

Lab 07: Handling Exceptions

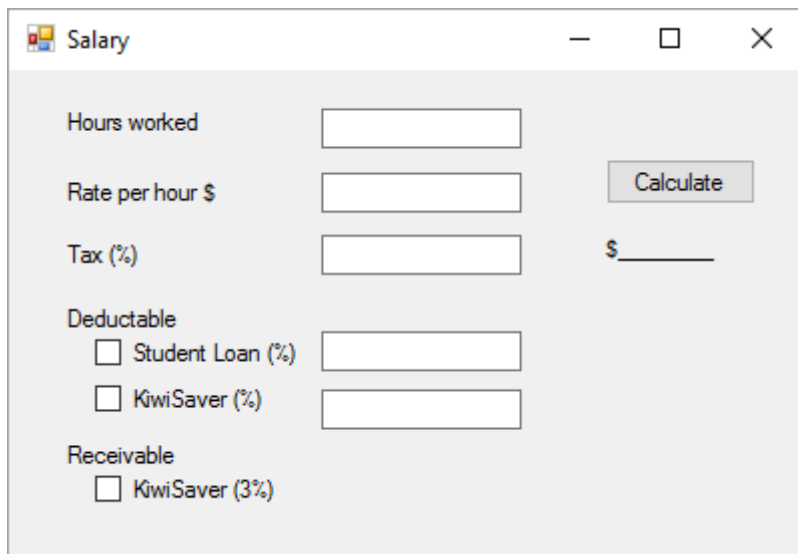
When an application needs to interact with user inputs and peripheral equipment, it poses risk of generating an exception (run-time error). When this occurs, the application may stop abruptly when exception is unhandled. An exception can be arrested when it occurs, giving the application a chance to perform the next line (or a specific section) of code.

An implementation of this may be in form of displaying an error message, rollback of transactions, undo of changes, or simple ignore and more to next execution.

Using traditional techniques to manually add error-detecting code around every statement is cumbersome, time consuming, and error prone in its own right.

In this exercise, we will use *Try...Catch...Finally* technique to handle an exception. We will reuse the exercise from previous lab, and look at how this technique of handling an exception differs from the traditional technique used there.

Please refer to lab exercise with the following sample interface.



The screenshot shows a window titled "Salary" with a standard Windows title bar (minimize, maximize, close buttons). The window contains a form with the following elements:

- Hours worked:** A text input field.
- Rate per hour \$:** A text input field.
- Tax (%):** A text input field.
- Deductable:** A section header followed by two checkboxes:
 - ☐ Student Loan (%): A text input field.
 - ☐ KiwiSaver (%): A text input field.
- Receivable:** A section header followed by one checkbox:
 - ☐ KiwiSaver (3%): A text input field.
- Calculate:** A button located to the right of the "Rate per hour \$" field.
- \$:** A text input field located below the "Calculate" button.

Error-detecting using traditional technique

Get the inputs from user and store the values into corresponding variables, using the built-in syntax of *.TryParse*.

```
//There are 2 common ways to get input from a textbox (string type), and _  
//convert it to double type. They are Convert.ToDouble() and double.TryParse  
//double.TryParse will automatically set the converted value to zero if it's an incorrect type  
//The format is double.TryParse (Input, out Output)  
double.TryParse(txtKiwiSaverEmployeeRate.Text, out d_KiwiSaverEmployeeRate);  
double.TryParse(txtHoursWorked.Text, out d_HourWorked);  
double.TryParse(txtWageRate.Text, out d_PayRate);  
double.TryParse(txtTaxRate.Text, out d_TaxRate);  
double.TryParse(txtLoanRate.Text, out d_LoanRate);
```

We can simplify this by adding them into an error-detecting block.

Error-detecting using *Try...Catch...Finally* technique

```
private void GetDataFromUser()  
{  
    try          //This Try block is where normal codes for execution are placed  
    {  
        d_KiwiSaverEmployeeRate = Convert.ToDouble(txtKiwiSaverEmployeeRate.Text);  
        d_HourWorked = Convert.ToDouble(txtHoursWorked.Text);  
        d_PayRate = Convert.ToDouble(txtWageRate.Text);  
        d_TaxRate = Convert.ToDouble(txtTaxRate.Text);  
        d_LoanRate = Convert.ToDouble(txtLoanRate.Text);  
    }  
    catch        //Codes in this catch block will be executed when an error occurs  
    {  
        MessageBox.Show("Error: Invalid input. Please enter numeric values only.");  
    }  
    finally      //This block is optional. Codes written here will execute whether or not error happens  
    {  
        //This is useful in situation where certain codes must be executed regardless  
        //Example: to write a log entry, etc.  
    }  
    //When an error occurs, this is the end of execution  
    //Subsequent codes will not be executed  
}
```

Full codes

```

public partial class frmSalary : Form
{
    1 reference
    public frmSalary()...

    //Declare the required variables for storing inputs from user
    //Use double type, to allow 2-digit decimal point, e.g. 120.45
    double d_HourWorked, d_PayRate, d_TaxRate, d_LoanRate, d_KiwiSaverEmployeeRate, d_KiwiSaverEmployerRate = 3 / 100;

    //Also declare variables needed for calculation
    double d_GrossPay, d_Loan, d_Tax, d_KiwiSaverEmployee, d_KiwiSaverEmployer, d_NetPay;

    1 reference
    private void GetDataFromUser()
    {
        try          //This Try block is where normal codes for execution are placed
        {
            d_KiwiSaverEmployeeRate = Convert.ToDouble(txtKiwiSaverEmployeeRate.Text);
            d_HourWorked = Convert.ToDouble(txtHoursWorked.Text);
            d_PayRate = Convert.ToDouble(txtWageRate.Text);
            d_TaxRate = Convert.ToDouble(txtTaxRate.Text);
            d_LoanRate = Convert.ToDouble(txtLoanRate.Text);
        }
        catch        //Codes in this catch block will be executed when an error occurs
        {
            MessageBox.Show("Error: Invalid input. Please enter numeric values only.");
        }
        finally      //This block is optional. Codes written here will execute whether or not error happens
        {
            //This is useful in situation where certain codes must be executed regardless
            //Example: to write a log entry, etc.
        }
        //When an error occurs, this is the end of execution
        //Subsequent codes will not be executed
    }

    1 reference
    private void CalcNetPay()...

    1 reference
    private void btnCalc_Click(object sender, EventArgs e)...
```