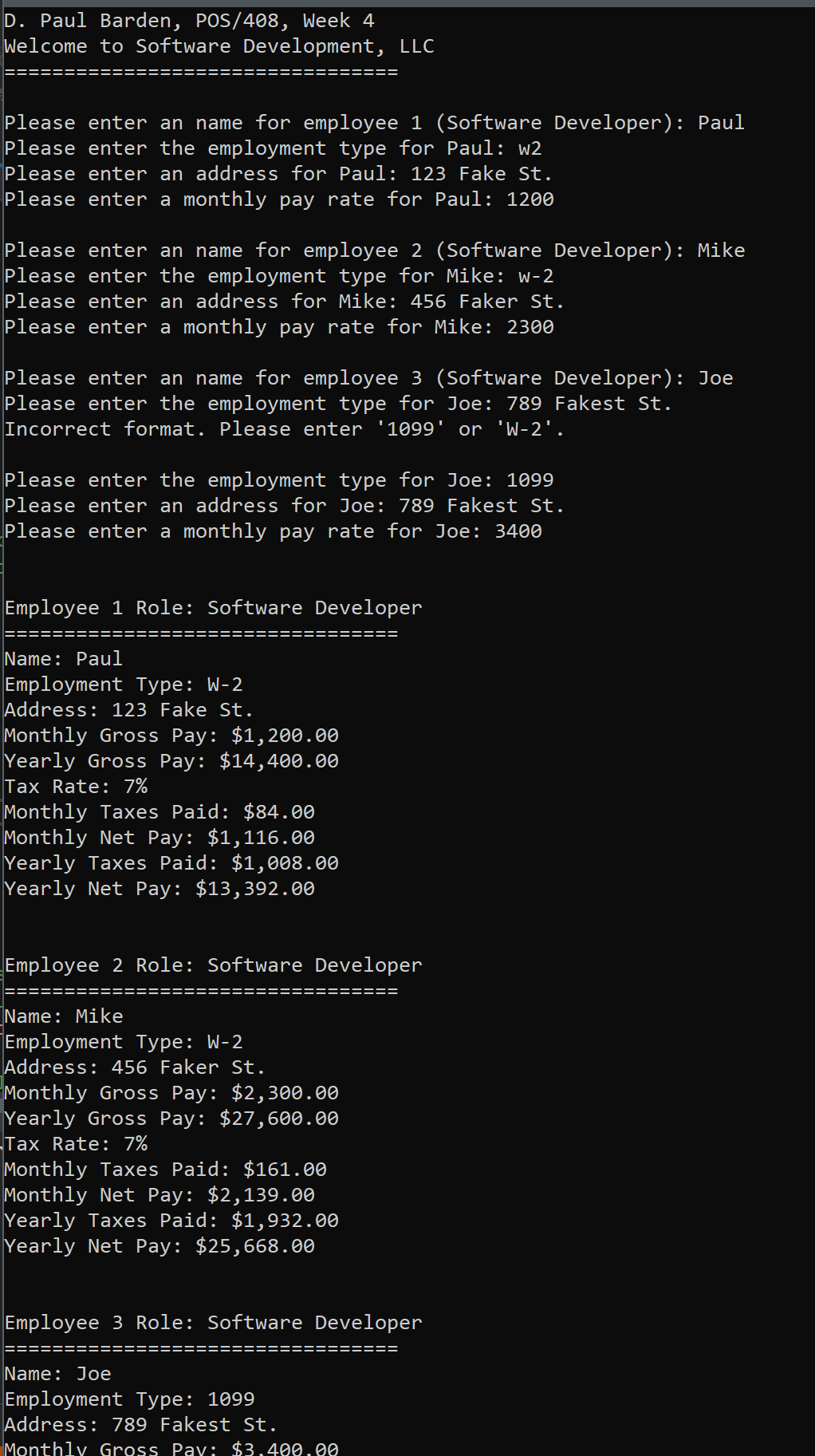
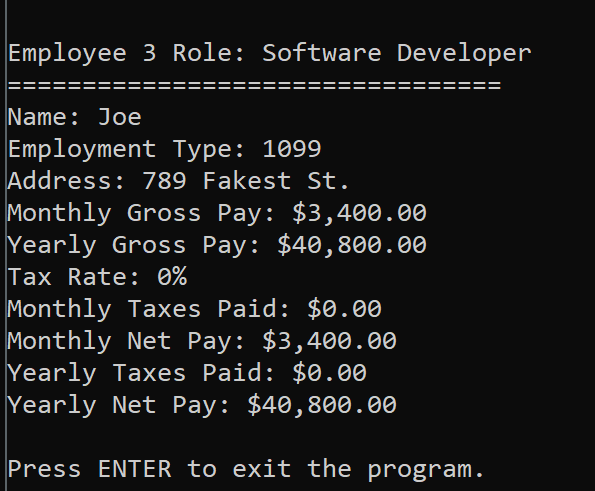
Demonstration of the program running (with intended flow and sample of input validation):





Source code is below; comments are in **BOLD** to explain the application logic:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleAppWeek4

{

class Program

{

static void Main(string[] args)

{

**// Create a new Scanner object (defined on line 237) which handles the collection**

**// of user input and includes limited validation for types string and double**

Scanner InputScanner = new Scanner();

**// Create a list of Software Developers as Employee objects**

List<Employee> Employees = new List<Employee>();

Console.WriteLine("D. Paul Barden, POS/408, Week 4");

Console.WriteLine("Welcome to Software Development, LLC");

Console.WriteLine("=================================");

Console.WriteLine("");

**// actions are performed 3 times, one for each employee**

**// as per the application requirements**

for (int n = 0; n < 3; n++)

{

**// Use the Scanner object to read user input. The user prompt is passed in.**

**// InputScanner() returns a string which is assigned to the UserInput string variable**

string UserInput = InputScanner.Read("Please enter an name for employee " + (n + 1) + " (Software Developer): ");

**// Create a new Employee object using the name assigned to the UserInput variable above**

**// changed from: Employee SoftwareDeveloper = new Employee(UserInput);**

Employees.Add(new Employee(UserInput));

**// InputScanner() returns a string which is assigned to the UserInput string variable**

UserInput = InputScanner.Read("Please enter the employment type for " + Employees[n].GetName() + ": ");

bool EmpTypeCheck = false;

**// checks for possible variations of W-2 and correct them**

if ((UserInput.ToUpper() == "W2") || (UserInput.ToUpper() == "W-2"))

{

**// correct user input and set the boolean to true to bypass the while loop below**

UserInput = "W-2";

EmpTypeCheck = true;

}

**// if the user input is acceptable, set the boolean to true to bypass the while loop below**

if ((UserInput == "1099") || (UserInput == "W-2"))

{

EmpTypeCheck = true;

}

**// if input wasn't acceptable and the boolean wasn't set, this loop executes until**

**// the user inputs an acceptable answer for the employment type**

while (!EmpTypeCheck)

{

**// display error message to user**

Console.WriteLine("Incorrect format. Please enter '1099' or 'W-2'.");

Console.WriteLine("");

**// prompt for another entry and perform checks/corrections**

UserInput = InputScanner.Read("Please enter the employment type for " + Employees[n].GetName() + ": ");

if ((UserInput.ToUpper() == "W2") || (UserInput.ToUpper() == "W-2"))

{

UserInput = "W-2";

EmpTypeCheck = true;

}

else if (UserInput == "1099")

{

EmpTypeCheck = true;

}

}

**// set employee type if it passes the above error checks**

Employees[n].SetEmploymentType(UserInput);

**// InputScanner() returns a string which is assigned to the UserInput string variable**

UserInput = InputScanner.Read("Please enter an address for " + Employees[n].GetName() + ": ");

**// Use UserInput to assign a new address to the employee**

Employees[n].SetAddress(UserInput);

**// InputScanner() returns a double which is assigned to the UserNumInput double variable**

double UserNumInput = InputScanner.ReadDouble("Please enter a monthly pay rate for " + Employees[n].GetName() + ": ");

Employees[n].SetMonthlyGrossPay(UserNumInput);

if (Employees[n].GetEmploymentType() == "W-2")

{

**// Sets a tax rate for W-2 employees**

Employees[n].SetTaxRate(7.0);

}

**// Create space bwtween sets of input for easier viewing**

Console.WriteLine("");

}

for (int n = 0; n < 3; n++)

{

**// Display all employee information stored**

**// Now updated to display each employee's info**

Console.WriteLine("");

Console.WriteLine("Employee " + (n + 1) + " Role: Software Developer");

Console.WriteLine("=================================");

Console.WriteLine("Name: {0}", Employees[n].GetName());

Console.WriteLine("Employment Type: {0}", Employees[n].GetEmploymentType());

Console.WriteLine("Address: {0}", Employees[n].GetAddress());

Console.WriteLine("Monthly Gross Pay: ${0:n2}", Employees[n].GetMonthlyGrossPay());

Console.WriteLine("Yearly Gross Pay: ${0:n2}", Employees[n].GetYearlyGrossPay());

Console.WriteLine("Tax Rate: {0}%", Employees[n].GetTaxRate());

Console.WriteLine("Monthly Taxes Paid: ${0:n2}", Employees[n].GetTaxesPaid());

Console.WriteLine("Monthly Net Pay: ${0:n2}", Employees[n].GetNetPay());

Console.WriteLine("Yearly Taxes Paid: ${0:n2}", Employees[n].GetYearlyTaxesPaid());

Console.WriteLine("Yearly Net Pay: ${0:n2}", Employees[n].GetYearlyNetPay());

Console.WriteLine("");

}

**// Added to prevent the application from closing until user has a chance to view information** output above

Console.WriteLine("Press ENTER to exit the program.");

string NewInput = Console.ReadLine();

}

}

**// Defines the Employee object**

class Employee

{

**// Attributes belonging to the instance object**

private string Name;

private string Address;

private double MonthlyGrossPay;

private double NetPay;

private double TaxRate;

private string EmploymentType;

**// Initializes the Employee object using a string which is then assigned to the name instance variable**

public Employee(string name)

{

this.Name = name;

}

**// Returns the name as a string**

public string GetName()

{

return this.Name;

}

**// Sets a new employee name**

public void SetName(string NewName)

{

this.Name = NewName;

}

**// Returns the address as a string**

public string GetAddress()

{

return this.Address;

}

**// Sets a new employee address**

public void SetAddress(string NewAddress)

{

this.Address = NewAddress;

}

**// Returns the gross pay as a double**

public double GetMonthlyGrossPay()

{

return this.MonthlyGrossPay;

}

**// Sets a new employee monthly gross pay**

public void SetMonthlyGrossPay(double AdjustedGross)

{

this.MonthlyGrossPay = AdjustedGross;

}

**// Returns the net pay as a double**

public double GetNetPay()

{

this.NetPay = this.MonthlyGrossPay - GetTaxesPaid();

return this.NetPay;

}

**// Returns the tax rate as a double**

public double GetTaxRate()

{

return TaxRate;

}

**// Sets a new employee tax rate as a percentage**

public void SetTaxRate(double Rate)

{

this.TaxRate = Rate;

}

**// Returns the tax amount deducted from the monthly pay**

public double GetTaxesPaid()

{

double TaxAmount = this.MonthlyGrossPay \* (this.TaxRate / 100);

return TaxAmount;

}

**// Returns the gross yearly pay as a double**

public double GetYearlyGrossPay()

{

return this.MonthlyGrossPay \* 12;

}

**// Returns the net yearly pay as a double**

public double GetYearlyNetPay()

{

return this.NetPay \* 12;

}

**// Returns the tax amount deducted from the yearly pay**

public double GetYearlyTaxesPaid()

{

double TaxAmount = this.MonthlyGrossPay \* (this.TaxRate / 100);

return TaxAmount \* 12;

}

**// Returns the Employment Type as a string**

public string GetEmploymentType()

{

return this.EmploymentType;

}

**// Sets the Employment Type**

public void SetEmploymentType(string emptype)

{

this.EmploymentType = emptype;

}

}

**// Defines the Scanner object**

class Scanner

{

**// Instantiates the Scanner**

public Scanner() { }

**// Used to return a string collected from user on the console**

**// Takes the user message prompt passed in as a parameter**

public string Read(string Prompt)

{

**// Initialize UserInput as an empty string**

string UserInput = "";

**// This loop is performed until the user enters a valid string**

while (UserInput == "")

{

try

{

**// Write the prompt to the console and collect user input from the same line**

Console.Write(Prompt);

string ProcessInput = Console.ReadLine();

**// If nothing is entered, an exception will be thrown and its message will be displayed on the console**

if (ProcessInput == "")

{

throw new Exception("Error: No argument entered.");

}

**// If something is entered, assign it to the string object**

else

{

UserInput = ProcessInput;

}

}

**// Generic exceptions are caught and displayed, though in a full**

**// program this can be an opportunity to correct errors**

catch (Exception e)

{

Console.WriteLine(e.Message);

}

}

**// Returns user input as a string**

return UserInput;

}

**// Used to return a double collected from user on the console**

**// Takes the user message prompt passed in as a parameter**

public double ReadDouble(string Prompt)

{

**// Initialize UserInput as zero**

double UserInput = 0;

**// This loop is performed until the user enters a valid double**

while (UserInput == 0)

{

try

{

**// Write the prompt to the console and collect user input from the same line**

Console.Write(Prompt);

double ProcessInput = Convert.ToDouble(Console.ReadLine());

**// Executes when user enters a value 0 or less**

if (ProcessInput < 0 || ProcessInput == 0)

{

throw new Exception("Error: Number must be greater than zero.");

}

**// Executes if user enters number more than 0**

else if (ProcessInput > 0)

{

UserInput = ProcessInput;

}

**// Otherwise an exception is thrown**

else

{

throw new Exception("Error: No valid argument entered.");

}

}

**// Catches generic exceptions and displays message on the console**

catch (Exception e)

{

Console.WriteLine(e.Message);

}

}

**// returns input as a double**

return UserInput;

}

}

}