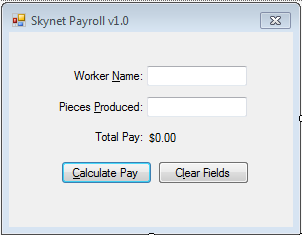
NETD 3202 – Lab 1

# The Scenario

As outlined in the background from Incorporeal Incorporated, their employees are paid to send text messages (usually filled with lies) to their clients. Although employees have yet to be paid, the company has come up with a piecework payroll plan to pay employees… it just hasn’t been implemented yet.

Part of an application has been inherited from another corporation that Incorporeal Incorporated has insisted they own the rights to adapt for commercial purposes. This includes the screenshot of a Windows form as shown below and a partly complete VB class file that has been provided. Victor from Incorporeal Incorporated has asked us to investigate WPF for improved scaling at high resolutions (he insists this will be useful as more features are added). We will be creating a new form similar to the old one and adapting the provided class, as well as making several changes. All currency will be in dollars.

# The Existing Form



This form is designed to receive a worker’s name and a number of pieces produced. When the Calculate Pay button is clicked, the textbox currently labelled “Pieces Produced” should be validated to check for a positive whole number. Depending on the number provided, a worker’s pay is calculated. This pay is then displayed in a label shown as “Total Pay”.

When the Clear Fields button is clicked, both text boxes and the label containing the resulting pay are cleared, and focus is set to the field for entering the worker’s name.

# Required Changes for the Form Design

* First, the form must be entirely re-created as a WPF Application using C# or VB
* Change the title – ensure there is absolutely no mention of Skynet; consider using Incorporeal Incorporated or IncInc instead
* IncInc’s application is designed to pay workers for sending text messages; it should read “Messages Sent” instead of “Pieces Produced”
* Add the IncInc logo – or something else suitable – as an icon

# Required Changes for the Worker Class

* PieceworkWorker.txt has been provided in the Lab 1 assignment folder - it has three sections indicated with block comments that start with “TO DO” and these parts must be completed
* Finish the findPay() method such that a worker’s calculated pay is based on the following table, **rounded to the nearest cent**:

|  |  |
| --- | --- |
| Messages Sent | Price Paid Per Message for All Messages |
| 1 – 2499 | 0.022 |
| 2500 – 4999 | 0.024 |
| 5000 – 7499 | 0.027 |
| 7500 – 10000 | 0.031 |
| 10000+ | 0.035 |

* In the property Set procedures for the worker’s name, code validation ensuring that a name has been entered for the worker (it does not matter what is entered as long as it isn’t blank)
* In the property Set procedure for the number of messages sent, code validation to ensure that the messages are entered as a whole number greater than zero
* Validation messages for the name and number of messages should be provided to the user with a message box or using an error provider – do not put the error messages in a label on the form

# If You Choose to Code Your Piecework Worker Class From Scratch

If you instead **choose** to start from scratch, ensure that your worker class includes all of the following:

* + a parametrized constructor and a default constructor
  + instance variables to hold the worker’s name and number of sent messages
  + shared variables to hold running totals for the number of employees processed, total of all calculated pay, and total number of messages sent
  + read/write property procedures for the worker’s name and number of messages
  + a read-only property procedure to access the worker’s calculated pay
  + shared, read-only property procedures to access the summary data, including the number of employees processed, total of all calculated pay, and total number of messages sent
  + a findPay() method to calculate a worker’s pay, which also updates the summary values above

This list may also be useful to any student who wants to ensure their worker class is complete.

# General Function

* When the Calculate Pay button is clicked, the textbox currently labelled “Pieces Produced” should be validated to check for a positive whole number. Depending on the number provided, a worker’s pay is calculated as shown in the table above
* When a user clicks Calculate, the program attempts to create a new piecework worker object. The validation of this worker will take place within the worker class
* If the worker’s name is not entered, a descriptive validation message will appear (in a message box or error provider) and the pay will not calculate
* If the messages sent is not entered as a positive integer, a descriptive validation message will appear (in a message box or error provider) and the pay will not calculate
* If everything validates properly, the worker’s calculated pay will be calculated (in the class / business logic) and displayed in an appropriate label (in the form / presentation logic), similarly to the screenshot above.
* When the worker’s pay is entered, the entry fields and Calculate button should be disabled / greyed out and focus set to the Clear button
* When the Clear button is pressed, both text boxes and the label containing the resulting pay are cleared, the textboxes and Calculate button are re-enabled, and focus is set to the field for entering the worker’s name

# Additional Considerations

* Pay attention to when your labels are cleared, and ensure no incorrect information is ever on-screen
* The program must be adequately documented:
  + functions and event handlers should all have block comments
  + calculations, decisions and iteration should be explained with brief comments
  + there should be a header at the top of each form’s code (and any other modules) including your name, the last modified date, and a description
* Adhere to an approved style guide and ensure your variable names and other elements are properly cased and adequately descriptive
* Consider accessibility; this application must be easily usable with keyboard only and any fields the user interacts with must have useful tooltips

# Assessment

Each student’s work will be submitted individually and assessed by the instructor using the NETD 3202 Lab Rubric available on DC Connect. Reasonably detailed feedback will be provided.

# Enhancements for Stock

* 2 bonus stock if submitted with a useful planning document, including (at a minimum) reasonably complete pseudocode and/or a flowchart
* 2 bonus stock for validating that the entered name contains at least 2 alphabetic characters
* 2 bonus stock if the payment plan is implemented using two single-dimensional arrays with four elements each using either a For loop or a Select Case
* 1 bonus stock if your form also includes a functional Exit button
* 1 bonus stock if a block comment is included at the very bottom of your worker class code file that briefly answers the question, “Why are some property procedures in this class read/write while others are read-only?”, including reference to at least one example of each from this lab
* Stock bonuses will be reduced by 1 if any added functionality allows the program to crash