Task 3

I opted for a modular approach towards designing this program. Specifically, I wanted a program that was clearly segmented into different methods that interacted with each other, allowing for the program to be run in its entirety from just a few lines within the Main() method. I also made good use of the PDL (as devised in Task 1) to help me structure the architecture of the program and make decisions about what to implement next. I also made some methods private in order to avoid using methods outside of their intended scope. Finally, I added descriptions in the form of comments throughout the files to allow for ease of understanding. The end product is a simple program that takes a .csv file, prompts the user for inputs, and returns a .txt file containing key metrics related to hours worked per department.

The program is rich in variables of all kinds, including strings, integers, arrays, doubles, and lists. For example, the Main() method has one vitally important string variable for the file path and two equally important array variables for the .csv file lines and employee array (string and class object arrays, respectively). A list array was also vital in the construction of the GetDepartment() function, as it permits the program to check if a userInput string is an element of the validOptions list via the .Contains() method. Integer and double variables were crucial to the creation of the calculation methods provided in the Employee class. Since the employees array object declared in the Main() method consisted of string values, adding the weekday hours per employee would have resulted in string concatenation. Therefore, the CalculateTotalDepartmentHours() method required the conversion of these values from strings into integers in order to correctly determine the total hours worked per department. In hindsight, considering the possibility of using decimals to represent half an hour (e.g., 7.5 to represent 7 and a half hours), it may be better to return a double or other decimal-permitting value type. However, since the .csv file provided only included integers for hours, I opted to keep these values for this function. Finally, the use of double variables was instrumental in the creation of the CalculateAverageDepartmenthours() method. Since there is the possibility of a decimal value being returned from averaging the work hours of a given department, the method needs to return a double value; otherwise, it would return a value that is inaccurate.

For loops and while loops were used aplenty in this program. Much of the core functionality relies on these structures, especially with regards obtaining user input. For example, the StartUpProcess() method heavily relies on a while loop to check for incorrect user input. It also prompts the user to place a .csv file into the resources folder should they input “n” or “no” for the prompt (although adding a .csv file to the folder while the program is running would unfortunately not do anything to change the outcome, it still serves as a check to remind the user to add an appropriate file). With regards for loops, they were instrumental in iterating over the length of the employees array for all the calculation methods.

Conditional statements are also quite numerous in this program. If and else if statements are crucial in the CreateTextFiles() method, which prompts the user to answer with either “y” or “n” to create other text files. If “y” or “yes”, the program re-runs, allowing the user to input another department into the console and creates the associated TXT file. If “n” or “no”, the program closes.

Finally, collections in the form of lists were crucial in the creation of this program. They were particularly important in establishing the GetEmployeesWithMostHours() method, where three lists were created to house hours worked, employee names, and top employees (hoursList, employeeNames, and topEmployees, respectively). The non-rigid, mutable characteristics of these lists were instrumental in being able to hold important information and add to it. Moreover, these lists could be affixed with necessary methods such as .Max() to determine the employee with the highest number of hours worked per department.

Thus concludes my report on the functionality of the code. For more details on the precise functioning of the methods, please refer to the comments in the Program.cs and Employee.cs files.