CECS 475

Software Dev With Frameworks

Lab Assignment 2

Alfredo Vargas
014270722
Professor Nguyen
Due Date: 17 September 2018

Program

```
class Program
    {
        /// <summary>
        ///     delegate function, used for sorting list
        /// </summary>
        /// <param name="a"></param>
        /// <param name="b"></param>
        /// <returns></returns>
        public delegate bool CompareDelegate(object a, object b);

        /// <summary>
        ///     print the items of the list
        /// </summary>
        /// <param name="employees"></param>
        static void printEmployees(List<IPayable> employees)
        {
            foreach (IPayable employee in employees)
            {
                Console.WriteLine(employee.ToString());
                Console.WriteLine();
            }
        }

        /// <summary>
        ///     user options menu
        /// </summary>
        /// <returns> user's menu choice </returns>
        static int menu()
        {
            bool validEntry = false;
            int choice = 0;
            while (!validEntry)
            {
                Console.WriteLine(
                    "Lab Assignment 2\nMenu:" +
                    "\n[1] Sort by Last Name (Desc)" +
                    "\n[2] Sort by Pay Amount (Asc)" +
                    "\n[3] Sort by Social Security (Desc)" +
                    "\n[4] Sort by Last Name (Asc), Pay Amount (Desc) - LINQ" +
                    "\n[0] Exit");
                Console.Write("\nChoice: ");
                try
                {
                    choice = Convert.ToInt32(Console.ReadLine());
                    validEntry = true;
                }
                catch (FormatException)
                {
                    Console.WriteLine("\nInput is not a string value.");
                    Console.WriteLine();
                    continue;
                }
            }
            return choice;
        }
        static void Main(string[] args)
        {
            List<IPayable> employeeList = new List<IPayable>();
            employeeList.Add(new SalariedEmployee("John", "Smith", "111-11-1111", 700M));
            employeeList.Add(new SalariedEmployee("Antonio", "Smith", "555-55-5555", 800M));
            employeeList.Add(new SalariedEmployee("Victor", "Smith", "444-44-4444", 600M));
            employeeList.Add(new HourlyEmployee("Karen", "Price", "222-22-2222", 16.75M, 40M));
            employeeList.Add(new HourlyEmployee("Ruben", "Zamora", "666-66-6666", 20.00M, 40M));
            employeeList.Add(new CommissionEmployee("Sue", "Jones", "333-33-3333", 10000M, .06M));
            employeeList.Add(new BasePlusCommissionEmployee("Bob", "Lewis", "777-77-7777", 5000M, .04M,
300M));
            employeeList.Add(new BasePlusCommissionEmployee("Lee", "Duarte", "888-88-888", 5000M, .04M,
300M));
            bool exit_program = false;
            while (!exit_program)
            {
                switch (menu())
                {
                    case 1:
                        // Using IComparer Interface
                        // sort using a delegate that references the inline function
                        employeeList.Sort(delegate (IPayable x, IPayable y)
                        {
                            Employee a = (Employee)x;
                            Employee b = (Employee)y;
                            if (a.LastName == null && b.LastName == null) return 0;
                            else if (a.LastName == null) return 1;
                            else if (b.LastName == null) return 0;
                            else return a.LastName.CompareTo(b.LastName);
                        }
                        );
                        printEmployees(employeeList);
                        break;
                    case 2:
                        // Using IComparer interface
                        // using a comparer class, sort the list in ascending order
                        var watch = System.Diagnostics.Stopwatch.StartNew();
                        Employee_SortByPayAmount_AscendingOrder eAsc = new
Employee_SortByPayAmount_AscendingOrder();
                        employeeList.Sort(eAsc);
                        watch.Stop();
                        printEmployees(employeeList);
                        Console.WriteLine("\nRun Time: " + watch.ElapsedMilliseconds + "ms\n");
                        break;
                    case 3:
                        // Using selection sort and delegate
                        // Using a delegate object, it will be referencing
                        // the function wanted for the sorting algo,
                        //      it is then sent to the selectionSort class to be used.
                        //      - it will sort the list in descending order by employees SSN
                        CompareDelegate EmployeeCompare = new CompareDelegate(Employee.SSNIsGreater);
                        SelectionSortClass.Sort(employeeList, EmployeeCompare);
                        printEmployees(employeeList);
                        break;
                    case 4:
                        // Using LINQ sorting
                        // it will sort the list ascending based on last name,
                        // then descending by payment amount
                        var OrderBy = from employee in employeeList
```

```
                                      orderby ((Employee)employee).LastName, employee.GetPaymentAmount()
descending
                                      select employee;
                        foreach (var employee in OrderBy)
                        {
                            Console.WriteLine(employee);
                            Console.WriteLine();
                        }
                        break;
                    case 0:
                        exit_program = true;
                        break;
                    default:
                        Console.WriteLine("\nNot a valid menu option.\n");
                        break;
                }
            }
            Console.Write("Press any key to continue...");
            Console.ReadKey(true);
        } // end Main
    }
```

SelectionSortClass

```
    /// <summary>
    ///     Sorting the list of IPayable objects using a selection sort
    /// </summary>
    class SelectionSortClass
    {
        /// <summary>
        ///     Sort method
        ///         Compare two employee objects then swap items for sort
        /// </summary>
        /// <param name="employees">
        ///     List of employees
        /// </param>
        /// <param name="gtMethod">
        ///     function used to compare objects is passed in as a delegate param
        /// </param>
        static public void Sort(List<IPayable> employees, Program.CompareDelegate gtMethod)
        {
            var smallest = 0;
            for (int i = 0; i < employees.Count - 1; i++)
            {
                smallest = i;
                for (int j = i + 1; j < employees.Count; j++)
                {
                    Employee a = (Employee)employees[j];
                    Employee b = (Employee)employees[smallest];
                    if (gtMethod(b, a))
                    {
                        swap(employees, j, smallest);
                    }
                }
            }
        }

        /// <summary>
        ///     Swap two items of a list
        /// </summary>
        /// <param name="list">
        ///     List of employees
        /// </param>
        /// <param name="a"></param>
        /// <param name="b"></param>
        static void swap(List<IPayable> list, int a, int b)
        {
            var t = list[a];
            list[a] = list[b];
            list[b] = t;
        }
    }
```

Employee_SortByPayAmount_AscendingOrder

```
/// <summary>
  ///     Sorting class extending IComparer interface
  /// </summary>
  class Employee_SortByPayAmount_AscendingOrder : IComparer<IPayable>
```

```
{
    /// <summary>
    ///     inherited method compare
    ///         - compare two IPayable objects by using
    ///           GetPaymentAmount()
    /// </summary>
    /// <param name="x"></param>
    /// <param name="y"></param>
    /// <returns>
    ///     result of comparison
    ///         - (-1) if x preceedes y
    ///         - (0) if x is same as y
    ///         - (1) if x follows y
    /// </returns>
    int IComparer<IPayable>.Compare(IPayable x, IPayable y)
    {
        if (x == null && y == null) return 0;
        else if (x == null) return 0;
        else if (y == null) return 1;
        else if (x.GetPaymentAmount() > y.GetPaymentAmount()) return 1;
        else if (x.GetPaymentAmount() < y.GetPaymentAmount()) return -1;
        else return 0;
    }
}
```

IPayable

```
/// <summary>
///     IPayable interface
///         - extends IComparable interface
/// </summary>
interface IPayable : IComparable<IPayable>
{
    decimal GetPaymentAmount(); // calculate payment
}
```

Employee

```
    /// <summary>
    ///     static function comparing two objects by SSN
    /// </summary>
    /// <param name="a"></param>
    /// <param name="b"></param>
    /// <returns>
    ///     - return false: (1) if a follows b
    ///     - return false: (0) if a is same as b
    ///     - return true: (-1) if a preceedes b
    ///     - return false: (default) return false
```

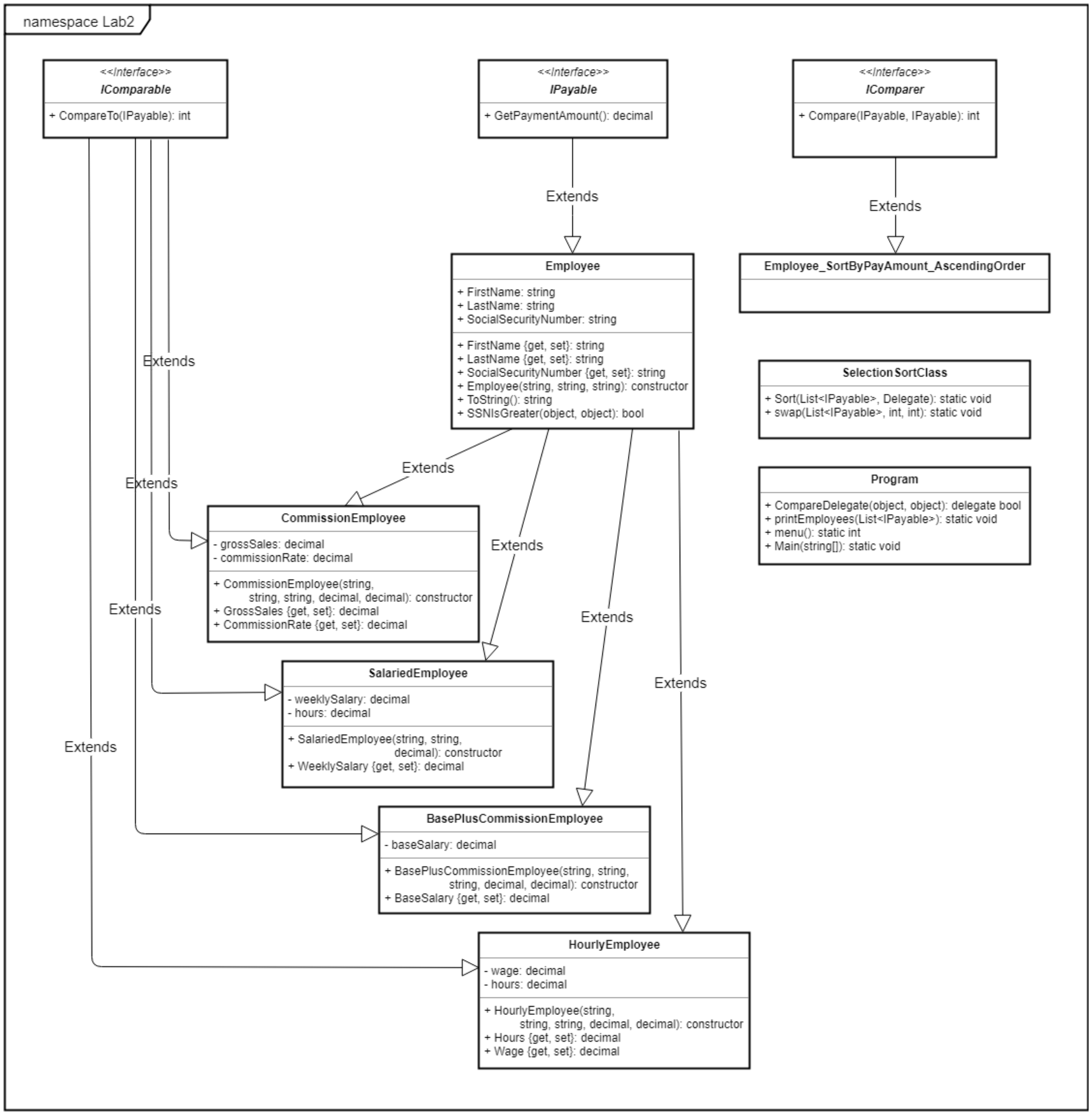
```
        /// </returns>
        public static bool SSNIsGreater(object a, object b)
        {
            Employee e1 = (Employee)a;
            Employee e2 = (Employee)b;
            switch (e1.SocialSecurityNumber.CompareTo(e2.SocialSecurityNumber))
            {
                case 1:
                    return false;
                case 0:
                    return false;
                case -1:
                    return true;
                default:
                    return false;
            }
        }
```

Run Time Output

```
Lab Assignment 2

Menu:

[1] Sort by Last Name (Desc)

[2] Sort by Pay Amount (Asc)

[3] Sort by Social Security (Desc)

[4] Sort by Last Name (Asc), Pay Amount (Desc) - LINQ

[0] Exit


Choice: 2

base-salaried commission employee: Bob Lewis

social security number: 777-77-7777

gross sales: $5,000.00

commission rate: 0.04; base salary: $300.00


base-salaried commission employee: Lee Duarte

social security number: 888-88-888

gross sales: $5,000.00

commission rate: 0.04; base salary: $300.00


salaried employee: Victor Smith
```

```
social security number: 444-44-4444

weekly salary: $600.00


commission employee: Sue Jones

social security number: 333-33-3333

gross sales: $10,000.00

commission rate: 0.06


hourly employee: Karen Price

social security number: 222-22-2222

hourly wage: $16.75; hours worked: 40.00


salaried employee: John Smith

social security number: 111-11-1111

weekly salary: $700.00


salaried employee: Antonio Smith

social security number: 555-55-5555

weekly salary: $800.00


hourly employee: Ruben Zamora

social security number: 666-66-6666

hourly wage: $20.00; hours worked: 40.00


Run Time: 2ms


Lab Assignment 2

Menu:

[1] Sort by Last Name (Desc)

[2] Sort by Pay Amount (Asc)

[3] Sort by Social Security (Desc)

[4] Sort by Last Name (Asc), Pay Amount (Desc) - LINQ

[0] Exit

Choice: ….
```

UML



namespace Lab2
<<Interface>>
IComparable
+ CompareTo(IPayable): int
<<Interface>>
IPayable
+ GetPaymentAmount(): decimal
<<Interface>>
IComparer
+ Compare(IPayable, IPayable): int
Extends
Extends
Employee
+ FirstName: string
+ LastName: string
+ SocialSecurityNumber: string
+ FirstName {get, set}: string
+ LastName {get, set}: string
+ SocialSecurityNumber {get, set}: string
+ Employee(string, string, string): constructor
+ ToString(): string
+ SSNIsGreater(object, object): bool
Employee_SortByPayAmount_AscendingOrder
SelectionSortClass
+ Sort(List<IPayable>, Delegate): static void
+ swap(List<IPayable>, int, int): static void
Program
+ CompareDelegate(object, object): delegate bool
+ printEmployees(List<IPayable>): static void
+ menu(): static int
+ Main(string[]): static void
Extends
Extends
Extends
Extends
Extends
Extends
Extends
Extends
Extends
CommissionEmployee
- grossSales: decimal
- commissionRate: decimal
+ CommissionEmployee(string, string, string, decimal, decimal): constructor
+ GrossSales {get, set}: decimal
+ CommissionRate {get, set}: decimal
SalariedEmployee
- weeklySalary: decimal
- hours: decimal
+ SalariedEmployee(string, string, decimal): constructor
+ WeeklySalary {get, set}: decimal
BasePlusCommissionEmployee
- baseSalary: decimal
+ BasePlusCommissionEmployee(string, string, string, decimal, decimal): constructor
+ BaseSalary {get, set}: decimal
HourlyEmployee
- wage: decimal
- hours: decimal
+ HourlyEmployee(string, string, string, decimal, decimal): constructor
+ Hours {get, set}: decimal
+ Wage {get, set}: decimal