t]
Tasks Completed in 3 months (probability < 0.1)
Tasks Completed in 4 months (probability > 0.1 && <0.3)
Tasks Completed in 5 months (probability > 0.3 && <0.6)
Tasks Completed in 6 months (probability > 0.6 && <0.9)
Tasks Completed in 3 months (probability > 0.9)
Placement of Completed Tasks from month [t]
Amount of Transaction Coordinators per Month[t]
Amount of Managing Attorneys per Month [t]
Total Tasks per Month [t] (Tasks Per Month [t] + Tasks Per Month [t-1] – Completed Tasks [t]
Completed Tasks [t+3]
Completed Tasks [t+4]
Completed Tasks [t+5]
Completed Tasks [t+6]
Completed Tasks [t+7]
Profit Projection
Income Projection
User Interface
Tax Rate & Inflation Rate
Interest Rate
Growth Rate
Income Variables
Cost Variables
-Basic Fee -Transport Fee -Average Purchase Price
Amount of Paralegals (1 Paralegal per 50 tasks && if cost is covered)
Amount of Deeds Office Attorneys (1 per every 200 tasks completed)
Amount of Admin Staff (1 Admin Personal per 75 tasks)
Amount of Receptionists (1 per every 50 employees)
Cost Projection
Growth Analysis
Cash Flow Analysis
Break-Even Analysis
1 Messenger Allocation per 10 Paralegals
1 Finance Personal Allocation per 10 Paralegals
1 Operations Manager Allocation per 10 Paralegals
Total Amount of Employees
Paralegal Cost
Transaction Coordinator Cost
Managing Attorney Cost
Messenger Cost
Reception Cost
Finance Personal Cost
Admin Staff Cost
Deeds Office attorney Cost
Resource Limits
Utilities Cost
Advertising
Entertainment
Financial Charges
Maintenance Cost
Membership Cost
Subscription Costs

Figure 3 – Functional Diagram.

Figure 3 is a clearer description of the model that has been built in Java. The model starts at the fixed series, were an array is created that is identical to the DWR series. Here a triangle configuration of the months tasks are created. Next, at probability generation random probabilities are generated for each triangle array index. These probabilities are then checked against the acceptance level, once a task is accepted it is stored in another triangle array, with the task accepted value of 1. These arrays are then summed and is seen as conveyance tasks generated by a branch. The amount of branches are saved in a matrix where the amount of branches are the amount of columns and the rows is the 60 month period projection of the tasks generated for each month. Seen below is the algorithm for a branch that has generated tasks, where n is the triangle array number and t is the index of array n:

$$If\ BranchProbabilities\ [n,t] > home\ loan\ acceptance\ level$$

$$Task\ Accepted\ [n.t] = 1$$

The summation of the tasks accepted of an array produces the tasks generated by the branch n and the month, $t,$ of that branch:

$$branch\ [n,t] = \sum Task\ Accepted[n,t]$$

The total of the tasks per month is generated by addition through an algorithm, where the branch is identified and added to a matrix that stores the tasks per month generated by all the branches over the 60 month period. Algorithm shown below:

$$Tasks\ per\ Month\ [t] = branch[n,t]$$

For the first branch and for the other branches:

$$Tasks\ per\ Month\ [t] = branch[n+1,t+3] + Tasks\ per\ Month[t]$$

From the amount of branches, the amount of transaction coordinators are determined and the amount of managing attorneys per month in the final projection.

The transaction coordinators and managing attorneys are allocated on the same basis, which is the amount of branches that are open at a time. From the tasks per month index the amount of completed tasks are calculated. In each month a certain amount of tasks are generated. A dynamic matrix is generated from the amount of tasks per month, probabilities are then generated for each task. When the probabilities are between, over or above the limits they will be allocated towards a corresponding index in the tasks completed matrix which is in sync with all other projections. Seen below is the allocation algorithm:

$Probabilities\ [j] = Math.random()$

$If\ Probabilities\ [j] < 0.1$

$Completed\ Tasks[t+3] = Completed\ Tasks[t+3] + 1$

3 Month Completion Time

$If\ Probabilities\ [j] > 0.1\ \&\ Probabilities\ [j] < 0.3$

$Completed\ Tasks[t+4] = Completed\ Tasks[t+4] + 1$

4 Month Completion Time

$If\ Probabilities\ [j] > 0.3\ \&\ Probabilities\ [j] < 0.6$

$Completed\ Tasks[t+5] = Completed\ Tasks[t+5] + 1$

5 Month Completion Time

$If\ Probabilities\ [j] > 0.6\ \&\ Probabilities\ [j] < 0.9$

$Completed\ Tasks[t+6] = Completed\ Tasks[t+6] + 1$

6 Month Completion Time

$If\ Probabilities\ [j] > 0.9$

$Completed\ Tasks[t+7] = Completed\ Tasks[t+7] + 1$

7 Month Completion Time

From here the total tasks per month for DWR is determined and the amount of deeds office attorneys. The total tasks per month matrix is the core for DWR, this is where the amount paralegals and admin staff is determined. The amount of finance personal, messengers and operations managers are determined from the amount of paralegals. The capacity of each finance personal, messenger and operations manager can separately be adjusted in the model.

The total personal is then calculated with the relevant cost per index, this will give the total costs per month, resulting in the cost projection.

The income projection is generated from the Tasks per month, this is the amount of new tasks per month, where the basic fee and conveyance fee is latched to the task and synced with the appropriate branch for the conveyance task fee. This will result in the income projection, where the income is given per month.

The profit projection is then calculated, taking interest rate, inflation and the tax rate into account. This will result in the final net profit projection. As a result, the model will calculate the break-even point, annual growth, amount of capital needed before the break-even point and annual cash flow. Along with in the end interface is the table shown in figure 1 showing the amount of resources needed each month.

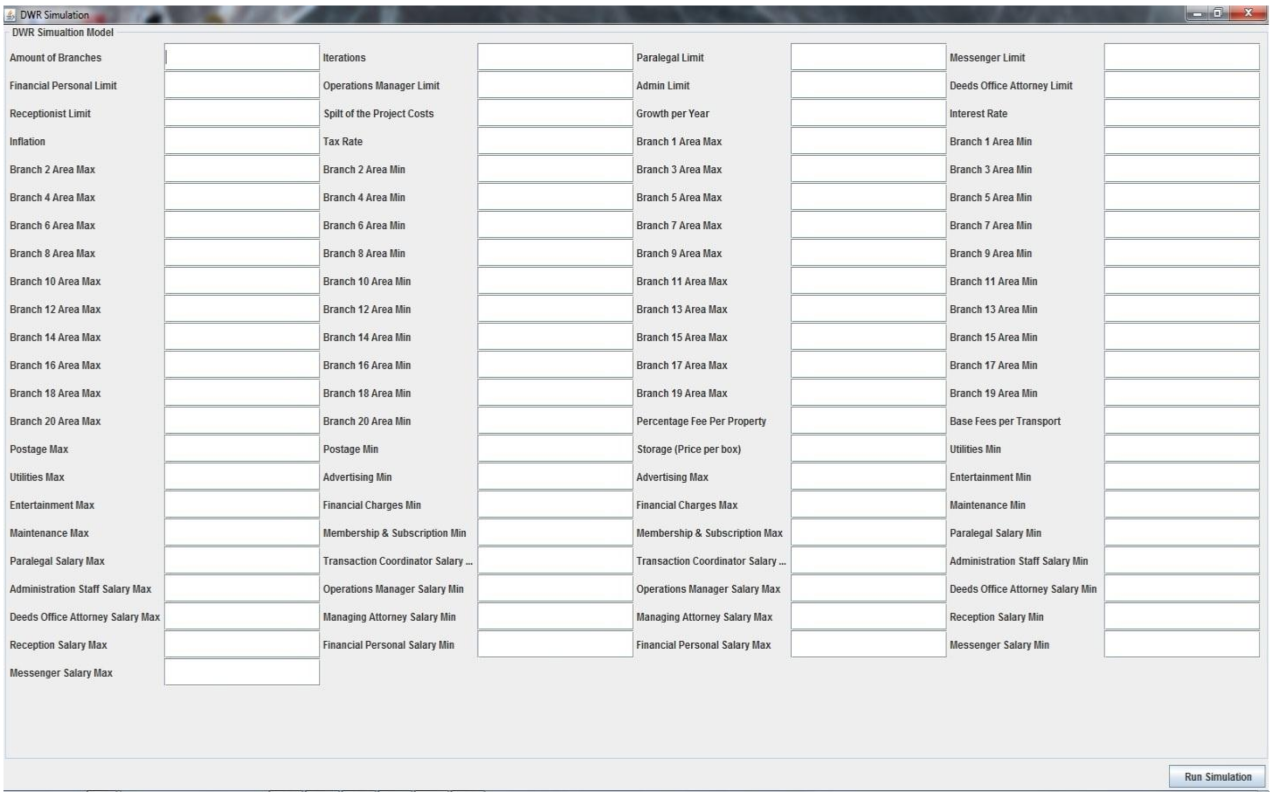
3.4 Chapter Summary

A financial simulation model has been built in basic Java code. If the model structure is incorrect it is simple and efficient to change the structure within the model. The simulation model will take all variables except depreciation into account of DWR and simulate the financial scenarios for DWR.

Chapter 4 - Results

4.1 User Interface

The user interface will be a simple grid layout. The user performing the simulation will have to know the financial status and costs of DWR, the user will have to be knowledgeable. The grid layout will ensure that the user will be able to simply and easily insert the different variable values without confusion. After the values have been inserted, a run button is visible to perform the simulation. Different simulations will be able to be performed based on the different limits, cost variables, interest rates and growth. A design of the user interface is seen in figure 4 below.



DWR Simualtion Model

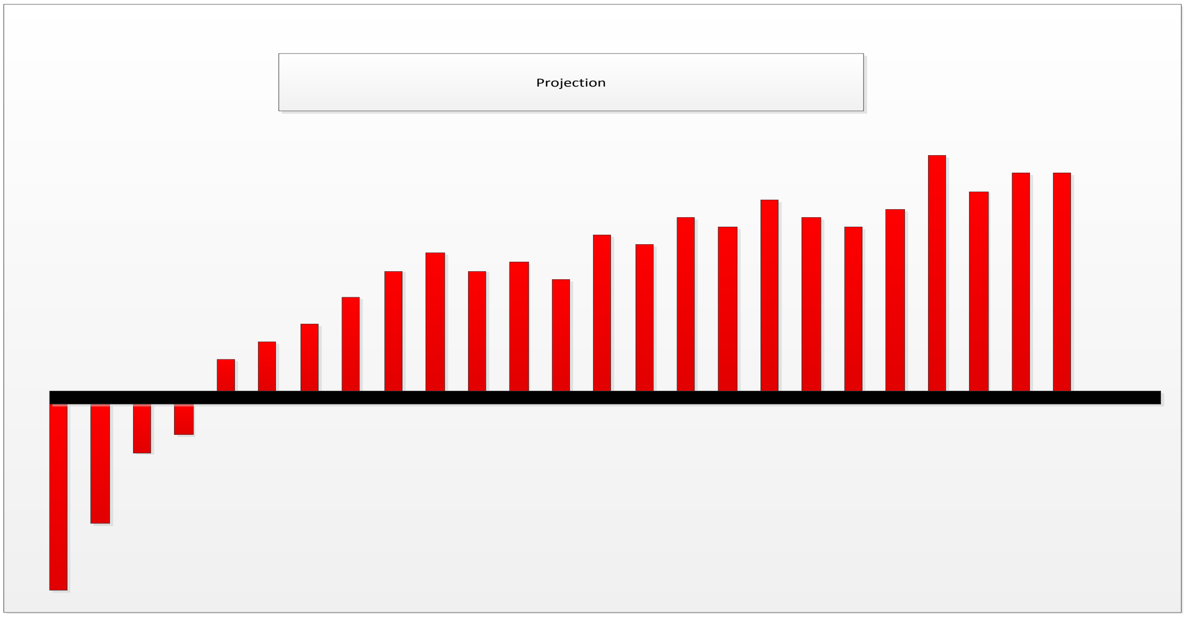
Amount of Branches	Iterations	Paralegal Limit	Messenger Limit
Financial Personal Limit	Operations Manager Limit	Admin Limit	Deeds Office Attorney Limit
Receptionist Limit	Spilt of the Project Costs	Growth per Year	Interest Rate
Inflation	Tax Rate	Branch 1 Area Max	Branch 1 Area Min
Branch 2 Area Max	Branch 2 Area Min	Branch 3 Area Max	Branch 3 Area Min
Branch 4 Area Max	Branch 4 Area Min	Branch 5 Area Max	Branch 5 Area Min
Branch 6 Area Max	Branch 6 Area Min	Branch 7 Area Max	Branch 7 Area Min
Branch 8 Area Max	Branch 8 Area Min	Branch 9 Area Max	Branch 9 Area Min
Branch 10 Area Max	Branch 10 Area Min	Branch 11 Area Max	Branch 11 Area Min
Branch 12 Area Max	Branch 12 Area Min	Branch 13 Area Max	Branch 13 Area Min
Branch 14 Area Max	Branch 14 Area Min	Branch 15 Area Max	Branch 15 Area Min
Branch 16 Area Max	Branch 16 Area Min	Branch 17 Area Max	Branch 17 Area Min
Branch 18 Area Max	Branch 18 Area Min	Branch 19 Area Max	Branch 19 Area Min
Branch 20 Area Max	Branch 20 Area Min	Percentage Fee Per Property	Base Fees per Transport
Postage Max	Postage Min	Storage (Price per box)	Utilities Min
Utilities Max	Advertising Min	Advertising Max	Entertainment Min
Entertainment Max	Financial Charges Min	Financial Charges Max	Maintenance Min
Maintenance Max	Membership & Subscription Min	Membership & Subscription Max	Paralegal Salary Min
Paralegal Salary Max	Transaction Coordinator Salary ...	Transaction Coordinator Salary ...	Administration Staff Salary Min
Administration Staff Salary Max	Operations Manager Salary Min	Operations Manager Salary Max	Deeds Office Attorney Salary Min
Deeds Office Attorney Salary Max	Managing Attorney Salary Min	Managing Attorney Salary Max	Reception Salary Min
Reception Salary Max	Financial Personal Salary Min	Financial Personal Salary Max	Messenger Salary Min
Messenger Salary Max			

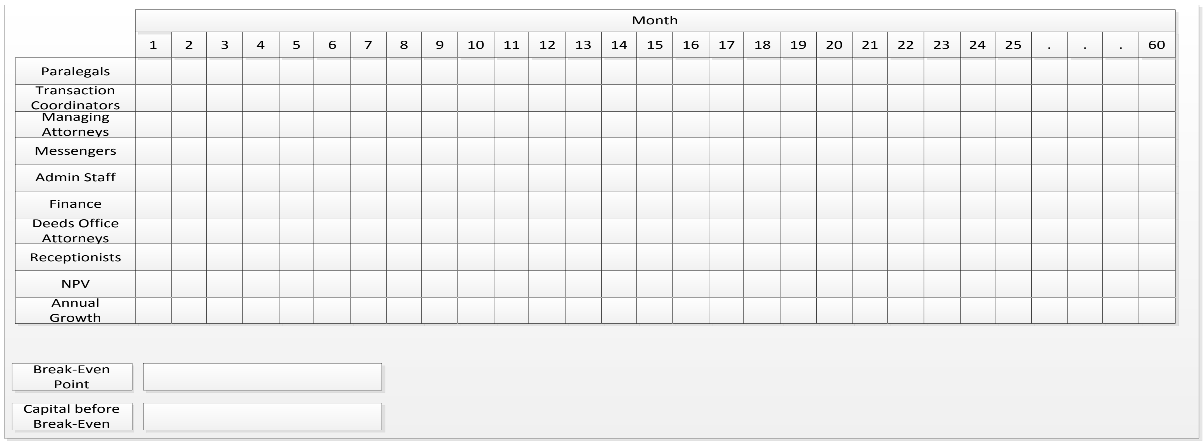
Figure 4 – User-Interface.

4.2 End Interface

The end interface will display the end result of the simulation, which is the project for a 60 month period. The interface displays the amount of employees every month, the annual growth, capital needed before the breaking even point, when the break-even point occurs and the present value of the project every month. A design of the end interface is seen in figure 5.

The graphs will be located in the C Drive directory where 3 graphs can be seen. The three graphs show the amount of profit per month simulated, the total amount of tasks and amount incoming tasks per month. The table is given as an output, from the model, the net present value of the project is given, the amount of capital needed before an actual profit is made. Also given and seen is the time at which the break-even point occurs.





	Month																												
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	.	.	.	60
Paralegals																													
Transaction Coordinators																													
Managing Attorneys																													
Messengers																													
Admin Staff																													
Finance																													
Deeds Office Attorneys																													
Receptionists																													
NPV																													
Annual Growth																													

Break-Even Point	
Capital before Break-Even	

Figure 5 End - Interface.

4.3 Verification

Sensitivity analysis has been completed for the model. The model was evaluated on the basis of the profit and the value of the project as this is the most important for DWR. So influences to these results are significant. For the verification, the growth of the project per year was changed while the other variables remained constant. If needed by DWR the maximum tasks per month can be changed. This will not affect the model as only numbers will change.

It is important to note that many variables were kept constant and the values were placed under assumptions. These assumptions include the following:

- Iterations the model runs for 1000 iterations.
- The amount of branches to be opened at the end of the 60 month period is 15.
- The workload for a paralegal is 50 conveyance tasks.
- Financial personal can only apply their resources to 10 paralegals at any time, the same is true for the operations manager and messengers.
- Admin personal can take a workload of 75 conveyance tasks.
- Deeds Office Attorneys can take a workload of 200 conveyance tasks.
- A receptionist can take a workload of 75 people.
- The costs that apply to the project in total are equal 75%.
- Growth is set at 5% as standard.
- Interest Rate is set at 8.5%.
- Inflation is set at 5%.
- The taxation on the profit is 32%.
- The amount of the fee taken according to the property value is 30% per property.
- The base fee is R3500.
- All branch values were set between R800 000 and R320 000.
- Utility costs were set at R8000 to R6000 per month.
- Advertising costs were set at R7000 to R3200 per month.
- Entertainment costs were set at R2000 to R500 per month.
- Financial chargers were set at R6000 to R5000 per month.
- Maintenance costs were set at R2500 to R500 per month.
- Membership and subscription costs were set at R25000 to R20000 per month.

- Postage costs were set at R9000 to R 5000 per month.
- The storage costs were R3.1 per box with the assumption that 18 files can be stored in a box.
- Paralegal salary was set at R25 000 to R20 000.
- Transaction Coordinator salary was set at R24 000 to R20 000.
- Administration Staff salary was set at R12 000 to R10 000.
- Operations Manager salary was set at R26 000 to R21 000.
- Deeds Office Attorney salary was set at R28 000 to R24 000.
- Managing Attorney salary was set at R45 000 to R30 000.
- Receptionist salary was set at R10 000 to R8 000.
- Financial salary was set at R20 000 to R15 000.
- Messenger salary was set at R12 000 to R10 000.

These assumptions were used to create the base, expected, model output. From here the growth, interest rate and the inflation rate have been adjusted to verify the model.

The growth was originally at 5%, the adjusted growth was 8% and 2%. Graphs have been created to clarify the difference. The adjusted growth revealed what was expected, for the 8% growth more profit had been eminent and for the 2% less profit was portrayed. The same results had been shown from value point of view where the maximum value had been reached much earlier for the 8% growth and the max value had been reached much slower for the 2% growth. Figure 6 and Figure 7 show the difference for the 8% growth. Figure 8 and Figure 9 shows the difference for the 2% growth.

The interest rate was kept at 8.5% and was adjusted to 10% and 7%. The interest only affects the value of the project not the profit of the project. The results were expected and clarified that if the interest rate increases the value of the project will reach the maximum value more rapidly seen in Figure 10 and Figure 11.

The inflation rate was kept at 5% and was adjusted to 6.5% and 3.5%. The inflation only affects the value of the project not the profit created by the project. The results, like the interest was expected, if the inflation rate increases the value rate to the maximum value decreases and when the inflation rate decreases the maximum value is reached more rapidly. Seen in Figure 12 and Figure 13.

The reason for the maximum value is due to the series that was given by DWR. The series eventually reaches a maximum value seen by the graphs generated by the model. Seen in Figure 14, Figure 15 and Figure 16.

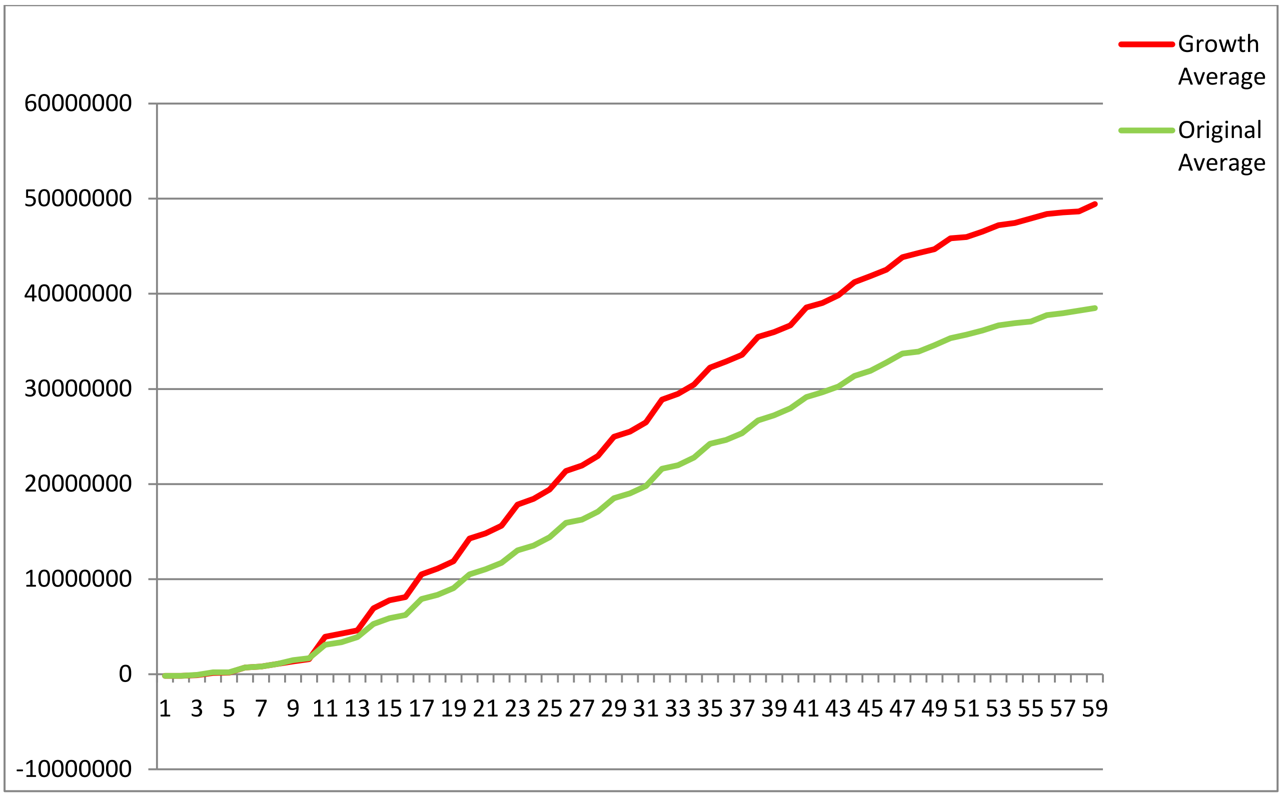


Figure 6 – Growth Profit Difference (Growth = 8%)

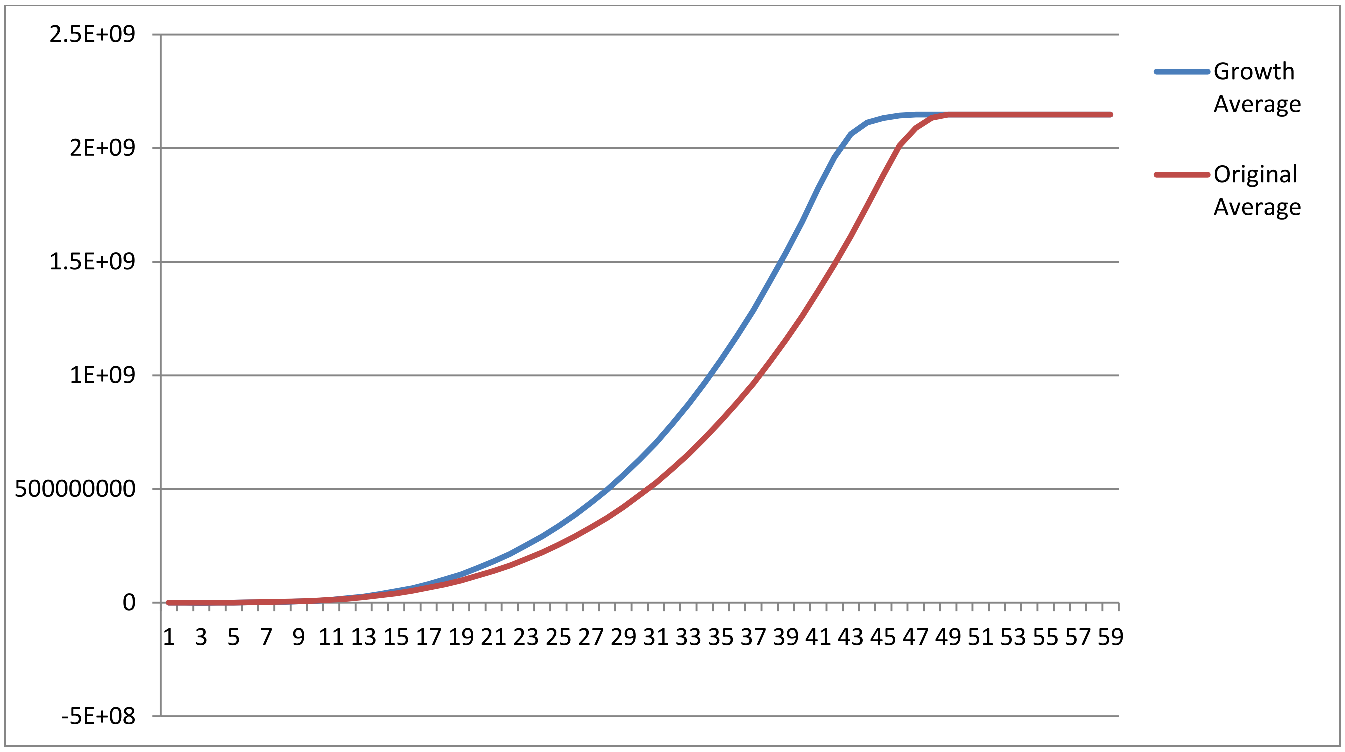


Figure 7 – Growth Value Difference (Growth = 8%)

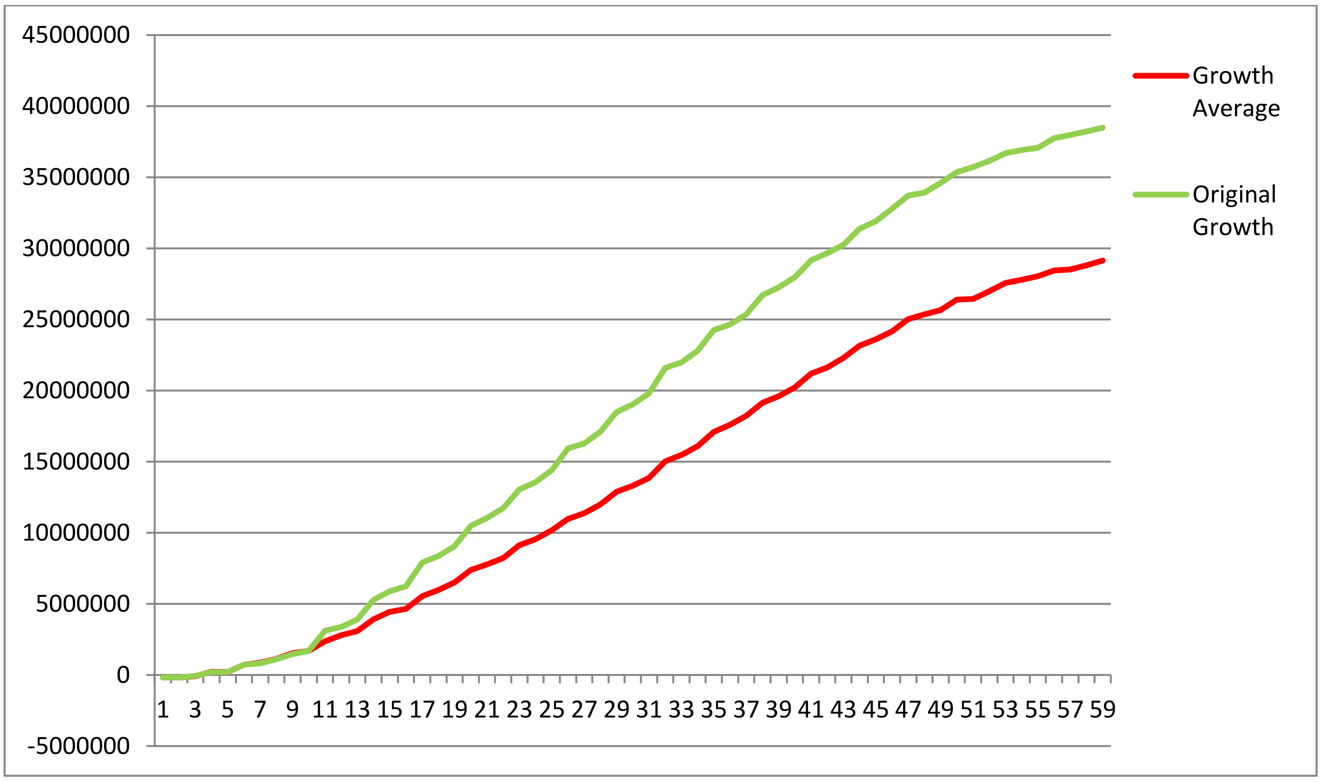


Figure 8 - Growth Profit Difference (Growth = 2%)

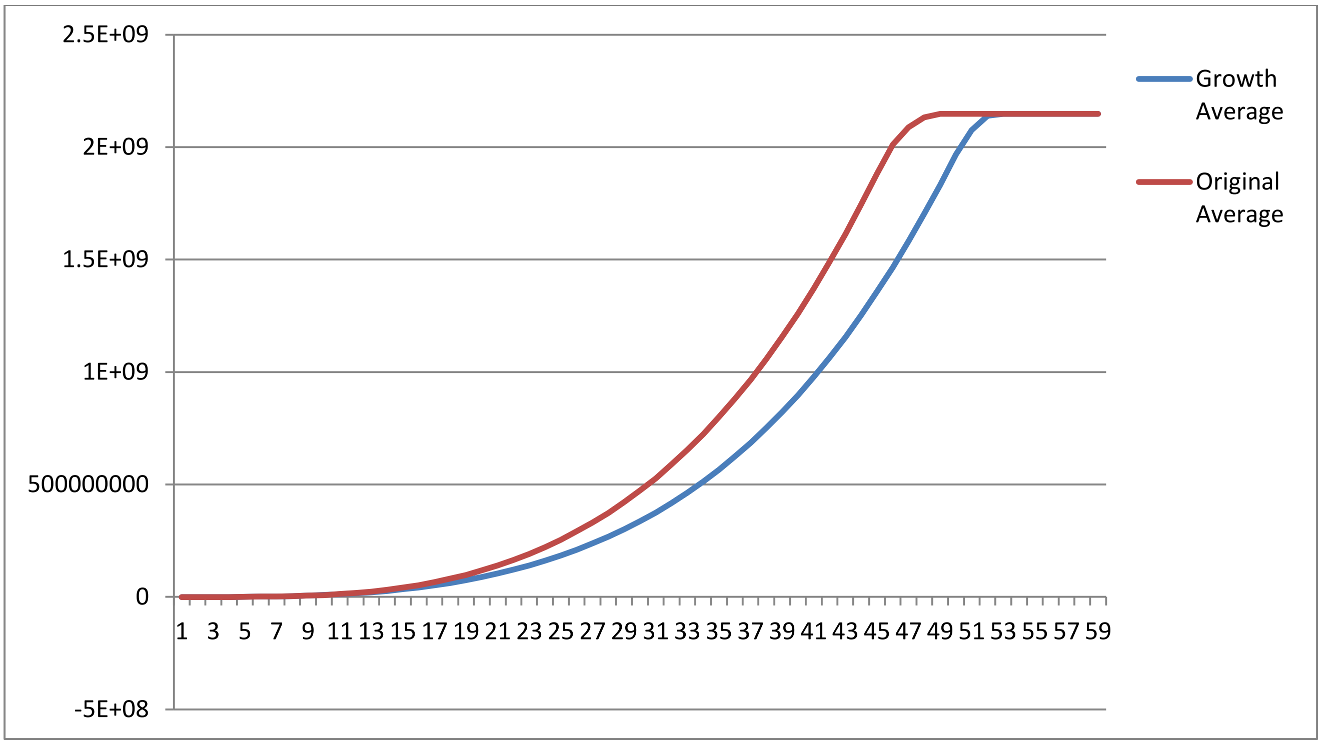


Figure 9 – Growth Value Difference (Growth = 2%)

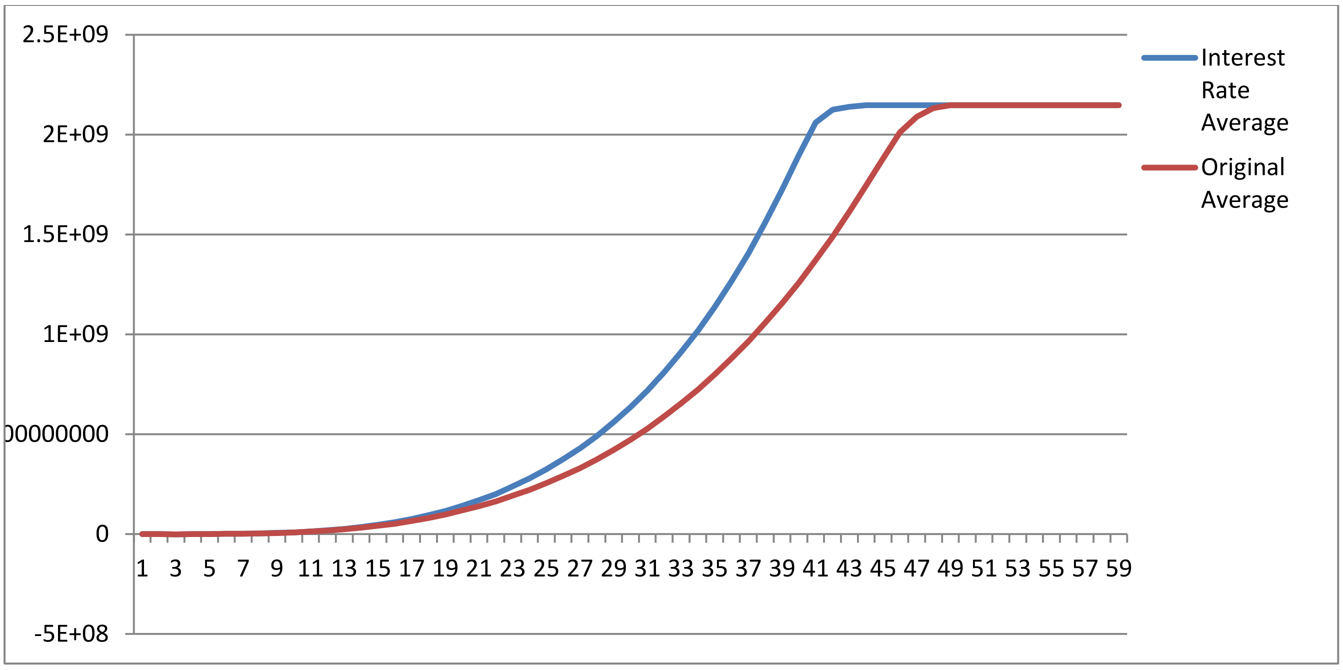


Figure 10 – Interest Rate Value Difference (Interest Rate = 10%)

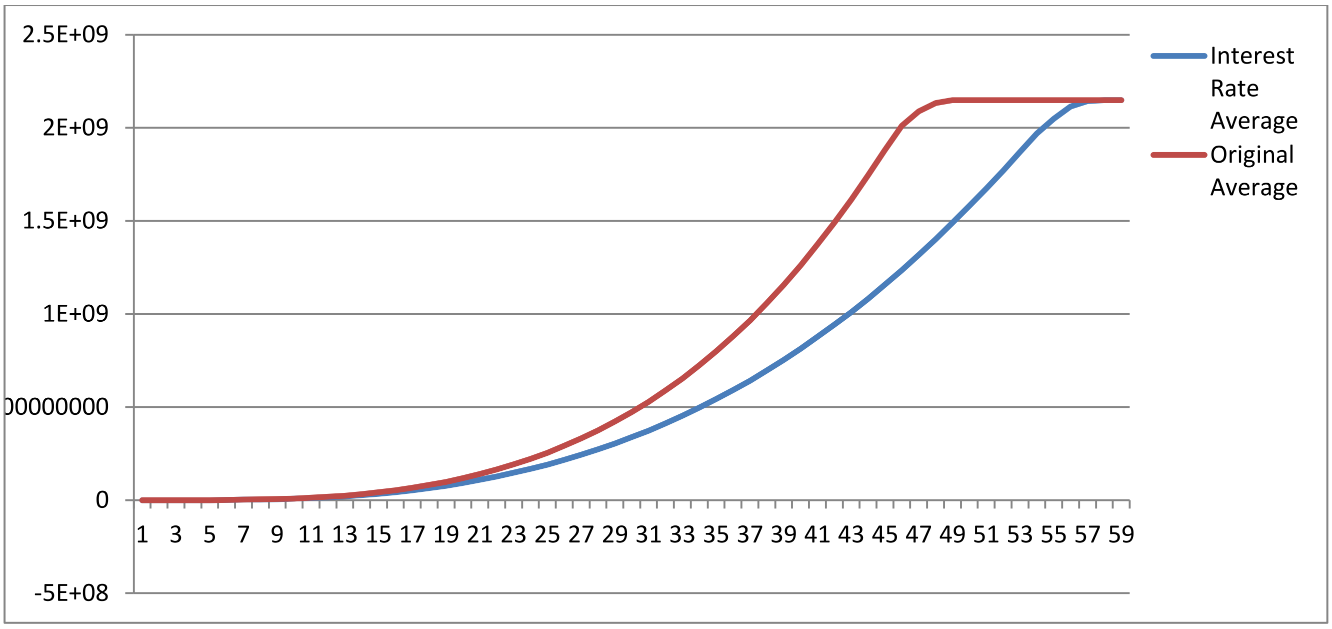


Figure 11 – Interest Rate Difference (Interest Rate = 7%)

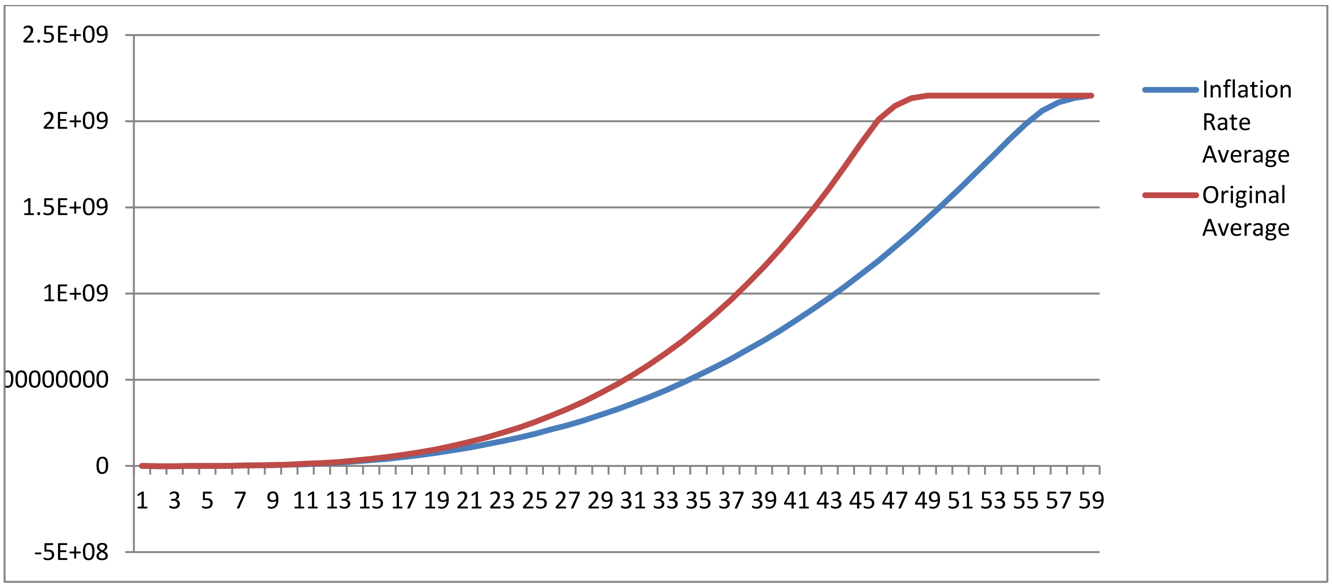


Figure 12 – Inflation Rate Difference (Inflation Rate = 6.5%)

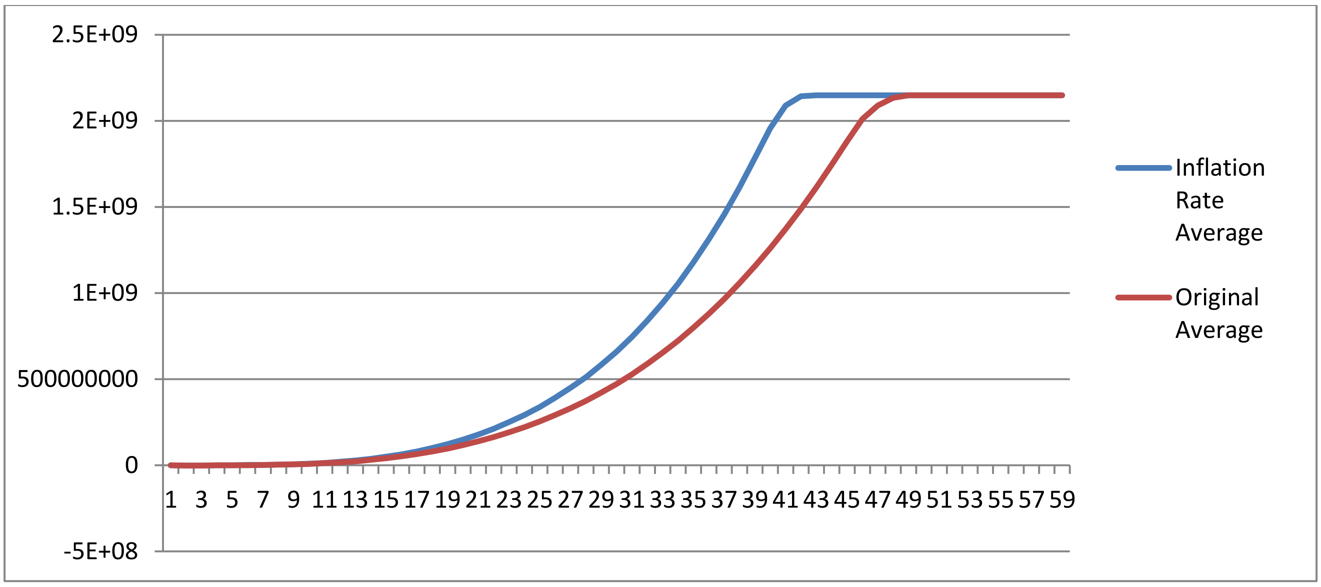


Figure 13 – Inflation Rate Difference (Inflation Rate = 3.5%)

Illustrated by the graphs the model output changes accordingly as the variables change. The variables (Growth, Interest and Inflation) will influence the models' output more significantly and this was proven with correlation coefficients between the original model output and the different changes made. Seen in Table 1 below.

Correlation Coefficient Growth (8%)	
On Profit	1.0000
On Value	1.0000
Correlation Coefficient Growth (2%)	
On Profit	0.999017
On Value	0.982545
Correlation Coefficient Interest Rate (10%)	
On Profit	0.999907
On Value	0.978262
Correlation Coefficient Interest Rate (7%)	
On Profit	0.999964
On Value	0.96884
Correlation Coefficient Inflation Rate (6.5%)	
On Profit	0.999966
On Value	0.966173
Correlation Coefficient Inflation Rate (3.5%)	
On Profit	0.999942
On Value	0.975135

Table 1 – Correlation Coefficients

4.4 Further Development

Further development is eminent since DWR will be moving their operations to another office complex soon. This will mean that more other costs are incurred. Or can simply be added to other sections of the model, if the costing structures are the same. Structure of the branch allocation can easily be changed with minimal knowledge of Java. If DWR want their branch allocation (when a branch opens) and resource allocation (when resources are allocated) it should be changed within the model. More specifically costs can be added like depreciation which was excluded as per requirements.

The model can easily be adapted to fit the total operations of DWR. The most important attribute of the model is that it is written in simple Java code. This concludes that anyone with minimal knowledge of Java will be able to change the structure, with the correct application of the business structures.

4.5 Chapter Summary

The simulation model of DWR is simple to configure, in Java, the basis of the model is secure as proofed by the sensitivity analysis. The model has a user-friendly interface and the output of the model is sensible, meaning that it is simple to gather information from the scenario created by the user. The future of the DWR model can be created to be more complex in building more specific costs into the model such as depreciation and perhaps even debt of DWR.

Chapter 5 - Conclusion

The end result of the model is a generic model, stand-alone program, created in Java. This model will simulate DWR business scenarios, taking costs, growth, tax, inflation and income into account. The costs, income, tax rate, inflation and growth rates are inserted into the model via a user interface. Once the variables are inserted; the simulation results will be generated in terms of profit made across a 60 month projection period.

More information is given than a simple projection. The net present value is calculated from the profit projection. This will display the value of the project of DWR. The break-even point is also calculated and can be seen from the projection, together with the break-even point is the amount of capital needed before the break-even point is reached. This is significant for attracting investors and crucial to indicate how the costing structure should be changed or should remain the same.

The model also indicated the amount of resources that are needed at the end of every month, at the end of the simulation in a grid form. This includes the amount of:

- Paralegals
- Managing Attorneys
- Transaction Coordinators
- Deeds Office Attorneys
- Receptionists
- Messengers
- Admin Staff
- Finance Personal

The output (graphs generated and table) of the DWR model is necessary for business decisions within DWR as it is then possible to allocate the amount of resources needed before the work load becomes overwhelming for the existing resources. Thus the model is used for an indirect forecasting purpose.

Different scenarios can be generated from each run. Comparing each scenario in terms of all the variables, net present value and break-even point, will assist DWR in making informed decisions for the future and attracting investors into DWR.

The model is coded in basic programming code. So that if the structure of the model needs to be modified it will be possible with minor difficulty and allows for further development. If DWR are growing they can adapt the model for different structures.

Throughout the investigation and building stages, crucial questions were revealed, such as; how the business structure should evolve and how certain allocations should be completed, in terms of supporting the resources. The model is practically sound for change, making it dynamic and user friendly giving the freedom to test scenarios and business decision freely, reducing uncertainty and increasing better decision making in business.

The final output from the model is three graphs that show the simulated conveyance tasks completed per month, the profit after tax and the incoming amount of tasks per month. Also giving a final table of the break-even point, the amount of capital needed until the break-even point is met and the amount of resources needed every month. Figure 5 clarifies what the final outputs look like.

References

1. Savage, S.L. 2003. Decision making with Insight. Thomson Learning, Brooks/Cole, 10 Davis Drive, Belmont, California, USA.
2. Winston, W.L. & Venkataramanan, Munirpallam. 2003. Introduction To Mathematical Programming, vol.1. 4th ed. Brooks/Cole.
3. Newman, D.G., Eschenbach, G.T. & Lavelle, J.P. 2009. Engineering Economics Analysis, 10th ed. Oxford University Press.
4. Seal, W., Garrison, R.H. & Noreen, E.W. 2009. Management Accounting, 3rd ed. McGraw Hill.
5. Wenderholm, E. 2004. Eclpss: a Java-based framework for parallel ecosystem simulation and modelling. Elsevier : Environmental Modelling & Software, 20 : 1081 – 1100.
6. Fenton, F.H., Cherry, E.M., Hastings, H.M. & Evans, S.J. 2001. Real-time computer simulations of excitable media : JAVA as a scientific language and as a wrapper for C and FORTAN programs. Elsevier : BioSystems, 64 : 73 – 96.
7. Lee, S. , Posarac, D. & Ellis,N. 2011. Process simulations and economic analysis of biodiesel production processes using fresh and waste vegetable oil and supercritical methanol. Elsevier : Chemical Engineering Research and Design. 89 : 2626 – 2642.
8. Bao, B., El-Halwagi, M.M. & Elbashir, N.O. 2010. Simulation, integration, and economic analysis of gas-to-liquid processes. Elsevier : Fuel Processing Technology. 91 : 703 – 713.
9. Liang, D.Y. 2011. Introduction to Java Programming. 8th ed. Prentice Hall.

Appendix A
(Model Code)

The code below shows the full model code for Java, this includes the visual interface, end interface, calculations and economic analysis.

```
import javax.swing.*;

import javax.swing.border.TitledBorder;

import java.awt.BorderLayout;

import java.awt.Toolkit;

import java.awt.datatransfer.Clipboard;

import java.awt.datatransfer.DataFlavor;

import java.awt.datatransfer.StringSelection;

import java.awt.event.*;


import javax.swing.JComponent;

import javax.swing.JFrame;

import javax.swing.JOptionPane;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.KeyStroke;


import java.awt.*;

import java.io.File;

import java.io.IOException;

import java.util.StringTokenizer;


import org.jfree.chart.ChartFactory;

import org.jfree.chart.JFreeChart;

import org.jfree.chart.plot.PlotOrientation;

import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.chart.ChartUtilities;


@SuppressWarnings("serial")
public class ProjectFinal extends JFrame{
private JTextField jtfAmountOfBranches = new JTextField();
private JTextField jtfIterations = new JTextField();
private JTextField jtfParalegalLimit = new JTextField();
private JTextField jtfMessengerLimit = new JTextField();
private JTextField jtfFinancialLimit = new JTextField();
private JTextField jtfOperations_Manager_Limit = new JTextField();
private JTextField jtfAdmin_Limit = new JTextField();
private JTextField jtfDeeds_Office_Attorney_Limit = new JTextField();
private JTextField jtfReceptionist_limit = new JTextField();
private JTextField jtfSplit = new JTextField();
private JTextField jtfGrowth = new JTextField();
private JTextField jtfInterest = new JTextField();
private JTextField jtfInflation = new JTextField();
private JTextField jtfTax = new JTextField();


private JTextField jtfBranch1_max = new JTextField();
private JTextField jtfBranch1_min = new JTextField();
private JTextField jtfBranch2_max = new JTextField();
private JTextField jtfBranch2_min = new JTextField();
private JTextField jtfBranch3_max = new JTextField();
private JTextField jtfBranch3_min = new JTextField();
private JTextField jtfBranch4_max = new JTextField();
private JTextField jtfBranch4_min = new JTextField();
private JTextField jtfBranch5_max = new JTextField();

private JTextField jtfBranch5_min = new JTextField();

private JTextField jtfBranch6_max = new JTextField();

private JTextField jtfBranch6_min = new JTextField();

private JTextField jtfBranch7_max = new JTextField();

private JTextField jtfBranch7_min = new JTextField();

private JTextField jtfBranch8_max = new JTextField();

private JTextField jtfBranch8_min = new JTextField();

private JTextField jtfBranch9_max = new JTextField();

private JTextField jtfBranch9_min = new JTextField();

private JTextField jtfBranch10_max = new JTextField();

private JTextField jtfBranch10_min = new JTextField();

private JTextField jtfBranch11_max = new JTextField();

private JTextField jtfBranch11_min = new JTextField();

private JTextField jtfBranch12_max = new JTextField();

private JTextField jtfBranch12_min = new JTextField();

private JTextField jtfBranch13_max = new JTextField();

private JTextField jtfBranch13_min = new JTextField();

private JTextField jtfBranch14_max = new JTextField();

private JTextField jtfBranch14_min = new JTextField();

private JTextField jtfBranch15_max = new JTextField();

private JTextField jtfBranch15_min = new JTextField();

private JTextField jtfBranch16_max = new JTextField();

private JTextField jtfBranch16_min = new JTextField();

private JTextField jtfBranch17_max = new JTextField();

private JTextField jtfBranch17_min = new JTextField();

private JTextField jtfBranch18_max = new JTextField();
private JTextField jtfBranch18_min = new JTextField();

private JTextField jtfBranch19_max = new JTextField();

private JTextField jtfBranch19_min = new JTextField();

private JTextField jtfBranch20_max = new JTextField();

private JTextField jtfBranch20_min = new JTextField();


private JTextField jtfPerctage_fee_per_property = new JTextField();


private JTextField jtfBase_fees = new JTextField();


private JTextField jtfUtilities_min = new JTextField();

private JTextField jtfUtilities_max = new JTextField();


private JTextField jtfAdvertising_min = new JTextField();

private JTextField jtfAdvertising_max = new JTextField();


private JTextField jtfEntertainment_min = new JTextField();

private JTextField jtfEntertainment_max = new JTextField();


private JTextField jtfFinancial_Charges_min = new JTextField();

private JTextField jtfFinancial_Charges_max = new JTextField();


private JTextField jtfMaintenance_min = new JTextField();

private JTextField jtfMaintenance_max = new JTextField();


private JTextField jtfMembership_Subscription_min = new JTextField();

private JTextField jtfMembership_Subscription_max = new JTextField();
private JTextField jtfPostage_max = new JTextField();
private JTextField jtfPostage_min = new JTextField();

private JTextField jtfPrice_per_box = new JTextField();

private JTextField jtfParalegal_min = new JTextField();
private JTextField jtfParalegal_max = new JTextField();

private JTextField jtfTransaction_Co_min = new JTextField();
private JTextField jtfTransaction_Co_max = new JTextField();

private JTextField jtfAdmin_min = new JTextField();
private JTextField jtfAdmin_max = new JTextField();

private JTextField jtfOps_Man_min = new JTextField();
private JTextField jtfOps_Man_max = new JTextField();

private JTextField jtfDO_Att_min = new JTextField();
private JTextField jtfDO_Att_max = new JTextField();

private JTextField jtfMan_Att_min = new JTextField();
private JTextField jtfMan_Att_max = new JTextField();

private JTextField jtfReception_min = new JTextField();
private JTextField jtfReception_max = new JTextField();
private JTextField jtfFinancial_min = new JTextField();
private JTextField jtfFinancial_max = new JTextField();


private JTextField jtfMessenger_min = new JTextField();
private JTextField jtfMessenger_max = new JTextField();


private JButton jbtRUN_SIMULATION = new JButton(" Run Simulation");


public ProjectFinal(){
	JPanel p1 = new JPanel(new GridLayout(25,100));
	p1.add(new JLabel(" Amount of Branches"));
	p1.add(jtfAmountOfBranches);
	p1.add(new JLabel(" Iterations"));
	p1.add(jtfIterations);
	p1.add(new JLabel(" Paralegal Limit"));
	p1.add(jtfParalegalLimit);
	p1.add(new JLabel(" Messenger Limit"));
	p1.add(jtfMessengerLimit);
	p1.add(new JLabel(" Financial Personal Limit"));
	p1.add(jtfFinancialLimit);
	p1.add(new JLabel(" Operations Manager Limit"));
	p1.add(jtfOperations_Manager_Limit);
	p1.add(new JLabel(" Admin Limit"));
	p1.add(jtfAdmin_Limit);
	p1.add(new JLabel(" Deeds Office Attorney Limit"));
	p1.add(jtfDeeds_Office_Attorney_Limit);
	p1.add(new JLabel(" Receptionist Limit"));
```

```
p1.add(jtfReceptionist_limit);

p1.add(new JLabel(" Spilt of the Project Costs"));

p1.add(jtfSplit);

p1.add(new JLabel(" Growth per Year "));

p1.add(jtfGrowth);

p1.add(new JLabel(" Interest Rate "));

p1.add(jtfInterest);

p1.add(new JLabel(" Inflation "));

p1.add(jtfInflation);

p1.add(new JLabel(" Tax Rate "));

p1.add(jtfTax);


p1.add(new JLabel(" Branch 1 Area Max "));

p1.add(jtfBranch1_max);

p1.add(new JLabel(" Branch 1 Area Min "));

p1.add(jtfBranch1_min);


p1.add(new JLabel(" Branch 2 Area Max "));

p1.add(jtfBranch2_max);

p1.add(new JLabel(" Branch 2 Area Min "));

p1.add(jtfBranch2_min);


p1.add(new JLabel(" Branch 3 Area Max "));

p1.add(jtfBranch3_max);

p1.add(new JLabel(" Branch 3 Area Min "));

p1.add(jtfBranch3_min);
```

```
p1.add(new JLabel(" Branch 4 Area Max "));
p1.add(jtfBranch4_max);
p1.add(new JLabel(" Branch 4 Area Min "));
p1.add(jtfBranch4_min);

p1.add(new JLabel(" Branch 5 Area Max "));
p1.add(jtfBranch5_max);
p1.add(new JLabel(" Branch 5 Area Min "));
p1.add(jtfBranch5_min);

p1.add(new JLabel(" Branch 6 Area Max "));
p1.add(jtfBranch6_max);
p1.add(new JLabel(" Branch 6 Area Min "));
p1.add(jtfBranch6_min);

p1.add(new JLabel(" Branch 7 Area Max "));
p1.add(jtfBranch7_max);
p1.add(new JLabel(" Branch 7 Area Min "));
p1.add(jtfBranch7_min);

p1.add(new JLabel(" Branch 8 Area Max "));
p1.add(jtfBranch8_max);
p1.add(new JLabel(" Branch 8 Area Min "));
p1.add(jtfBranch8_min);

p1.add(new JLabel(" Branch 9 Area Max "));
p1.add(jtfBranch9_max);
```

```
p1.add(new JLabel(" Branch 9 Area Min "));

p1.add(jtfBranch9_min);


p1.add(new JLabel(" Branch 10 Area Max "));

p1.add(jtfBranch10_max);

p1.add(new JLabel(" Branch 10 Area Min "));

p1.add(jtfBranch10_min);


p1.add(new JLabel(" Branch 11 Area Max "));

p1.add(jtfBranch11_max);

p1.add(new JLabel(" Branch 11 Area Min "));

p1.add(jtfBranch11_min);


p1.add(new JLabel(" Branch 12 Area Max "));

p1.add(jtfBranch12_max);

p1.add(new JLabel(" Branch 12 Area Min "));

p1.add(jtfBranch12_min);


p1.add(new JLabel(" Branch 13 Area Max "));

p1.add(jtfBranch13_max);

p1.add(new JLabel(" Branch 13 Area Min "));

p1.add(jtfBranch13_min);


p1.add(new JLabel(" Branch 14 Area Max "));

p1.add(jtfBranch14_max);

p1.add(new JLabel(" Branch 14 Area Min "));

p1.add(jtfBranch14_min);
```

```
p1.add(new JLabel(" Branch 15 Area Max "));

p1.add(jtfBranch15_max);

p1.add(new JLabel(" Branch 15 Area Min "));

p1.add(jtfBranch15_min);


p1.add(new JLabel(" Branch 16 Area Max "));

p1.add(jtfBranch16_max);

p1.add(new JLabel(" Branch 16 Area Min "));

p1.add(jtfBranch16_min);


p1.add(new JLabel(" Branch 17 Area Max "));

p1.add(jtfBranch17_max);

p1.add(new JLabel(" Branch 17 Area Min "));

p1.add(jtfBranch17_min);


p1.add(new JLabel(" Branch 18 Area Max "));

p1.add(jtfBranch18_max);

p1.add(new JLabel(" Branch 18 Area Min "));

p1.add(jtfBranch18_min);


p1.add(new JLabel(" Branch 19 Area Max "));

p1.add(jtfBranch19_max);

p1.add(new JLabel(" Branch 19 Area Min "));

p1.add(jtfBranch19_min);


p1.add(new JLabel(" Branch 20 Area Max "));
```

```
p1.add(jtfBranch20_max);

p1.add(new JLabel(" Branch 20 Area Min "));

p1.add(jtfBranch20_min);


p1.add(new JLabel(" Percentage Fee Per Property "));

p1.add(jtfPerctage_fee_per_property);


p1.add(new JLabel(" Base Fees per Transport "));

p1.add(jtfBase_fees);


p1.add(new JLabel(" Postage Max "));

p1.add(jtfPostage_max);


p1.add(new JLabel(" Postage Min "));

p1.add(jtfPostage_min);


p1.add(new JLabel(" Storage (Price per box) "));

p1.add(jtfPrice_per_box);


p1.add(new JLabel(" Utilities Min "));

p1.add(jtfUtilities_min);

p1.add(new JLabel(" Utilities Max "));

p1.add(jtfUtilities_max);


p1.add(new JLabel(" Advertising Min "));

p1.add(jtfAdvertising_min);

p1.add(new JLabel(" Advertising Max "));
```

```
p1.add(jtfAdvertising_max);


p1.add(new JLabel(" Entertainment Min "));

p1.add(jtfEntertainment_min);

p1.add(new JLabel(" Entertainment Max "));

p1.add(jtfEntertainment_max);


p1.add(new JLabel(" Financial Charges Min "));

p1.add(jtfFinancial_Charges_min);

p1.add(new JLabel(" Financial Charges Max "));

p1.add(jtfFinancial_Charges_max);


p1.add(new JLabel(" Maintenance Min "));

p1.add(jtfMaintenance_min);

p1.add(new JLabel(" Maintenance Max "));

p1.add(jtfMaintenance_max);


p1.add(new JLabel(" Membership & Subscription Min "));

p1.add(jtfMembership_Subscription_min);

p1.add(new JLabel(" Membership & Subscription Max "));

p1.add(jtfMembership_Subscription_max);


p1.add(new JLabel(" Paralegal Salary Min "));

p1.add(jtfParalegal_min);

p1.add(new JLabel(" Paralegal Salary Max "));

p1.add(jtfParalegal_max);
```

```
p1.add(new JLabel(" Transaction Coordinator Salary Min "));

p1.add(jtfTransaction_Co_min);

p1.add(new JLabel(" Transaction Coordinator Salary Max "));

p1.add(jtfTransaction_Co_max);


p1.add(new JLabel(" Administration Staff Salary Min "));

p1.add(jtfAdmin_min);

p1.add(new JLabel(" Administration Staff Salary Max "));

p1.add(jtfAdmin_max);


p1.add(new JLabel(" Operations Manager Salary Min "));

p1.add(jtfOps_Man_min);

p1.add(new JLabel(" Operations Manager Salary Max "));

p1.add(jtfOps_Man_max);


p1.add(new JLabel(" Deeds Office Attorney Salary Min "));

p1.add(jtfDO_Att_min);

p1.add(new JLabel(" Deeds Office Attorney Salary Max "));

p1.add(jtfDO_Att_max);


p1.add(new JLabel(" Managing Attorney Salary Min "));

p1.add(jtfMan_Att_min);

p1.add(new JLabel(" Managing Attorney Salary Max "));

p1.add(jtfMan_Att_max);


p1.add(new JLabel(" Reception Salary Min "));

p1.add(jtfReception_min);
```

```
        p1.add(new JLabel(" Reception Salary Max "));
        p1.add(jtfReception_max);


        p1.add(new JLabel(" Financial Personal Salary Min "));
        p1.add(jtfFinancial_min);
        p1.add(new JLabel(" Financial Personal Salary Max "));
        p1.add(jtfFinancial_max);


        p1.add(new JLabel(" Messenger Salary Min "));
        p1.add(jtfMessenger_min);
        p1.add(new JLabel(" Messenger Salary Max "));
        p1.add(jtfMessenger_max);


        p1.setBorder(new TitledBorder(" DWR Simualtion Model "));


        JPanel p2 = new JPanel(new FlowLayout(FlowLayout.RIGHT));
        p2.add(jbtRUN_SIMULATION);


        add(p1, BorderLayout.CENTER);
        add(p2, BorderLayout.SOUTH);


        jbtRUN_SIMULATION.addActionListener( new ButtonListener());



    }
```

```
public class ButtonListener implements ActionListener {


        public void actionPerformed(ActionEvent e) {


                // TODO Auto-generated method stub

                double iterations = Double.parseDouble(jtfIterations.getText());

                double amount_of_branches
=Double.parseDouble(jtfAmountOfBranches.getText());

                double [] tasks_per_month = new double [60];

                double [] total_tasks_per_month = new double [60];

                double [][] growth_per_month = new double [(int)amount_of_branches][60];



                for (int i = 0 ; i < tasks_per_month.length ; i++){

                        total_tasks_per_month[i]=0;

                }

                for (int i = 0 ; i < tasks_per_month.length ; i++){

                        tasks_per_month[i]=0;

                }


                double commision = 8;


                double paralegal_limit = Double.parseDouble(jtfParalegalLimit.getText());


                double messenger_limit = Double.parseDouble(jtfMessengerLimit.getText());

                double fin_limit = Double.parseDouble(jtfFinancialLimit.getText());

                double ops_man_limit
=Double.parseDouble(jtfOperations_Manager_Limit.getText());

                double admin_limit = Double.parseDouble(jtfAdmin_Limit.getText());
```

```
                    double DO_att_limit =
Double.parseDouble(jtfDeeds_Office_Attorney_Limit.getText());

                    double reception_limit = Double.parseDouble(jtfReceptionist_limit.getText());

                    double split = Double.parseDouble(jtfSplit.getText())/100;


                    double growth = Double.parseDouble(jtfGrowth.getText());


                    double interest = Double.parseDouble(jtfInterest.getText());

                    double inflation = Double.parseDouble(jtfInflation.getText());


                    double real_interest_rate = ((interest/100) - (inflation/100))/(1 + (inflation/100));


                    double tax = Double.parseDouble(jtfTax.getText());


                    double ratio_fee_property_worth =
Double.parseDouble(jtfPerctage_fee_per_property.getText());



                    double branch1max = Double.parseDouble(jtfBranch1_max.getText());

                    double branch1min = Double.parseDouble(jtfBranch1_min.getText());

                    double branch1= Math.random()*(branch1max - branch1min) + branch1min;


                    double branch2max = Double.parseDouble(jtfBranch2_max.getText());

                    double branch2min = Double.parseDouble(jtfBranch2_min.getText());

                    double branch2= Math.random()*(branch2max - branch2min) + branch2min;


                    double branch3max = Double.parseDouble(jtfBranch3_max.getText());

                    double branch3min = Double.parseDouble(jtfBranch3_min.getText());
                    double branch3= Math.random()*(branch3max - branch3min) + branch3min;


                    double branch4max = Double.parseDouble(jtfBranch4_max.getText());
                    double branch4min = Double.parseDouble(jtfBranch4_min.getText());
                    double branch4= Math.random()*(branch4max - branch4min) + branch4min;


                    double branch5max = Double.parseDouble(jtfBranch5_max.getText());
                    double branch5min = Double.parseDouble(jtfBranch5_min.getText());
                    double branch5= Math.random()*(branch5max - branch5min) + branch5min;


                    double branch6max = Double.parseDouble(jtfBranch6_max.getText());
                    double branch6min = Double.parseDouble(jtfBranch6_min.getText());
                    double branch6= Math.random()*(branch6max - branch6min) + branch6min;


                    double branch7max = Double.parseDouble(jtfBranch7_max.getText());
                    double branch7min = Double.parseDouble(jtfBranch7_min.getText());
                    double branch7= Math.random()*(branch7max - branch7min) + branch7min;


                    double branch8max = Double.parseDouble(jtfBranch8_max.getText());
                    double branch8min = Double.parseDouble(jtfBranch8_min.getText());
                    double branch8= Math.random()*(branch8max - branch8min) + branch8min;


                    double branch9max = Double.parseDouble(jtfBranch9_max.getText());
                    double branch9min = Double.parseDouble(jtfBranch9_min.getText());
                    double branch9= Math.random()*(branch9max - branch9min) + branch9min;


                    double branch10max = Double.parseDouble(jtfBranch10_max.getText());
                    double branch10min = Double.parseDouble(jtfBranch10_min.getText());
                    double branch10= Math.random()*(branch10max - branch10min) + branch10min;

                    double branch11max = Double.parseDouble(jtfBranch11_max.getText());
                    double branch11min = Double.parseDouble(jtfBranch11_min.getText());
                    double branch11= Math.random()*(branch11max - branch11min) + branch11min;

                    double branch12max = Double.parseDouble(jtfBranch12_max.getText());
                    double branch12min = Double.parseDouble(jtfBranch12_min.getText());
                    double branch12= Math.random()*(branch12max - branch12min) + branch12min;

                    double branch13max = Double.parseDouble(jtfBranch13_max.getText());
                    double branch13min = Double.parseDouble(jtfBranch13_min.getText());
                    double branch13= Math.random()*(branch13max - branch13min) + branch13min;

                    double branch14max = Double.parseDouble(jtfBranch14_max.getText());
                    double branch14min = Double.parseDouble(jtfBranch14_min.getText());
                    double branch14= Math.random()*(branch14max - branch14min) + branch14min;

                    double branch15max = Double.parseDouble(jtfBranch15_max.getText());
                    double branch15min = Double.parseDouble(jtfBranch15_min.getText());
                    double branch15= Math.random()*(branch15max - branch15min) + branch15min;

                    double branch16max = Double.parseDouble(jtfBranch16_max.getText());
                    double branch16min = Double.parseDouble(jtfBranch16_min.getText());
                    double branch16= Math.random()*(branch16max - branch16min) + branch16min;
```

```
                    double branch17max = Double.parseDouble(jtfBranch17_max.getText());

                    double branch17min = Double.parseDouble(jtfBranch17_min.getText());

                    double branch17= Math.random()*(branch17max - branch17min) + branch17min;


                    double branch18max = Double.parseDouble(jtfBranch18_max.getText());

                    double branch18min = Double.parseDouble(jtfBranch18_min.getText());

                    double branch18= Math.random()*(branch18max - branch18min) + branch18min;


                    double branch19max = Double.parseDouble(jtfBranch19_max.getText());

                    double branch19min = Double.parseDouble(jtfBranch19_min.getText());

                    double branch19= Math.random()*(branch19max - branch19min) + branch19min;


                    double branch20max = Double.parseDouble(jtfBranch20_max.getText());

                    double branch20min = Double.parseDouble(jtfBranch20_min.getText());

                    double branch20= Math.random()*(branch20max - branch20min) + branch20min;





                    //Costs!


double fee = Double.parseDouble(jtfBase_fees.getText());


double utilities_max = Double.parseDouble(jtfUtilities_max.getText());

double utilities_min = Double.parseDouble(jtfUtilities_min.getText());

double utilities = Math.random()*(utilities_max - utilities_min) + utilities_min;
double utilities_true = utilities*split;

double advertising_max = Double.parseDouble(jtfAdvertising_max.getText());
double advertising_min = Double.parseDouble(jtfAdvertising_min.getText());
double advertising = Math.random()*(advertising_max - advertising_min) + advertising_min;

double advertising_true=advertising*split;



double entertainment_max = Double.parseDouble(jtfEntertainment_max.getText());
double entertainment_min = Double.parseDouble(jtfEntertainment_min.getText());
double entertainment = Math.random()*(entertainment_max - entertainment_min)+
entertainment_min;

double entertainment_true = entertainment*split;

double fin_charge_max = Double.parseDouble(jtfFinancial_Charges_max.getText());
double fin_charge_min = Double.parseDouble(jtfFinancial_Charges_min.getText());
double fin_charge = Math.random()*(fin_charge_max - fin_charge_min)+ fin_charge_min;

double fin_charge_true = fin_charge*split;

double maintenance_max = Double.parseDouble(jtfMaintenance_max.getText());
double maintenance_min = Double.parseDouble(jtfMaintenance_min.getText());
double maintenance = Math.random()*(maintenance_max - maintenance_min)+ maintenance_min;

double maintenance_true = maintenance*split;
double membership_subscription_max =
Double.parseDouble(jtfMembership_Subscription_max.getText());

double membership_subscription_min =
Double.parseDouble(jtfMembership_Subscription_min.getText());

double membership_subscription = Math.random()*(membership_subscription_max -
membership_subscription_min)+ membership_subscription_min;


double membership_subscription_true = membership_subscription*split;


double storage_per_box = Double.parseDouble(jtfPrice_per_box.getText());

double files_per_box = 18;


double postage_max = Double.parseDouble(jtfPostage_max.getText());

double postage_min = Double.parseDouble(jtfPostage_min.getText());

double postage = Math.random()*(postage_max - postage_min) + postage_min;


double postage_true = postage*split;




double paralegal_max = Double.parseDouble(jtfParalegal_max.getText());

double paralegal_min = Double.parseDouble(jtfParalegal_min.getText());

double paralegal_salary = Math.random()*(paralegal_max - paralegal_min) + paralegal_min;


double transaction_co_max = Double.parseDouble(jtfTransaction_Co_max.getText());

double transaction_co_min = Double.parseDouble(jtfTransaction_Co_min.getText());

double transaction_co_salary = Math.random()*(transaction_co_max - transaction_co_min) +
transaction_co_min;


double admin_max = Double.parseDouble(jtfAdmin_max.getText());
double admin_min = Double.parseDouble(jtfAdmin_min.getText());

double admin_salary = Math.random()*(admin_max - admin_min) + admin_min;


double ops_manager_max = Double.parseDouble(jtfOps_Man_max.getText());

double ops_manager_min = Double.parseDouble(jtfOps_Man_min.getText());

double ops_manager_salary = Math.random()*(ops_manager_max - ops_manager_min) +
ops_manager_min;


double DO_att_max = Double.parseDouble(jtfDO_Att_max.getText());

double DO_att_min = Double.parseDouble(jtfDO_Att_min.getText());

double DO_att_salary = Math.random()*(DO_att_max - DO_att_min) + DO_att_min;


double man_att_max = Double.parseDouble(jtfMan_Att_max.getText());

double man_att_min = Double.parseDouble(jtfMan_Att_min.getText());

double man_att_salary = Math.random()*(man_att_max - man_att_min) + man_att_min;


double reception_max = Double.parseDouble(jtfReception_max.getText());

double reception_min = Double.parseDouble(jtfReception_min.getText());

double reception_salary = Math.random()*(reception_max - reception_min) + reception_min;


double financial_max = Double.parseDouble(jtfFinancial_max.getText());

double financial_min = Double.parseDouble(jtfFinancial_min.getText());

double financial_salary = Math.random()*(financial_max - financial_min) + financial_min;


double messenger_max = Double.parseDouble(jtfMessenger_max.getText());

double messenger_min = Double.parseDouble(jtfMessenger_min.getText());

double messenger_salary = Math.random()*(messenger_max - messenger_min) + messenger_min;
double [] cost_projection = new double[60];
double [] income_projection = new double[60];
double [] profit_projection = new double[60];
double [] after_tax_profit_projection = new double [60];

double [] transaction_co_s = new double [60];
double []amount_paralegals = new double [60];
double []messengers = new double [60];
double []fin = new double [60];
double []ops_man = new double [60];
double []admin = new double [60];
double []DO_att = new double [60];
double []Reception = new double [60];
double []Total_em = new double [60];
double [] man_att=new double[60];
double [] tasks_done  = new double[60];
double [] storage = new double [60];
double [] branch_income_projection = new double[60];

for (int i = 0 ; i < cost_projection.length ; i++){
        cost_projection[i]=0;
        income_projection[i]=0;
        profit_projection[i]=0;
        transaction_co_s[i]=0;
        man_att[i]=0;
        amount_paralegals[i]=0;
        messengers[i]=0;
```

```
            fin[i]=0;

            ops_man[i]=0;

            admin[i]=0;

            DO_att[i]=0;

            Reception[i]=0;

            Total_em[i] = 0;

            tasks_done[i]=0;

            storage[i]=0;

            branch_income_projection[i]=0;

}


// current

// COSTS!!




double [][] branches = new double [(int)amount_of_branches][60];




for (int b = 0; b < branches.length ; b++ ){

double [] totaltasks = new double[60];

            totaltasks[0] = 0;

            totaltasks[1] = 0;

 double [] avertasks = new double [60];

            avertasks[0] = 0;

            avertasks[1] = 0;


for (int u = 0; u < iterations; u++){                                    //Original Branch!!!!
```

```
double [][] probabilities = new double[60][];
        //month1
        probabilities[0]  = new double [1];
        probabilities[0][0] = 0;

        //month2
        probabilities[1]  = new double [1];
        probabilities[1][0] = 0;

        //month3 - month60

        /*month3*/  probabilities[2]   = new double [2];
        /*month4*/  probabilities[3]   = new double [5];
        /*month5*/  probabilities[4]   = new double [7];
        /*month6*/  probabilities[5]   = new double [10];
        /*month7*/  probabilities[6]   = new double [11];
        /*month8*/  probabilities[7]   = new double [12];
        /*month9*/  probabilities[8]   = new double [13];
        /*month10*/ probabilities[9]   = new double [15];
        /*month11*/ probabilities[10]  = new double [17];
        /*month12*/ probabilities[11]  = new double [19];
        /*month13*/ probabilities[12]  = new double [19];
        /*month14*/ probabilities[13]  = new double [20];
        /*month15*/ probabilities[14]  = new double [22];
        /*month16*/ probabilities[15]  = new double [25];
        /*month17*/ probabilities[16]  = new double [25];
        /*month18*/ probabilities[17]  = new double [28];
```

```
/*month19*/ probabilities[18]  = new double [28];
/*month20*/ probabilities[19]  = new double [30];
/*month21*/ probabilities[20]  = new double [30];
/*month22*/ probabilities[21]  = new double [30];
/*month23*/ probabilities[22]  = new double [30];
/*month24*/ probabilities[23]  = new double [30];
/*month25*/ probabilities[24]  = new double [30];
/*month26*/ probabilities[25]  = new double [32];
/*month27*/ probabilities[26]  = new double [32];
/*month28*/ probabilities[27]  = new double [33];
/*month29*/ probabilities[28]  = new double [33];
/*month30*/ probabilities[29]  = new double [35];
/*month31*/ probabilities[30]  = new double [35];
/*month32*/ probabilities[31]  = new double [35];
/*month33*/ probabilities[32]  = new double [37];
/*month34*/ probabilities[33]  = new double [37];
/*month35*/ probabilities[34]  = new double [37];
/*month36*/ probabilities[35]  = new double [37];
/*month37*/ probabilities[36]  = new double [37];
/*month38*/ probabilities[37]  = new double [37];
/*month39*/ probabilities[38]  = new double [37];
/*month40*/ probabilities[39]  = new double [37];
/*month41*/ probabilities[40]  = new double [37];
/*month42*/ probabilities[41]  = new double [37];
/*month43*/ probabilities[42]  = new double [37];
/*month44*/ probabilities[43]  = new double [37];
/*month45*/ probabilities[44]  = new double [37];
```

```
/*month46*/ probabilities[45]  = new double [37];

/*month47*/ probabilities[46]  = new double [37];

/*month48*/ probabilities[47]  = new double [37];

/*month49*/ probabilities[48]  = new double [37];

/*month50*/ probabilities[49]  = new double [37];

/*month51*/ probabilities[50]  = new double [37];

/*month52*/ probabilities[51]  = new double [37];

/*month53*/ probabilities[52]  = new double [37];

/*month54*/ probabilities[53]  = new double [37];

/*month55*/ probabilities[54]  = new double [37];

/*month56*/ probabilities[55]  = new double [37];

/*month57*/ probabilities[56]  = new double [37];

/*month58*/ probabilities[57]  = new double [37];

/*month59*/ probabilities[58]  = new double [37];

/*month60*/ probabilities[59]  = new double [37];


int [][] tasks = new int[60][];


tasks[0]  = new int [1];

tasks[0][0] = 0;


tasks[1]  = new int [1];

tasks[1][0] = 0;


/*month3*/  tasks[2]   = new int [2];

/*month4*/  tasks[3]   = new int [5];

/*month5*/  tasks[4]   = new int [7];
```

```
/*month6*/  tasks[5]   = new int [10];
/*month7*/  tasks[6]   = new int [11];
/*month8*/  tasks[7]   = new int [12];
/*month9*/  tasks[8]   = new int [13];
/*month10*/ tasks[9]   = new int [15];
/*month11*/ tasks[10]  = new int [17];
/*month12*/ tasks[11]  = new int [19];
/*month13*/ tasks[12]  = new int [19];
/*month14*/ tasks[13]  = new int [20];
/*month15*/ tasks[14]  = new int [22];
/*month16*/ tasks[15]  = new int [25];
/*month17*/ tasks[16]  = new int [25];
/*month18*/ tasks[17]  = new int [28];
/*month19*/ tasks[18]  = new int [28];
/*month20*/ tasks[19]  = new int [30];
/*month21*/ tasks[20]  = new int [30];
/*month22*/ tasks[21]  = new int [30];
/*month23*/ tasks[22]  = new int [30];
/*month24*/ tasks[23]  = new int [30];
/*month25*/ tasks[24]  = new int [30];
/*month26*/ tasks[25]  = new int [32];
/*month27*/ tasks[26]  = new int [32];
/*month28*/ tasks[27]  = new int [33];
/*month29*/ tasks[28]  = new int [33];
/*month30*/ tasks[29]  = new int [35];
/*month31*/ tasks[30]  = new int [35];
/*month32*/ tasks[31]  = new int [35];
```

```
/*month33*/ tasks[32]  = new int [37];
/*month34*/ tasks[33]  = new int [37];
/*month35*/ tasks[34]  = new int [37];
/*month36*/ tasks[35]  = new int [37];
/*month37*/ tasks[36]  = new int [37];
/*month38*/ tasks[37]  = new int [37];
/*month39*/ tasks[38]  = new int [37];
/*month40*/ tasks[39]  = new int [37];
/*month41*/ tasks[40]  = new int [37];
/*month42*/ tasks[41]  = new int [37];
/*month43*/ tasks[42]  = new int [37];
/*month44*/ tasks[43]  = new int [37];
/*month45*/ tasks[44]  = new int [37];
/*month46*/ tasks[45]  = new int [37];
/*month47*/ tasks[46]  = new int [37];
/*month48*/ tasks[47]  = new int [37];
/*month49*/ tasks[48]  = new int [37];
/*month50*/ tasks[49]  = new int [37];
/*month51*/ tasks[50]  = new int [37];
/*month52*/ tasks[51]  = new int [37];
/*month53*/ tasks[52]  = new int [37];
/*month54*/ tasks[53]  = new int [37];
/*month55*/ tasks[54]  = new int [37];
/*month56*/ tasks[55]  = new int [37];
/*month57*/ tasks[56]  = new int [37];
/*month58*/ tasks[57]  = new int [37];
/*month59*/ tasks[58]  = new int [37];
```

```
            /*month60*/ tasks[59]  = new int [37];


            double [] tasksgenerated = new double[60];

            tasksgenerated[0] = 0;

            tasksgenerated[1] = 0;




// probabiliets in a month per task

for (int i = 2; i < probabilities.length; i++){

 for (int k = 0; k < probabilities[i].length; k++){

                        probabilities[i][k] = Math.random();

            }

   }

// assign value to a task accepted

for ( int i = 2; i < probabilities.length; i++){

            for (int k = 0; k < probabilities[i].length; k++){


                        if ( probabilities[i][k] < .42 ){


                                    tasks[i][k] = 1;}


                         else{

                                    ;

                         }


                        }

            }
            double y;

            // total tasks generated per month

             for (int i = 2; i < tasks.length; i++) {

              for(int j = 0; j < tasks[i].length; j++){

                            tasksgenerated[i] = (tasks[i][j]+tasksgenerated[i]);

              }

              }

             for (int i = 2; i < totaltasks.length;i++)

                           totaltasks[i]=tasksgenerated[i] + totaltasks[i];

             for (int i = 0; i < tasksgenerated.length; i++)


             for(int j = 0; j < avertasks.length; j++){

                           y = Math.floor(Math.random()*2);

                           if (y == 0){

                           avertasks[j]=Math.ceil(totaltasks[j]/iterations);}

                           else{

                           avertasks[j]=Math.floor(totaltasks[j]/iterations);}

                           }

}









            for (int j = 0 ; j < 60; j++ ){

    branches[b][j]=avertasks[j];
```

```
            }



for (int i = 0; i < growth_per_month.length ; i++ ){

            for (int j = 0 ; j < growth_per_month[i].length ; j++){


                        growth_per_month[i][j] = (growth/100)*branches[i][j];

            }

            }

for (int i = 0 ; i < growth_per_month.length ; i++){

            for (int k = 0 ; k < tasks_per_month.length-11 ; k ++){

            branches[i][k] = growth_per_month[i][k] + branches[i][k];


}

}


}

// income from branches

for ( int i = 0; i < branches[0].length; i++){

            branch_income_projection[i] =
branches[0][i]*branch1*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i];

}

for ( int i = 0; i < branches[1].length -3; i++){

            branch_income_projection[i+3] =
branches[1][i]*branch2*(ratio_fee_property_worth/100)*(commision/100)+
branch_income_projection[i+3];

}

for ( int i = 0; i < branches[2].length -6; i++){
```

```
            branch_income_projection[i+6] =
branches[2][i]*branch3*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i+6];

}

for ( int i = 0; i < branches[3].length -9; i++){

            branch_income_projection[i+9] =
branches[3][i]*branch4*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i+9];

}

for ( int i = 0; i < branches[4].length -12; i++){

            branch_income_projection[i+12] =
branches[4][i]*branch5*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i+12];

}

for ( int i = 0; i < branches[5].length -15; i++){

            branch_income_projection[i+15] =
branches[5][i]*branch6*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i+15];

}

for ( int i = 0; i < branches[6].length -18; i++){

            branch_income_projection[i+18] =
branches[6][i]*branch7*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i+18];

}

for ( int i = 0; i < branches[7].length -21; i++){

            branch_income_projection[i+21] =
branches[7][i]*branch8*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i+21];

}

for ( int i = 0; i < branches[8].length -24; i++){

            branch_income_projection[i+24] =
branches[8][i]*branch9*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i+24];
```

```
}

for ( int i = 0; i < branches[9].length -27; i++){

            branch_income_projection[i+27] =
branches[9][i]*branch10*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i+27];

}

for ( int i = 0; i < branches[10].length -30; i++){

            branch_income_projection[i+30] =
branches[10][i]*branch11*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i+30];

}

for ( int i = 0; i < branches[11].length -33; i++){

            branch_income_projection[i+33] =
branches[11][i]*branch12*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i+33];

}

for ( int i = 0; i < branches[12].length -36; i++){

            branch_income_projection[i+36] =
branches[12][i]*branch13*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i+36];

}

for ( int i = 0; i < branches[13].length -39; i++){

            branch_income_projection[i+39] =
branches[13][i]*branch14*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i+39];

}

for ( int i = 0; i < branches[14].length -42; i++){

            branch_income_projection[i+42] =
branches[14][i]*branch15*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i+42];

}

if (amount_of_branches ==16){
```

```
for ( int i = 0; i < branches[15].length -45; i++){

            branch_income_projection[i+45] =
branches[15][i]*branch16*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i+45];

}}


if (amount_of_branches ==17){

for ( int i = 0; i < branches[15].length -45; i++){

            branch_income_projection[i+45] =
branches[15][i]*branch16*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i+45];}

for ( int i = 0; i < branches[16].length -48; i++){

            branch_income_projection[i+48] =
branches[16][i]*branch17*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i+48];

}}


if (amount_of_branches ==18){

for ( int i = 0; i < branches[15].length -45; i++){

            branch_income_projection[i+45] =
branches[15][i]*branch16*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i+45];}

for ( int i = 0; i < branches[16].length -48; i++){

            branch_income_projection[i+48] =
branches[16][i]*branch17*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i+48];}

for ( int i = 0; i < branches[17].length -51; i++){

            branch_income_projection[i+51] =
branches[17][i]*branch18*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i+51];

}}
```

```
if (amount_of_branches ==19){

for ( int i = 0; i < branches[15].length -45; i++){

            branch_income_projection[i+45] =
branches[15][i]*branch16*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i+45];}

for ( int i = 0; i < branches[16].length -48; i++){

            branch_income_projection[i+48] =
branches[16][i]*branch17*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i+48];}

for ( int i = 0; i < branches[17].length -51; i++){

            branch_income_projection[i+51] =
branches[17][i]*branch18*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i+51];}

for ( int i = 0; i < branches[18].length -54; i++){

            branch_income_projection[i+54] =
branches[18][i]*branch19*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i+54];

}}
```

```
if (amount_of_branches ==19){

for ( int i = 0; i < branches[15].length -45; i++){

            branch_income_projection[i+45] =
branches[15][i]*branch16*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i+45];}

for ( int i = 0; i < branches[16].length -48; i++){

            branch_income_projection[i+48] =
branches[16][i]*branch17*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i+48];}

for ( int i = 0; i < branches[17].length -51; i++){

            branch_income_projection[i+51] =
branches[17][i]*branch18*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i+51];}

for ( int i = 0; i < branches[18].length -54; i++){
```

```
            branch_income_projection[i+54] =
branches[18][i]*branch19*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i+54];}

for ( int i = 0; i < branches[19].length -57; i++){

            branch_income_projection[i+57] =
branches[19][i]*branch20*(ratio_fee_property_worth/100)*(commision/100) +
branch_income_projection[i+57];

}}




for (int i = 0 ;i < branches[0].length;i++){

            tasks_per_month[i]=tasks_per_month[i] + branches[0][i];}


            for ( int i = 0 ; i < branches[1].length - 3 ; i++ ){

                        tasks_per_month[i+3]= tasks_per_month[i+3] + branches[1][i];

            }

            for ( int i = 0 ; i < branches[2].length - 6 ; i++ ){

                        tasks_per_month[i+6]= tasks_per_month[i+6] + branches[2][i];

            }

            for ( int i = 0 ; i < branches[3].length - 9 ; i++ ){

                        tasks_per_month[i+9]= tasks_per_month[i+9] + branches[3][i];

            }

            for ( int i = 0 ; i < branches[4].length - 12 ; i++ ){

                        tasks_per_month[i+12]= tasks_per_month[i+12] + branches[4][i];

            }

            for ( int i = 0 ; i < branches[5].length - 15 ; i++ ){

                        tasks_per_month[i+15]= tasks_per_month[i+15] + branches[5][i];

            }

            for ( int i = 0 ; i < branches[6].length - 18 ; i++ ){
```

```
        tasks_per_month[i+18]= tasks_per_month[i+18] + branches[6][i];
}
for ( int i = 0 ; i < branches[7].length - 21 ; i++ ){
        tasks_per_month[i+21]= tasks_per_month[i+21] + branches[7][i];
}
for ( int i = 0 ; i < branches[8].length - 24 ; i++ ){
        tasks_per_month[i+24]= tasks_per_month[i+24] + branches[8][i];
}
for ( int i = 0 ; i < branches[9].length - 27 ; i++ ){
        tasks_per_month[i+27]= tasks_per_month[i+27] + branches[9][i];
}
for ( int i = 0 ; i < branches[10].length - 30 ; i++ ){
        tasks_per_month[i+30]= tasks_per_month[i+30] + branches[10][i];
}
for ( int i = 0 ; i < branches[11].length - 33 ; i++ ){
        tasks_per_month[i+33]= tasks_per_month[i+33] + branches[11][i];
}
for ( int i = 0 ; i < branches[12].length - 36 ; i++ ){
        tasks_per_month[i+36]= tasks_per_month[i+36] + branches[12][i];
}
for ( int i = 0 ; i < branches[13].length - 39 ; i++ ){
        tasks_per_month[i+39]= tasks_per_month[i+39] + branches[13][i];
}
for ( int i = 0 ; i < branches[14].length - 42 ; i++ ){
        tasks_per_month[i+42]= tasks_per_month[i+42] + branches[14][i];
}
```

```
if (amount_of_branches == 16){

        for ( int i = 0 ; i < branches[15].length - 45 ; i++ ){

                tasks_per_month[i+45]= tasks_per_month[i+45] + branches[15][i];

        }

}


if (amount_of_branches == 17){

        for ( int i = 0 ; i < branches[15].length - 45 ; i++ ){

                tasks_per_month[i+45]= tasks_per_month[i+45] + branches[15][i];

        }

        for ( int i = 0 ; i < branches[16].length - 47 ; i++ ){

                tasks_per_month[i+48]= tasks_per_month[i+48] + branches[16][i];

        }

}

if (amount_of_branches == 18){

        for ( int i = 0 ; i < branches[15].length - 45 ; i++ ){

                tasks_per_month[i+45]= tasks_per_month[i+45] + branches[15][i];

        }

        for ( int i = 0 ; i < branches[16].length - 48 ; i++ ){

                tasks_per_month[i+48]= tasks_per_month[i+48] + branches[16][i];

        }

        for ( int i = 0 ; i < branches[17].length - 51 ; i++ ){

                tasks_per_month[i+51]= tasks_per_month[i+51] + branches[17][i];

        }

}

if (amount_of_branches == 19){

        for ( int i = 0 ; i < branches[15].length - 45 ; i++ ){
                tasks_per_month[i+45]= tasks_per_month[i+45] + branches[15][i];
        }
        for ( int i = 0 ; i < branches[16].length - 48 ; i++ ){
                tasks_per_month[i+48]= tasks_per_month[i+48] + branches[16][i];
        }
        for ( int i = 0 ; i < branches[17].length - 51 ; i++ ){
                tasks_per_month[i+51]= tasks_per_month[i+51] + branches[17][i];
        }
        for ( int i = 0 ; i < branches[18].length - 54 ; i++ ){
                tasks_per_month[i+54]= tasks_per_month[i+54] + branches[17][i];
        }
}
if (amount_of_branches == 19){
        for ( int i = 0 ; i < branches[15].length - 45 ; i++ ){
                tasks_per_month[i+45]= tasks_per_month[i+45] + branches[15][i];
        }
        for ( int i = 0 ; i < branches[16].length - 48 ; i++ ){
                tasks_per_month[i+48]= tasks_per_month[i+48] + branches[16][i];
        }
        for ( int i = 0 ; i < branches[17].length - 51 ; i++ ){
                tasks_per_month[i+51]= tasks_per_month[i+51] + branches[17][i];
        }
        for ( int i = 0 ; i < branches[18].length - 54 ; i++ ){
                tasks_per_month[i+54]= tasks_per_month[i+54] + branches[17][i];
        }
}
if (amount_of_branches == 20){
		for ( int i = 0 ; i < branches[15].length - 45 ; i++ ){

			tasks_per_month[i+45]= tasks_per_month[i+45] + branches[15][i];

		}

		for ( int i = 0 ; i < branches[16].length - 48 ; i++ ){

			tasks_per_month[i+48]= tasks_per_month[i+48] + branches[16][i];

		}

		for ( int i = 0 ; i < branches[17].length - 51 ; i++ ){

			tasks_per_month[i+51]= tasks_per_month[i+51] + branches[17][i];

		}

		for ( int i = 0 ; i < branches[18].length - 54 ; i++ ){

			tasks_per_month[i+54]= tasks_per_month[i+54] + branches[18][i];

		}

		for ( int i = 0 ; i < branches[19].length - 57 ; i++ ){

			tasks_per_month[i+57]= tasks_per_month[i+57] + branches[19][i];

		}

	}


	for ( int i = 0 ; i < tasks_per_month.length ; i++){

		if ( tasks_per_month [i] == 0){

			;

		}

		else{

		double prob[] = new double [(int)tasks_per_month[i]];

		for (int j = 0 ; j < prob.length ; j++){

		prob[j]=Math.random();

		if ( i < 57) {

		if ( prob[j] <0.1 ){
```

```
                        tasks_done[i+3] = tasks_done[i+3] + 1;

                        }}

                if ( i < 56){

                if (prob[j] >0.1 && prob[j] < 0.3){

                        tasks_done[i+4] = tasks_done[i+4] + 1;

                }}

                if ( i < 55){

                if (prob[j] >0.3 && prob[j] < 0.6){

                        tasks_done[i+5] = tasks_done[i+5] + 1;

                }}

                if ( i < 54){

                if (prob[j] >0.6 && prob[j] < 0.9){

                        tasks_done[i+6] = tasks_done[i+6] + 1;

                }}

                if ( i < 53){

                if (prob[j] >0.9){

                        tasks_done[i+7] = tasks_done[i+7] + 1;

                }}


        }

        }

        }

        for ( int i = 1 ; i < tasks_per_month.length ; i ++ ){

                total_tasks_per_month[i]= (tasks_per_month[i]  + total_tasks_per_month[i-1]) -
tasks_done[i];

        }

        for ( int i = 0 ; i < tasks_done.length ; i++){

                storage[i]= Math.ceil((tasks_done[i]/files_per_box))*storage_per_box;
```

```
}


for (int i = 0 ; i < transaction_co_s.length ; i++ ){


        if (i <9 ){

                transaction_co_s[i]=1;

        }

        if (i <18 && i >= 9 ){

                transaction_co_s[i]=2;

        }

        if (i <27 && i >= 18 ){

                transaction_co_s[i]=3;

        }

        if (i <36 && i >= 27 ){

                transaction_co_s[i]=4;

        }

        if (i <60 && i >= 36 ){

                transaction_co_s[i]=5;

        }

        if (amount_of_branches == 16){

                if (i <60 && i >= 45 ){

                        transaction_co_s[i]=6;

                }}

        if (amount_of_branches == 17){

                if (i <60 && i >= 45 ){

                        transaction_co_s[i]=6;

                }}
```

```
                    if (amount_of_branches == 18){
                              if (i <60 && i >= 45 ){
                                        transaction_co_s[i]=6;
                              }}
                    if (amount_of_branches == 19){
                              if (i <60 && i >= 54 ){
                                        transaction_co_s[i]=7;
                              }}
                    if (amount_of_branches == 19){
                    if (i <60 && i >= 54 ){
                                        transaction_co_s[i]=7;
                                        }}
          }
for (int i = 0 ; i < man_att.length ; i++ ){


                    if (i <9 ){
                              man_att[i]=1;
                    }
                    if (i <18 && i >= 9 ){
                              man_att[i]=2;
                    }
                    if (i <27 && i >= 18 ){
                              man_att[i]=3;
                    }
                    if (i <36 && i >= 27 ){
                              man_att[i]=4;
                    }
```

```
                if (i <60 && i >= 36 ){
                        man_att[i]=5;
                }
                if (amount_of_branches == 16){
                        if (i <60 && i >= 45 ){
                                man_att[i]=6;
                        }}
                if (amount_of_branches == 17){
                        if (i <60 && i >= 45 ){
                                man_att[i]=6;
                        }}
                if (amount_of_branches == 18){
                        if (i <60 && i >= 45 ){
                                man_att[i]=6;
                        }}
                if (amount_of_branches == 19){
                        if (i <60 && i >= 54 ){
                                man_att[i]=7;
                        }}
                if (amount_of_branches == 20){
                if (i <60 && i >= 54 ){
                        man_att[i]=7;
                                }}
        }


amount_paralegals[0]=1;
```

```
for (int i = 1; i < total_tasks_per_month.length; i++){
        if (total_tasks_per_month[i]%paralegal_limit > 6){
         amount_paralegals[i] = Math.ceil(total_tasks_per_month[i]/paralegal_limit);
        }


        else{
                amount_paralegals[i] =  amount_paralegals[i-1];
        }}


for (int i = 0; i < amount_paralegals.length; i++){
        messengers[i] = Math.ceil(amount_paralegals[i]/messenger_limit);}
for (int i = 0; i < amount_paralegals.length; i++){
        fin[i] = Math.ceil(amount_paralegals[i]/fin_limit);}
for (int i = 0; i < amount_paralegals.length; i++){
        ops_man[i] = Math.ceil(amount_paralegals[i]/ops_man_limit);}


for (int i = 0; i < amount_paralegals.length; i++){
        admin[i] = Math.ceil(total_tasks_per_month[i]/admin_limit);


        if(admin[i]<=1){
                admin[i]=1;
        }
}


for (int i = 0; i < amount_paralegals.length; i++){
        DO_att[i] = Math.ceil(tasks_done[i]/DO_att_limit);
        if(DO_att[i]<=1){
```

```
                    DO_att[i]=1;}

        }


for (int i = 0; i < Total_em.length ; i++){

        Total_em[i] = transaction_co_s[i] + man_att[i] + amount_paralegals[i] + messengers[i] + fin[i]
+ ops_man[i] + admin[i] + DO_att[i];

}


for (int i = 0 ; i < Reception.length; i++ ){


 Reception[i] = Math.ceil(Total_em[i]/reception_limit);


}


for (int i = 0 ; i < cost_projection.length; i++){

        cost_projection[i] = amount_paralegals[i]*paralegal_salary +
transaction_co_s[i]*transaction_co_salary + man_att[i]*man_att_salary

                        + messengers[i]*messenger_salary + fin[i]*financial_salary +
ops_man[i]*ops_manager_salary + admin[i]*admin_salary + DO_att[i]*DO_att_salary +
Reception[i]*reception_salary +

                        utilities_true + advertising_true + entertainment_true + fin_charge_true +
maintenance_true + membership_subscription_true + postage_true + storage[i];

}


for (int i = 0 ; i < income_projection.length ; i++){

        income_projection[i] = tasks_per_month[i]*fee + branch_income_projection[i];

}


for (int i = 0 ; i < profit_projection.length ; i++){
        profit_projection[i] = income_projection[i] - cost_projection[i];

}

//Analysis


double [] future_value = new double[60];

for(int i = 0; i < future_value.length; i++){

        future_value[i]=0;

}





for (int i = 0 ; i < after_tax_profit_projection.length ; i++){

        if(profit_projection[i]> 0){

                after_tax_profit_projection[i]=profit_projection[i];

        }

        else{

        after_tax_profit_projection[i] = profit_projection[i]*(1-(tax/100));

        }

        }


for(int i = 0; i < future_value.length ; i++){

        for (int p = 0 ; p <= i ; p++){

                future_value[i]
=(int)(after_tax_profit_projection[p]*Math.pow((1+real_interest_rate),p)+future_value[i]);

        }



}

//break-even
```

```
double break_even_point =0;


for (int i = 0 ; i < after_tax_profit_projection.length ; i++){
        if(after_tax_profit_projection[i]<0){
                break_even_point = break_even_point + 1;
        }
}

double capital_true = 0;
for(int i=0; i < break_even_point;i++){
        capital_true=after_tax_profit_projection[i]+capital_true;
}
DefaultCategoryDataset dataset = new DefaultCategoryDataset();

dataset.setValue(after_tax_profit_projection[0], "Profit", "1" );
dataset.setValue(after_tax_profit_projection[1], "Profit", "2" );
dataset.setValue(after_tax_profit_projection[2], "Profit", "3" );
dataset.setValue(after_tax_profit_projection[3], "Profit", "4" );
dataset.setValue(after_tax_profit_projection[4], "Profit", "5" );
dataset.setValue(after_tax_profit_projection[5], "Profit", "6" );
dataset.setValue(after_tax_profit_projection[6], "Profit", "7" );
dataset.setValue(after_tax_profit_projection[7], "Profit", "8" );
dataset.setValue(after_tax_profit_projection[8], "Profit", "9" );
dataset.setValue(after_tax_profit_projection[9], "Profit", "10" );
dataset.setValue(after_tax_profit_projection[10], "Profit", "11" );
dataset.setValue(after_tax_profit_projection[11], "Profit", "12" );

dataset.setValue(after_tax_profit_projection[12], "Profit", "13" );

dataset.setValue(after_tax_profit_projection[13], "Profit", "14" );

dataset.setValue(after_tax_profit_projection[14], "Profit", "15" );

dataset.setValue(after_tax_profit_projection[15], "Profit", "16" );

dataset.setValue(after_tax_profit_projection[16], "Profit", "17" );

dataset.setValue(after_tax_profit_projection[17], "Profit", "18" );

dataset.setValue(after_tax_profit_projection[18], "Profit", "19" );

dataset.setValue(after_tax_profit_projection[19], "Profit", "20" );

dataset.setValue(after_tax_profit_projection[20], "Profit", "21" );

dataset.setValue(after_tax_profit_projection[21], "Profit", "22" );

dataset.setValue(after_tax_profit_projection[22], "Profit", "23" );

dataset.setValue(after_tax_profit_projection[23], "Profit", "24" );

dataset.setValue(after_tax_profit_projection[24], "Profit", "25" );

dataset.setValue(after_tax_profit_projection[25], "Profit", "26" );

dataset.setValue(after_tax_profit_projection[26], "Profit", "27" );

dataset.setValue(after_tax_profit_projection[27], "Profit", "28" );

dataset.setValue(after_tax_profit_projection[28], "Profit", "29" );

dataset.setValue(after_tax_profit_projection[29], "Profit", "30" );

dataset.setValue(after_tax_profit_projection[30], "Profit", "31" );

dataset.setValue(after_tax_profit_projection[31], "Profit", "32" );

dataset.setValue(after_tax_profit_projection[32], "Profit", "33" );

dataset.setValue(after_tax_profit_projection[33], "Profit", "34" );

dataset.setValue(after_tax_profit_projection[34], "Profit", "35" );

dataset.setValue(after_tax_profit_projection[35], "Profit", "36" );

dataset.setValue(after_tax_profit_projection[36], "Profit", "37" );

dataset.setValue(after_tax_profit_projection[37], "Profit", "38" );
dataset.setValue(after_tax_profit_projection[38], "Profit", "39" );

dataset.setValue(after_tax_profit_projection[39], "Profit", "40" );

dataset.setValue(after_tax_profit_projection[40], "Profit", "41" );

dataset.setValue(after_tax_profit_projection[41], "Profit", "42" );

dataset.setValue(after_tax_profit_projection[42], "Profit", "43" );

dataset.setValue(after_tax_profit_projection[43], "Profit", "44" );

dataset.setValue(after_tax_profit_projection[44], "Profit", "45" );

dataset.setValue(after_tax_profit_projection[45], "Profit", "46" );

dataset.setValue(after_tax_profit_projection[46], "Profit", "47" );

dataset.setValue(after_tax_profit_projection[47], "Profit", "48" );

dataset.setValue(after_tax_profit_projection[48], "Profit", "49" );

dataset.setValue(after_tax_profit_projection[49], "Profit", "50" );

dataset.setValue(after_tax_profit_projection[50], "Profit", "51" );

dataset.setValue(after_tax_profit_projection[51], "Profit", "52" );

dataset.setValue(after_tax_profit_projection[52], "Profit", "53" );

dataset.setValue(after_tax_profit_projection[53], "Profit", "54" );

dataset.setValue(after_tax_profit_projection[54], "Profit", "55" );

dataset.setValue(after_tax_profit_projection[55], "Profit", "56" );

dataset.setValue(after_tax_profit_projection[56], "Profit", "57" );

dataset.setValue(after_tax_profit_projection[57], "Profit", "58" );

dataset.setValue(after_tax_profit_projection[58], "Profit", "59" );

dataset.setValue(after_tax_profit_projection[59], "Profit", "60" );




JFreeChart chart = ChartFactory.createBarChart("Profit Projection after tax", "Month", "Profit in
Rands", dataset, PlotOrientation.VERTICAL, false, true, false);
```

```
try { ChartUtilities.saveChartAsJPEG(new File("C:\\After Tax Projection.jpg"), chart, 1500, 700);

} catch (IOException a){

        System.err.println("Problem");

}



DefaultCategoryDataset datasetA = new DefaultCategoryDataset();


datasetA.setValue(tasks_per_month[0], "Tasks", "1" );

datasetA.setValue(tasks_per_month[1], "Tasks", "2" );

datasetA.setValue(tasks_per_month[2], "Tasks", "3" );

datasetA.setValue(tasks_per_month[3], "Tasks", "4" );

datasetA.setValue(tasks_per_month[4], "Tasks", "5" );

datasetA.setValue(tasks_per_month[5], "Tasks", "6" );

datasetA.setValue(tasks_per_month[6], "Tasks", "7" );

datasetA.setValue(tasks_per_month[7], "Tasks", "8" );

datasetA.setValue(tasks_per_month[8], "Tasks", "9" );

datasetA.setValue(tasks_per_month[9], "Tasks", "10" );

datasetA.setValue(tasks_per_month[10], "Tasks", "11" );

datasetA.setValue(tasks_per_month[11], "Tasks", "12" );

datasetA.setValue(tasks_per_month[12], "Tasks", "13" );

datasetA.setValue(tasks_per_month[13], "Tasks", "14" );

datasetA.setValue(tasks_per_month[14], "Tasks", "15" );

datasetA.setValue(tasks_per_month[15], "Tasks", "16" );

datasetA.setValue(tasks_per_month[16], "Tasks", "17" );

datasetA.setValue(tasks_per_month[17], "Tasks", "18" );

datasetA.setValue(tasks_per_month[18], "Tasks", "19" );
datasetA.setValue(tasks_per_month[19], "Tasks", "20" );

datasetA.setValue(tasks_per_month[20], "Tasks", "21" );

datasetA.setValue(tasks_per_month[21], "Tasks", "22" );

datasetA.setValue(tasks_per_month[22], "Tasks", "23" );

datasetA.setValue(tasks_per_month[23], "Tasks", "24" );

datasetA.setValue(tasks_per_month[24], "Tasks", "25" );

datasetA.setValue(tasks_per_month[25], "Tasks", "26" );

datasetA.setValue(tasks_per_month[26], "Tasks", "27" );

datasetA.setValue(tasks_per_month[27], "Tasks", "28" );

datasetA.setValue(tasks_per_month[28], "Tasks", "29" );

datasetA.setValue(tasks_per_month[29], "Tasks", "30" );

datasetA.setValue(tasks_per_month[30], "Tasks", "31" );

datasetA.setValue(tasks_per_month[31], "Tasks", "32" );

datasetA.setValue(tasks_per_month[32], "Tasks", "33" );

datasetA.setValue(tasks_per_month[33], "Tasks", "34" );

datasetA.setValue(tasks_per_month[34], "Tasks", "35" );

datasetA.setValue(tasks_per_month[35], "Tasks", "36" );

datasetA.setValue(tasks_per_month[36], "Tasks", "37" );

datasetA.setValue(tasks_per_month[37], "Tasks", "38" );

datasetA.setValue(tasks_per_month[38], "Tasks", "39" );

datasetA.setValue(tasks_per_month[39], "Tasks", "40" );

datasetA.setValue(tasks_per_month[40], "Tasks", "41" );

datasetA.setValue(tasks_per_month[41], "Tasks", "42" );

datasetA.setValue(tasks_per_month[42], "Tasks", "43" );

datasetA.setValue(tasks_per_month[43], "Tasks", "44" );

datasetA.setValue(tasks_per_month[44], "Tasks", "45" );

datasetA.setValue(tasks_per_month[45], "Tasks", "46" );
datasetA.setValue(tasks_per_month[46], "Tasks", "47" );

datasetA.setValue(tasks_per_month[47], "Tasks", "48" );

datasetA.setValue(tasks_per_month[48], "Tasks", "49" );

datasetA.setValue(tasks_per_month[49], "Tasks", "50" );

datasetA.setValue(tasks_per_month[50], "Tasks", "51" );

datasetA.setValue(tasks_per_month[51], "Tasks", "52" );

datasetA.setValue(tasks_per_month[52], "Tasks", "53" );

datasetA.setValue(tasks_per_month[53], "Tasks", "54" );

datasetA.setValue(tasks_per_month[54], "Tasks", "55" );

datasetA.setValue(tasks_per_month[55], "Tasks", "56" );

datasetA.setValue(tasks_per_month[56], "Tasks", "57" );

datasetA.setValue(tasks_per_month[57], "Tasks", "58" );

datasetA.setValue(tasks_per_month[58], "Tasks", "59" );

datasetA.setValue(tasks_per_month[59], "Tasks", "60" );


JFreeChart chartA = ChartFactory.createBarChart("Amount of Incoming Tasks per Month", "Month",
"Tasks per Manth", datasetA, PlotOrientation.VERTICAL, false, true, false);


try { ChartUtilities.saveChartAsJPEG(new File("C:\\Incoming Tasks Per Month.jpg"), chartA, 1500,
700);

} catch (IOException a){

        System.err.println("Problem");

}


DefaultCategoryDataset datasetB = new DefaultCategoryDataset();


datasetB.setValue(total_tasks_per_month[0], "Total Tasks", "1" );

datasetB.setValue(total_tasks_per_month[1], "Total Tasks", "2" );
datasetB.setValue(total_tasks_per_month[2], "Total Tasks", "3" );

datasetB.setValue(total_tasks_per_month[3], "Total Tasks", "4" );

datasetB.setValue(total_tasks_per_month[4], "Total Tasks", "5" );

datasetB.setValue(total_tasks_per_month[5], "Total Tasks", "6" );

datasetB.setValue(total_tasks_per_month[6], "Total Tasks", "7" );

datasetB.setValue(total_tasks_per_month[7], "Total Tasks", "8" );

datasetB.setValue(total_tasks_per_month[8], "Total Tasks", "9" );

datasetB.setValue(total_tasks_per_month[9], "Total Tasks", "10" );

datasetB.setValue(total_tasks_per_month[10], "Total Tasks", "11" );

datasetB.setValue(total_tasks_per_month[11], "Total Tasks", "12" );

datasetB.setValue(total_tasks_per_month[12], "Total Tasks", "13" );

datasetB.setValue(total_tasks_per_month[13], "Total Tasks", "14" );

datasetB.setValue(total_tasks_per_month[14], "Total Tasks", "15" );

datasetB.setValue(total_tasks_per_month[15], "Total Tasks", "16" );

datasetB.setValue(total_tasks_per_month[16], "Total Tasks", "17" );

datasetB.setValue(total_tasks_per_month[17], "Total Tasks", "18" );

datasetB.setValue(total_tasks_per_month[18], "Total Tasks", "19" );

datasetB.setValue(total_tasks_per_month[19], "Total Tasks", "20" );

datasetB.setValue(total_tasks_per_month[20], "Total Tasks", "21" );

datasetB.setValue(total_tasks_per_month[21], "Total Tasks", "22" );

datasetB.setValue(total_tasks_per_month[22], "Total Tasks", "23" );

datasetB.setValue(total_tasks_per_month[23], "Total Tasks", "24" );

datasetB.setValue(total_tasks_per_month[24], "Total Tasks", "25" );

datasetB.setValue(total_tasks_per_month[25], "Total Tasks", "26" );

datasetB.setValue(total_tasks_per_month[26], "Total Tasks", "27" );

datasetB.setValue(total_tasks_per_month[27], "Total Tasks", "28" );

datasetB.setValue(total_tasks_per_month[28], "Total Tasks", "29" );
datasetB.setValue(total_tasks_per_month[29], "Total Tasks", "30" );

datasetB.setValue(total_tasks_per_month[30], "Total Tasks", "31" );

datasetB.setValue(total_tasks_per_month[31], "Total Tasks", "32" );

datasetB.setValue(total_tasks_per_month[32], "Total Tasks", "33" );

datasetB.setValue(total_tasks_per_month[33], "Total Tasks", "34" );

datasetB.setValue(total_tasks_per_month[34], "Total Tasks", "35" );

datasetB.setValue(total_tasks_per_month[35], "Total Tasks", "36" );

datasetB.setValue(total_tasks_per_month[36], "Total Tasks", "37" );

datasetB.setValue(total_tasks_per_month[37], "Total Tasks", "38" );

datasetB.setValue(total_tasks_per_month[38], "Total Tasks", "39" );

datasetB.setValue(total_tasks_per_month[39], "Total Tasks", "40" );

datasetB.setValue(total_tasks_per_month[40], "Total Tasks", "41" );

datasetB.setValue(total_tasks_per_month[41], "Total Tasks", "42" );

datasetB.setValue(total_tasks_per_month[42], "Total Tasks", "43" );

datasetB.setValue(total_tasks_per_month[43], "Total Tasks", "44" );

datasetB.setValue(total_tasks_per_month[44], "Total Tasks", "45" );

datasetB.setValue(total_tasks_per_month[45], "Total Tasks", "46" );

datasetB.setValue(total_tasks_per_month[46], "Total Tasks", "47" );

datasetB.setValue(total_tasks_per_month[47], "Total Tasks", "48" );

datasetB.setValue(total_tasks_per_month[48], "Total Tasks", "49" );

datasetB.setValue(total_tasks_per_month[49], "Total Tasks", "50" );

datasetB.setValue(total_tasks_per_month[50], "Total Tasks", "51" );

datasetB.setValue(total_tasks_per_month[51], "Total Tasks", "52" );

datasetB.setValue(total_tasks_per_month[52], "Total Tasks", "53" );

datasetB.setValue(total_tasks_per_month[53], "Total Tasks", "54" );

datasetB.setValue(total_tasks_per_month[54], "Total Tasks", "55" );

datasetB.setValue(total_tasks_per_month[55], "Total Tasks", "56" );
datasetB.setValue(total_tasks_per_month[56], "Total Tasks", "57" );

datasetB.setValue(total_tasks_per_month[57], "Total Tasks", "58" );

datasetB.setValue(total_tasks_per_month[58], "Total Tasks", "59" );

datasetB.setValue(total_tasks_per_month[59], "Total Tasks", "60" );



JFreeChart chartB = ChartFactory.createBarChart("Total Tasks Per Month", "Month", "Tasks per
Manth", datasetB, PlotOrientation.VERTICAL, false, true, false);


try { ChartUtilities.saveChartAsJPEG(new File("C:\\Total Tasks Per Month.jpg"), chartB, 1500, 700);

} catch (IOException a){

        System.err.println("Problem");

}



DefaultCategoryDataset datasetC = new DefaultCategoryDataset();


datasetC.setValue(tasks_done[0], "Total Tasks", "1" );

datasetC.setValue(tasks_done[1], "Total Tasks", "2" );

datasetC.setValue(tasks_done[2], "Total Tasks", "3" );

datasetC.setValue(tasks_done[3], "Total Tasks", "4" );

datasetC.setValue(tasks_done[4], "Total Tasks", "5" );

datasetC.setValue(tasks_done[5], "Total Tasks", "6" );

datasetC.setValue(tasks_done[6], "Total Tasks", "7" );

datasetC.setValue(tasks_done[7], "Total Tasks", "8" );

datasetC.setValue(tasks_done[8], "Total Tasks", "9" );

datasetC.setValue(tasks_done[9], "Total Tasks", "10" );

datasetC.setValue(tasks_done[10], "Total Tasks", "11" );
datasetC.setValue(tasks_done[11], "Total Tasks", "12" );

datasetC.setValue(tasks_done[12], "Total Tasks", "13" );

datasetC.setValue(tasks_done[13], "Total Tasks", "14" );

datasetC.setValue(tasks_done[14], "Total Tasks", "15" );

datasetC.setValue(tasks_done[15], "Total Tasks", "16" );

datasetC.setValue(tasks_done[16], "Total Tasks", "17" );

datasetC.setValue(tasks_done[17], "Total Tasks", "18" );

datasetC.setValue(tasks_done[18], "Total Tasks", "19" );

datasetC.setValue(tasks_done[19], "Total Tasks", "20" );

datasetC.setValue(tasks_done[20], "Total Tasks", "21" );

datasetC.setValue(tasks_done[21], "Total Tasks", "22" );

datasetC.setValue(tasks_done[22], "Total Tasks", "23" );

datasetC.setValue(tasks_done[23], "Total Tasks", "24" );

datasetC.setValue(tasks_done[24], "Total Tasks", "25" );

datasetC.setValue(tasks_done[25], "Total Tasks", "26" );

datasetC.setValue(tasks_done[26], "Total Tasks", "27" );

datasetC.setValue(tasks_done[27], "Total Tasks", "28" );

datasetC.setValue(tasks_done[28], "Total Tasks", "29" );

datasetC.setValue(tasks_done[29], "Total Tasks", "30" );

datasetC.setValue(tasks_done[30], "Total Tasks", "31" );

datasetC.setValue(tasks_done[31], "Total Tasks", "32" );

datasetC.setValue(tasks_done[32], "Total Tasks", "33" );

datasetC.setValue(tasks_done[33], "Total Tasks", "34" );

datasetC.setValue(tasks_done[34], "Total Tasks", "35" );

datasetC.setValue(tasks_done[35], "Total Tasks", "36" );

datasetC.setValue(tasks_done[36], "Total Tasks", "37" );

datasetC.setValue(tasks_done[37], "Total Tasks", "38" );
datasetC.setValue(tasks_done[38], "Total Tasks", "39" );

datasetC.setValue(tasks_done[39], "Total Tasks", "40" );

datasetC.setValue(tasks_done[40], "Total Tasks", "41" );

datasetC.setValue(tasks_done[41], "Total Tasks", "42" );

datasetC.setValue(tasks_done[42], "Total Tasks", "43" );

datasetC.setValue(tasks_done[43], "Total Tasks", "44" );

datasetC.setValue(tasks_done[44], "Total Tasks", "45" );

datasetC.setValue(tasks_done[45], "Total Tasks", "46" );

datasetC.setValue(tasks_done[46], "Total Tasks", "47" );

datasetC.setValue(tasks_done[47], "Total Tasks", "48" );

datasetC.setValue(tasks_done[48], "Total Tasks", "49" );

datasetC.setValue(tasks_done[49], "Total Tasks", "50" );

datasetC.setValue(tasks_done[50], "Total Tasks", "51" );

datasetC.setValue(tasks_done[51], "Total Tasks", "52" );

datasetC.setValue(tasks_done[52], "Total Tasks", "53" );

datasetC.setValue(tasks_done[53], "Total Tasks", "54" );

datasetC.setValue(tasks_done[54], "Total Tasks", "55" );

datasetC.setValue(tasks_done[55], "Total Tasks", "56" );

datasetC.setValue(tasks_done[56], "Total Tasks", "57" );

datasetC.setValue(tasks_done[57], "Total Tasks", "58" );

datasetC.setValue(tasks_done[58], "Total Tasks", "59" );

datasetC.setValue(tasks_done[59], "Total Tasks", "60" );


JFreeChart chartC = ChartFactory.createBarChart(" Tasks Completed Per Month", "Month", "Tasks
per Manth", datasetC, PlotOrientation.VERTICAL, false, true, false);
```

```
try { ChartUtilities.saveChartAsJPEG(new File("C:\\ Tasks Completed Per Month.jpg"), chartC, 1500, 700);

} catch (IOException a){

	System.err.println("Problem");

}



JFrame frame = new JFrame();

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


Object rowData[][] = { { "Paralegals ", amount_paralegals[0], amount_paralegals[1],amount_paralegals[2],amount_paralegals[3],amount_paralegals[4],amount_paralegals[6],amount_paralegals[7],amount_paralegals[8],amount_paralegals[9],amount_paralegals[10],amount_paralegals[11],amount_paralegals[12],amount_paralegals[13],amount_paralegals[14],amount_paralegals[15],amount_paralegals[16],amount_paralegals[17],amount_paralegals[18],amount_paralegals[19],amount_paralegals[20],amount_paralegals[21],amount_paralegals[22],amount_paralegals[23],amount_paralegals[24],amount_paralegals[25],amount_paralegals[26],amount_paralegals[27],amount_paralegals[28],amount_paralegals[29],amount_paralegals[30],amount_paralegals[31],amount_paralegals[32],amount_paralegals[33],amount_paralegals[34],amount_paralegals[35],amount_paralegals[36],amount_paralegals[37],amount_paralegals[38],amount_paralegals[39],amount_paralegals[40],amount_paralegals[41],amount_paralegals[42],amount_paralegals[43],amount_paralegals[44],amount_paralegals[45],amount_paralegals[46],amount_paralegals[47],amount_paralegals[48],amount_paralegals[49],amount_paralegals[50],amount_paralegals[51],amount_paralegals[52],amount_paralegals[53],amount_paralegals[54],amount_paralegals[55],amount_paralegals[56],amount_paralegals[57],amount_paralegals[58],amount_paralegals[59] },

			{"Transaction Coordinators",transaction_co_s[0], transaction_co_s[1],transaction_co_s[2],transaction_co_s[3],transaction_co_s[4],transaction_co_s[6],transaction_co_s[7],transaction_co_s[8],transaction_co_s[9],transaction_co_s[10],transaction_co_s[11],transaction_co_s[12],transaction_co_s[13],transaction_co_s[14],transaction_co_s[15],transaction_co_s[16],transaction_co_s[17],transaction_co_s[18],transaction_co_s[19],transaction_co_s[20],transaction_co_s[21],transaction_co_s[22],transaction_co_s[23],transaction_co_s[24],transaction_co_s[25],transaction_co_s[26],transaction_co_s[27],transaction_co_s[28],transaction_co_s[29],transaction_co_s[30],transaction_co_s[31],transaction_co_s[32],transaction_co_s[33],transaction_co_s[34],transaction_co_s[35],transaction_co_s[36],transaction_co_s[37],transaction_co_s[38],transaction_co_s[39],transaction_co_s[40],transaction_co_s[41],transaction_co_s[42],transaction_co_s[43],transaction_co_s[44],transaction_co_s[45],transaction_co_s[46],transaction_co_s[47],transaction_co_s[48],transaction_co_s[49],transaction_co_s[50],transaction_co_s[51],transaction_co_s[52],transacti
```

```
on_co_s[53],transaction_co_s[54],transaction_co_s[55],transaction_co_s[56],transaction_co_s[57],t
ransaction_co_s[58],transaction_co_s[59] },

                {"Managing Attorneys ", man_att[0],
man_att[1],man_att[2],man_att[3],man_att[4],man_att[6],man_att[7],man_att[8],man_att[9],man_
att[10],man_att[11],man_att[12],man_att[13],man_att[14],man_att[15],man_att[16],man_att[17],
man_att[18],man_att[19],man_att[20],man_att[21],man_att[22],man_att[23],man_att[24],man_att
[25],man_att[26],man_att[27],man_att[28],man_att[29],man_att[30],man_att[31],man_att[32],man
_att[33],man_att[34],man_att[35],man_att[36],man_att[37],man_att[38],man_att[39],man_att[40],
man_att[41],man_att[42],man_att[43],man_att[44],man_att[45],man_att[46],man_att[47],man_att
[48],man_att[49],man_att[50],man_att[51],man_att[52],man_att[53],man_att[54],man_att[55],man
_att[56],man_att[57],man_att[58],man_att[59]  },

                {"Messengers ", messengers[0],
messengers[1],messengers[2],messengers[3],messengers[4],messengers[6],messengers[7],messeng
ers[8],messengers[9],messengers[10],messengers[11],messengers[12],messengers[13],messengers[
14],messengers[15],messengers[16],messengers[17],messengers[18],messengers[19],messengers[2
0],messengers[21],messengers[22],messengers[23],messengers[24],messengers[25],messengers[26]
,messengers[27],messengers[28],messengers[29],messengers[30],messengers[31],messengers[32],
messengers[33],messengers[34],messengers[35],messengers[36],messengers[37],messengers[38],m
essengers[39],messengers[40],messengers[41],messengers[42],messengers[43],messengers[44],mes
sengers[45],messengers[46],messengers[47],messengers[48],messengers[49],messengers[50],messe
ngers[51],messengers[52],messengers[53],messengers[54],messengers[55],messengers[56],messen
gers[57],messengers[58],messengers[59]  },

                {"Finance Personal ", fin[0],
fin[1],fin[2],fin[3],fin[4],fin[6],fin[7],fin[8],fin[9],fin[10],fin[11],fin[12],fin[13],fin[14],fin[15],fin[16],fi
n[17],fin[18],fin[19],fin[20],fin[21],fin[22],fin[23],fin[24],fin[25],fin[26],fin[27],fin[28],fin[29],fin[30],f
in[31],fin[32],fin[33],fin[34],fin[35],fin[36],fin[37],fin[38],fin[39],fin[40],fin[41],fin[42],fin[43],fin[44],
fin[45],fin[46],fin[47],fin[48],fin[49],fin[50],fin[51],fin[52],fin[53],fin[54],fin[55],fin[56],fin[57],fin[58]
,fin[59]  },

                {"Operations Manager ", ops_man[0],
ops_man[1],ops_man[2],ops_man[3],ops_man[4],ops_man[6],ops_man[7],ops_man[8],ops_man[9],
ops_man[10],ops_man[11],ops_man[12],ops_man[13],ops_man[14],ops_man[15],ops_man[16],ops
_man[17],ops_man[18],ops_man[19],ops_man[20],ops_man[21],ops_man[22],ops_man[23],ops_m
an[24],ops_man[25],ops_man[26],ops_man[27],ops_man[28],ops_man[29],ops_man[30],ops_man[
31],ops_man[32],ops_man[33],ops_man[34],ops_man[35],ops_man[36],ops_man[37],ops_man[38],
ops_man[39],ops_man[40],ops_man[41],ops_man[42],ops_man[43],ops_man[44],ops_man[45],ops
_man[46],ops_man[47],ops_man[48],ops_man[49],ops_man[50],ops_man[51],ops_man[52],ops_m
an[53],ops_man[54],ops_man[55],ops_man[56],ops_man[57],ops_man[58],ops_man[59]  },

                {"Admin Personal ", admin[0],
admin[1],admin[2],admin[3],admin[4],admin[6],admin[7],admin[8],admin[9],admin[10],admin[11],a
dmin[12],admin[13],admin[14],admin[15],admin[16],admin[17],admin[18],admin[19],admin[20],ad
min[21],admin[22],admin[23],admin[24],admin[25],admin[26],admin[27],admin[28],admin[29],admi
n[30],admin[31],admin[32],admin[33],admin[34],admin[35],admin[36],admin[37],admin[38],admin[
39],admin[40],admin[41],admin[42],admin[43],admin[44],admin[45],admin[46],admin[47],admin[48
],admin[49],admin[50],admin[51],admin[52],admin[53],admin[54],admin[55],admin[56],admin[57],a
dmin[58],admin[59]  },

                {"Deeds Office Attorney ", DO_att[0],
DO_att[1],DO_att[2],DO_att[3],DO_att[4],DO_att[6],DO_att[7],DO_att[8],DO_att[9],DO_att[10],DO_
att[11],DO_att[12],DO_att[13],DO_att[14],DO_att[15],DO_att[16],DO_att[17],DO_att[18],DO_att[19
],DO_att[20],DO_att[21],DO_att[22],DO_att[23],DO_att[24],DO_att[25],DO_att[26],DO_att[27],DO_
att[28],DO_att[29],DO_att[30],DO_att[31],DO_att[32],DO_att[33],DO_att[34],DO_att[35],DO_att[36
],DO_att[37],DO_att[38],DO_att[39],DO_att[40],DO_att[41],DO_att[42],DO_att[43],DO_att[44],DO_
att[45],DO_att[46],DO_att[47],DO_att[48],DO_att[49],DO_att[50],DO_att[51],DO_att[52],DO_att[53
],DO_att[54],DO_att[55],DO_att[56],DO_att[57],DO_att[58],DO_att[59]  },

                {"Reception ", Reception[0],
Reception[1],Reception[2],Reception[3],Reception[4],Reception[6],Reception[7],Reception[8],Recep
tion[9],Reception[10],Reception[11],Reception[12],Reception[13],Reception[14],Reception[15],Rece
ption[16],Reception[17],Reception[18],Reception[19],Reception[20],Reception[21],Reception[22],Re
ception[23],Reception[24],Reception[25],Reception[26],Reception[27],Reception[28],Reception[29],
Reception[30],Reception[31],Reception[32],Reception[33],Reception[34],Reception[35],Reception[3
6],Reception[37],Reception[38],Reception[39],Reception[40],Reception[41],Reception[42],Receptio
n[43],Reception[44],Reception[45],Reception[46],Reception[47],Reception[48],Reception[49],Recep
tion[50],Reception[51],Reception[52],Reception[53],Reception[54],Reception[55],Reception[56],Rec
eption[57],Reception[58],Reception[59]  },

                {"Total Employees ", Total_em[0],
Total_em[1],Total_em[2],Total_em[3],Total_em[4],Total_em[6],Total_em[7],Total_em[8],Total_em[
9],Total_em[10],Total_em[11],Total_em[12],Total_em[13],Total_em[14],Total_em[15],Total_em[16]
,Total_em[17],Total_em[18],Total_em[19],Total_em[20],Total_em[21],Total_em[22],Total_em[23],T
otal_em[24],Total_em[25],Total_em[26],Total_em[27],Total_em[28],Total_em[29],Total_em[30],Tot
al_em[31],Total_em[32],Total_em[33],Total_em[34],Total_em[35],Total_em[36],Total_em[37],Total
_em[38],Total_em[39],Total_em[40],Total_em[41],Total_em[42],Total_em[43],Total_em[44],Total_
em[45],Total_em[46],Total_em[47],Total_em[48],Total_em[49],Total_em[50],Total_em[51],Total_e
m[52],Total_em[53],Total_em[54],Total_em[55],Total_em[56],Total_em[57],Total_em[58],Total_em
[59]  },

                {"Total Capital before B-E ",(int)capital_true , " "," "," "," "," "," "," "," "," "," "," "," ","
"," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," ","
"," "," "," "," "," "," "," "," "," "," "," "," "," "  },

                {"Break Even Point ",break_even_point , " "," "," "," "," "," "," "," "," "," "," "," "," ","
"," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," ","
"," "," "," "," "," "," "," "," "," "," "," "," "  },

                {"After Tax Projection ", after_tax_profit_projection[0],
after_tax_profit_projection[1],after_tax_profit_projection[2],after_tax_profit_projection[3],after_ta
x_profit_projection[4],after_tax_profit_projection[6],after_tax_profit_projection[7],after_tax_profit
_projection[8],after_tax_profit_projection[9],after_tax_profit_projection[10],after_tax_profit_proje
```

```
ction[11],after_tax_profit_projection[12],after_tax_profit_projection[13],after_tax_profit_projectio
n[14],after_tax_profit_projection[15],after_tax_profit_projection[16],after_tax_profit_projection[17
],after_tax_profit_projection[18],after_tax_profit_projection[19],after_tax_profit_projection[20],aft
er_tax_profit_projection[21],after_tax_profit_projection[22],after_tax_profit_projection[23],after_t
ax_profit_projection[24],after_tax_profit_projection[25],after_tax_profit_projection[26],after_tax_
profit_projection[27],after_tax_profit_projection[28],after_tax_profit_projection[29],after_tax_prof
it_projection[30],after_tax_profit_projection[31],after_tax_profit_projection[32],after_tax_profit_p
rojection[33],after_tax_profit_projection[34],after_tax_profit_projection[35],after_tax_profit_proje
ction[36],after_tax_profit_projection[37],after_tax_profit_projection[38],after_tax_profit_projectio
n[39],after_tax_profit_projection[40],after_tax_profit_projection[41],after_tax_profit_projection[42
],after_tax_profit_projection[43],after_tax_profit_projection[44],after_tax_profit_projection[45],aft
er_tax_profit_projection[46],after_tax_profit_projection[47],after_tax_profit_projection[48],after_t
ax_profit_projection[49],after_tax_profit_projection[50],after_tax_profit_projection[51],after_tax_
profit_projection[52],after_tax_profit_projection[53],after_tax_profit_projection[54],after_tax_prof
it_projection[55],after_tax_profit_projection[56],after_tax_profit_projection[57],after_tax_profit_p
rojection[58],after_tax_profit_projection[59]  },

                    {"Future Value ", (int)future_value[0],
(int)future_value[1],(int)future_value[2],(int)future_value[3],(int)future_value[4],(int)future_value[6
],(int)future_value[7],(int)future_value[8],(int)future_value[9],(int)future_value[10],(int)future_valu
e[11],(int)future_value[12],(int)future_value[13],(int)future_value[14],(int)future_value[15],(int)futu
re_value[16],(int)future_value[17],(int)future_value[18],(int)future_value[19],(int)future_value[20],(
int)future_value[21],(int)future_value[22],(int)future_value[23],(int)future_value[24],(int)future_val
ue[25],(int)future_value[26],(int)future_value[27],(int)future_value[28],(int)future_value[29],(int)fut
ure_value[30],(int)future_value[31],(int)future_value[32],(int)future_value[33],(int)future_value[34]
,(int)future_value[35],(int)future_value[36],(int)future_value[37],(int)future_value[38],(int)future_v
alue[39],(int)future_value[40],(int)future_value[41],(int)future_value[42],(int)future_value[43],(int)f
uture_value[44],(int)future_value[45],(int)future_value[46],(int)future_value[47],(int)future_value[4
8],(int)future_value[49],(int)future_value[50],(int)future_value[51],(int)future_value[52],(int)future_
value[53],(int)future_value[54],(int)future_value[55],(int)future_value[56],(int)future_value[57],(int)
future_value[58],(int)future_value[59]  },



};

Object columnNames[] = { "Months", "1",
"2","3","4","5","6","7","8","9","10","11","12","13","14","15","16","17","18","19","20","21","22","23
","24","25","26","27","28","29","30","31","32","33","34","35","36","37","38","39","40","41","42","4
3","44","45","46","47","48","49","50","51","52","53","54","56","57","58","59","60"};

JTable table = new JTable(rowData, columnNames);


JScrollPane scrollPane = new JScrollPane(table);
```

```
frame.add(scrollPane, BorderLayout.CENTER);

frame.setSize(1600, 300);

frame.setVisible(true);


}

}



public static void main(String[] args){

ProjectFinal frame = new ProjectFinal();

frame.pack();

frame.setTitle("DWR Simulation");

frame.setLocationRelativeTo(null);

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

frame.setVisible(true);


}


public class ExcelAdapter implements ActionListener

  {

  private String rowstring,value;

  private Clipboard system;

  private StringSelection stsel;

  private JTable jTable1 ;

  /**

   * The Excel Adapter is constructed with a

   * JTable on which it enables Copy-Paste and acts
   * as a Clipboard listener.
   */
public ExcelAdapter(JTable myJTable)
  {
     jTable1 = myJTable;
     KeyStroke copy = KeyStroke.getKeyStroke(KeyEvent.VK_C,ActionEvent.CTRL_MASK,false);
     // Identifying the copy KeyStroke user can modify this
     // to copy on some other Key combination.
     KeyStroke paste = KeyStroke.getKeyStroke(KeyEvent.VK_V,ActionEvent.CTRL_MASK,false);
     // Identifying the Paste KeyStroke user can modify this
     //to copy on some other Key combination.
jTable1.registerKeyboardAction(this,"Copy",copy,JComponent.WHEN_FOCUSED);
jTable1.registerKeyboardAction(this,"Paste",paste,JComponent.WHEN_FOCUSED);
     system = Toolkit.getDefaultToolkit().getSystemClipboard();
  }
  /**
   * Public Accessor methods for the Table on which this adapter acts.
   */
public JTable getJTable() {return jTable1;}
public void setJTable(JTable jTable1) {this.jTable1=jTable1;}
  /**
   * This method is activated on the Keystrokes we are listening to
   * in this implementation. Here it listens for Copy and Paste ActionCommands.
   * Selections comprising non-adjacent cells result in invalid selection and
   * then copy action cannot be performed.
   * Paste is done by aligning the upper left corner of the selection with the
   * 1st element in the current selection of the JTable.
```

```
    */
public void actionPerformed(ActionEvent e)
  {
    if (e.getActionCommand().compareTo("Copy")==0)
    {
      StringBuffer sbf=new StringBuffer();
      // Check to ensure we have selected only a contiguous block of
      // cells
      int numcols=jTable1.getSelectedColumnCount();
      int numrows=jTable1.getSelectedRowCount();
      int[] rowsselected=jTable1.getSelectedRows();
      int[] colsselected=jTable1.getSelectedColumns();
      if (!((numrows-1==rowsselected[rowsselected.length-1]-rowsselected[0] &&
           numrows==rowsselected.length) &&
(numcols-1==colsselected[colsselected.length-1]-colsselected[0] &&
           numcols==colsselected.length)))
      {
        JOptionPane.showMessageDialog(null, "Invalid Copy Selection",
                                "Invalid Copy Selection",
                                JOptionPane.ERROR_MESSAGE);
        return;
      }
      for (int i=0;i<numrows;i++)
      {
        for (int j=0;j<numcols;j++)
        {
sbf.append(jTable1.getValueAt(rowsselected[i],colsselected[j]));
```

```
        if (j<numcols-1) sbf.append("\t");
      }
      sbf.append("\n");
    }
    stsel  = new StringSelection(sbf.toString());
    system = Toolkit.getDefaultToolkit().getSystemClipboard();
    system.setContents(stsel,stsel);
  }
  if (e.getActionCommand().compareTo("Paste")==0)
  {
     System.out.println("Trying to Paste");
     int startRow=(jTable1.getSelectedRows())[0];
     int startCol=(jTable1.getSelectedColumns())[0];
     try
     {
        String trstring= (String)(system.getContents(this).getTransferData(DataFlavor.stringFlavor));
        System.out.println("String is:"+trstring);
        StringTokenizer st1=new StringTokenizer(trstring,"\n");
        for(int i=0;st1.hasMoreTokens();i++)
        {
           rowstring=st1.nextToken();
           StringTokenizer st2=new StringTokenizer(rowstring,"\t");
           for(int j=0;st2.hasMoreTokens();j++)
           {
              value=(String)st2.nextToken();
              if (startRow+i< jTable1.getRowCount()  &&
                  startCol+j< jTable1.getColumnCount())
```

```
                jTable1.setValueAt(value,startRow+i,startCol+j);
              System.out.println("Putting "+ value+"atrow="+startRow+i+"column="+startCol+j);
            }
          }
        }
        catch(Exception ex){ex.printStackTrace();}
      }
    }
  }
  }
```