Motivation:

God Class is a class that has so many codes in a single class. Lots of methods and larger methods will be in god class. In this example parameters often used together. Methods use only a small subset of the fields in the class. They might move into a class containing just those field. Lots of code with lots of responsibility in one place. So, need to refactor this class by moving parameters to a class. Methods can be moved to another class. Distributed the responsibility.

**Before:**

class Program

{

static void Main(string[] args)

{

string w, rawTimeWorked;

int i;

double ttl, t;

List<TimeSheetEntry> ents = new List<TimeSheetEntry>();

Console.Write("Enter what you did:");

w = Console.ReadLine();

Console.Write("How long did you do it for:");

rawTimeWorked = Console.ReadLine();

while (double.TryParse(rawTimeWorked, out t) == false)

{

Console.WriteLine();

Console.WriteLine("Invalid number given.");

Console.Write("How long did you do it for:");

rawTimeWorked = Console.ReadLine();

}

TimeSheetEntry ent = new TimeSheetEntry();

ent.HoursWorked = t;

ent.WorkDone = w;

ents.Add(ent);

Console.Write("Do you want to enter more item (y/n): ");

string answer = Console.ReadLine();

bool cont = false;

if (answer.ToLower() == "yes")

cont = true;

while (cont == true)

{

Console.Write("Enter what you did: ");

w = Console.ReadLine();

Console.Write("How long did you do it for: ");

rawTimeWorked = Console.ReadLine();

while (double.TryParse(rawTimeWorked, out t) == false)

{

Console.WriteLine();

Console.WriteLine("Invalid number given.");

Console.Write("How long did you do it for:");

rawTimeWorked = Console.ReadLine();

}

ent.HoursWorked = t;

ent.WorkDone = t;

ents.Add(ent);

Console.Write("Do you want to enter more item (y/n): ");

answer = Console.ReadLine();

cont = false;

if (answer.ToLower() == "yes")

{

cont = true;

}

}

ttl = 0;

for (i = 0; i < ents.Count; i++)

{

if (ents[i].WorkDone.ToLower().Contains("acme"))

{

ttl += ents[i].HoursWorked;

}

}

Console.WriteLine("Simulating sending mail to acme:");

Console.WriteLine("Your bill is &" + ttl \* 150 + "for the hours worked.");

ttl = 0;

for (i = 0; i < ents.Count; i++)

{

if (ents[i].WorkDone.ToLower().Contains("abc"))

{

ttl += ents[i].HoursWorked;

}

}

Console.WriteLine("Simulating sending mail to abc:");

Console.WriteLine("Your bill is &" + ttl \* 150 + "for the hours worked.");

ttl = 0;

for (i = 0; i < ents.Count; i++)

{

ttl += ents[i].HoursWorked;

}

if (ttl > 40)

{

Console.WriteLine("You will get paid $" + (((ttl - 40) \* 15) + (40 \* 10)) + "for the hours work.");

}

else

{

Console.WriteLine("You will get paid $" + (ttl \* 10) + "for your time.");

}

Console.WriteLine();

Console.Write("Press any key to exit...............");

Console.ReadKey();

}

}

public class TimeSheetEntry

{

public double HoursWorked;

public string WorkDone; }

**Mechanics:**

Here you can see that class contain lots of bad variable names, lots of duplicate code, same parameters used multiple times. So, to refactor this code I am going to create a method called LoadTimeSheets(). Moving customer information to CustomerModel class, Employee information to EmployeeModel Class, Timesheet information to TimeSheetEntry class. There are some codes for customer’s billing information and employee payment information. So, remove the duplicate code and put those code into PayEmployee and BillCustomers methods.

**After:**

using GodClassLibrary;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using GodClassLibrary.Model;

namespace GodClass

{

class ProgramA

{

static void Main(string[] args)

{

List<TimeSheetEntryAModel> timeSheets = LoadTimeSheets(); // Load employees timesheet

List<CustomerModel> customers = DataAccess.GetCustomer(); // Load Customer's information

EmployeeModel currentEmployee = DataAccess.GetCurrentEmployee(); // Load Employee's information

customers.ForEach(x => BillCustomers(timeSheets, x));

PayEmployee(timeSheets, currentEmployee);

Console.WriteLine();

Console.Write("Press any key to exit...............");

Console.ReadKey();

}

/// <summary>

/// This method calculate the employee payment

/// </summary>

/// <param name="timeSheets"></param>

/// <param name="employee"></param>

public static void PayEmployee(List<TimeSheetEntryAModel> timeSheets, EmployeeModel employee)

{

decimal totalPay = TimeSheetProcessor.calculateEmployeePay(timeSheets, employee);

Console.WriteLine($"You will get paid${totalPay} for your time.");

}

/// <summary>

/// This method calculating the customer's bill

/// </summary>

/// <param name="timeSheets"></param>

/// <param name="customer"></param>

private static void BillCustomers(List<TimeSheetEntryAModel> timeSheets, CustomerModel customer)

{

double totalHours = TimeSheetProcessor.GetHoursWorkForComapany(timeSheets, customer.CustomerName);

Console.WriteLine($"Simulating sending mail to :{customer.CustomerName}");

Console.WriteLine("Your bill is &" + (decimal)totalHours \* customer.hourlyRateToBill + "for the hours worked.");

Console.WriteLine();

}

/// <summary>

/// Load employees work time

/// </summary>

/// <returns></returns>

private static List<TimeSheetEntryAModel> LoadTimeSheets()

{

List<TimeSheetEntryAModel> output = new List<TimeSheetEntryAModel>();

string entermoreTimeSheet = "";

do

{

Console.Write("Enter what you did: ");

string workDone = Console.ReadLine();

Console.Write("How long did you do it for: ");

string rawTimeWorked = Console.ReadLine();

double hoursworked;

while (double.TryParse(rawTimeWorked, out hoursworked) == false)

{

Console.WriteLine();

Console.WriteLine("Invalid number given.");

Console.Write("How long did you do it for:");

rawTimeWorked = Console.ReadLine();

}

TimeSheetEntryAModel timeSheet = new TimeSheetEntryAModel()

{

HoursWorked = hoursworked,

WorkDone = workDone

};

output.Add(timeSheet);

Console.Write("Do you want to enter more item (y/n): ");

entermoreTimeSheet = Console.ReadLine();

} while (entermoreTimeSheet.ToLower() == "yes");

Console.WriteLine();

return output;

}

}

}