## ASSUMPTIONS:

1. On the Tax Table I assume the input date will always fall within the range of the tax table. (01 July 2012 - 31 June 2013). The Tax table cannot have a null start or end date. The start and end dates must be valid according to the culture info used.
2. I assume that an Annual Salary cannot be less than the specified Minimum Annual salary, therefore I don’t allow for it.
3. I assume that a Super Rate must have a minimum and maximum allowed input.
4. The example input start date starts with day and then month, Therefore I assume the date input will always start with day followed by month, Even though the metric date format of YYYY-MM-DD is also accepted in New Zealand, the input dates in the examples never pass in the year and based on the assumption from point 1 above I assume it never will.
5. I assume that the Super Rate specified for each employee will always be in a numerical or percentage format. When the rate is specified as a percentage string ex. “10%”, The % sign gets removed and the super gets converted to a numerical value.
6. I assume that the data input will always be static; the application was written to read the input from a pre-determined path and does not refresh the data.
7. I assume the following data types in the input tables.
   1. TaxBrackets
      1. Min decimal
      2. Max decimal
      3. Tax decimal
      4. AdditionalCharge decimal
   2. TaxTable
      1. TaxPeriodStartDate datetime
      2. TaxPeriodEndDate datetime
   3. PaySlipEntry
      1. FirstName string
      2. LastName string
      3. AnnualSalary decimal
      4. SupperRate decimal
      5. PaymentStartDate datetime
   4. PaySlipTable
      1. MinAnnualSalary decimal
      2. MinSuperRate decimal
      3. MaxSuperRate decimal

## DECISIONS MADE:

1. Because the data input is static and the path to the files won’t change, I created a static class and defined the names of the various files used by the application. The class will not be instantiated so will not get any memory allocated to it at runtime.
2. I created a static class for all the display formatting. The properties and methods of this class cannot be changed at runtime and is not dependant on instance data from any class.
3. Reading the content of files, saving content to files and deleting files are static methods that always perform the same. These methods don’t have any state and gets disposed as soon as the method is complete. I therefore created a static Utilities class that holds these static IO methods.
4. I created a static DataObject class that loads an interface of Application objects. The DataObject class has no properties and cannot be instantiated.
5. The MonthPay class has 2 private methods that perform calculations based on input passed to the method. These methods do not access instance data and simply return calculations. The compiler ignores non virtual call sites to these methods and recommended that these methods get marked as static.
6. I implemented an IApplicationDataCollection Interface with a ValidateInputValues on the DataInput Classes. This allows each Datainput class to validate its expected input as per the assumptions made in the assumptions section above.

## DEPENDENCIES:

1. The console application depends on two input files. These files must exist for the application to execute successfully. In the console application's app.settings file there is a "DataFilesPath" setting that defines the directory to the input files. The "DataFilesPath" setting will be used for the full path to the data files.
2. The two input files are:
   1. TaxTable File: The TaxTable.json file defines the tax brackets used in the tax table.
   2. PaySlipTable File: PaySlipTable.json file is the employee input file for the payslip generation.

## THE SOLUTION:

1. I included the input datafiles mentioned in the dependencies above in the solution and marked them to be copied to the “bin\DataFiles” directory when the solution gets built. This was done for ease of execution.
2. I included the required Test framework execution folder “TestWindow” in the solution and also marked these files to be copied to the “bin\TestWindow” folder when the solution builds. These files are needed to execute the tests outside of solution.

Under normal circumstances I would not include this as part of the solution as I assume the build server will already have the necessary files to run the tests. I included tis because the requirement asked for a test harness to validate the solution

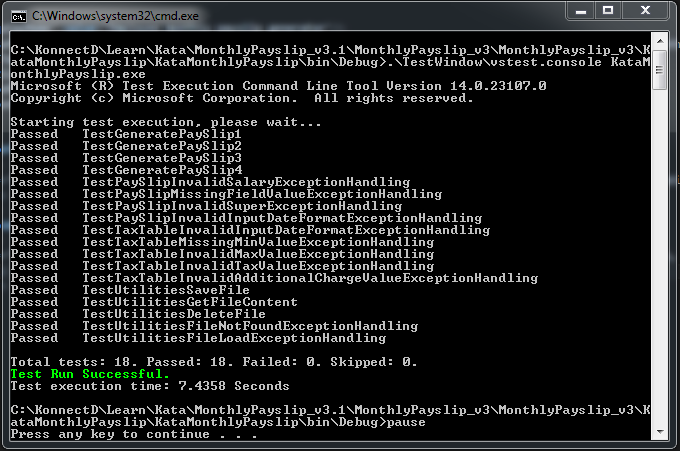
1. Finally I included a “RunTests.bat” file as part of the solution and marked it to get copied to the bin directory when the solution builds. This allows you to run the tests externally simply by calling the batch file in the bin directory.

## RUNNING APPLICATION TESTS:

The unit tests where built using the Microsoft.VisualStudio.QualityTools.UnitTestFramework v2.0.50727 and will require the .NET framework to be installed on the executing machine.

1. Open the directory where the files have been extracted to and browse to the solution sub directory.
2. Open the solution file and build the solution. A successful build will create a RunTests.bat file which can be used to run the tests outside of visual studio. This file is dependent on the TestWindow directory inside the bin directory as it uses the vstest.console.exe to run the tests.
3. Run all the tests.
4. Browse to the \KataMonthlyPayslip\bin\Debug\ sub directory.
5. Double click the RunTests.bat file.

Execution of the test script should produce the following output:



## RUNNING THE APPLICATION:

1. The application was built using .NET Framework 4.6 and executing the application will require the framework to be installed on the executing machine.
2. The application can be launched from either command line executing the Kata\_MonthlyPayslip.exe file or by double clicking the Kata\_MonthlyPayslip.exe file.

The application will display the information listed in point 1 and 2 below before prompting the user for input 3.

|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

| |

| Employee monthly payslip generator |

| |

|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

1. The tax table and payslip input data files exists and can be found in the

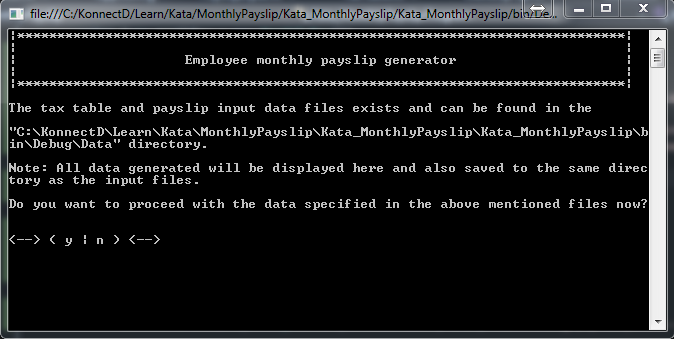
"C:\KonnectD\Learn\Kata\MonthlyPayslip\Kata\_MonthlyPayslip\Kata\_MonthlyPayslip\bin\Debug\Data" directory.

1. Note: All data generated will be displayed here and also saved to the same directory as the input files.
2. Do you want to proceed with the data specified in the above mentioned files now?

<--> ( y | n ) <-->

1. PROMPT: If the files exist the user will be prompted to continue,

*Else the user gets notified of the failure and the application will close.*



* 1. Generation starts.

*Generating pay slips.........*

* 1. Each entry in the input file will produce a line like the following.

[

{"Name":"David Rudd","PayPeriod":"01 March - 31 March","GrossIncome":"5004","IncomeTax":"922","NetIncome":"4082","Super":"450"},

{"Name":"Ryan Chen","PayPeriod":"01 March - 31 March","GrossIncome":"10000","IncomeTax":"2696","NetIncome":"7304","Super":"1000"},

{"Name":"Nico Swanepoel","PayPeriod":"01 September - 30 September","GrossIncome":"7083","IncomeTax":"1616","NetIncome":"5467","Super":"1487"},

{"Name":"Lizzy Fitgerald","PayPeriod":"01 April - 30 April","GrossIncome":"1500","IncomeTax":"0","NetIncome":"1500","Super":"75"}

]

* 1. Once the generation is complete and file has been saved the user will see the below output.

*Payslip generation complete.*

If the file could not be saved, the below message will be displayed.

*Payslip generation complete. Error trying to save the file.*

* 1. Exits the application

*Press any key to exit...*

