Міністерство освіти і науки України

Національний лісотехнічний університет України

Кафедра інформаційних технологій

**Звіт до лабораторної роботи**

з навчальної дисципліни

**«Методи та засоби ООАП»**

**Виконав:**

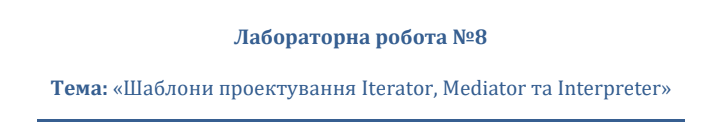
студент групи КНС-21

Марущак С. І.

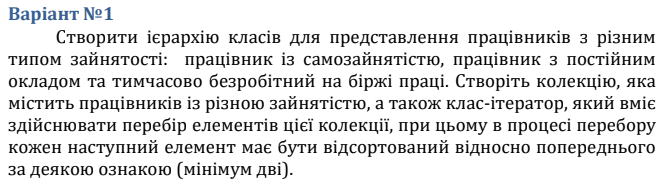
**Перевірив:**

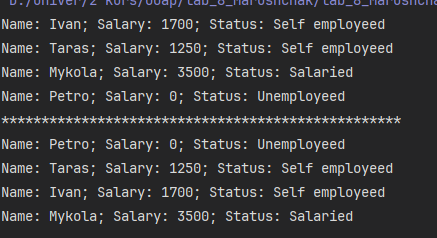
Волинець Є.О.

Львів – 2024

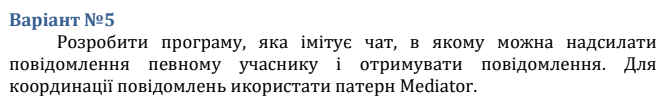


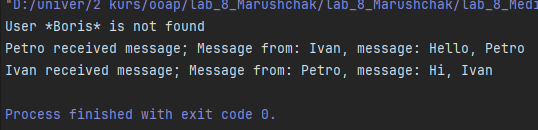




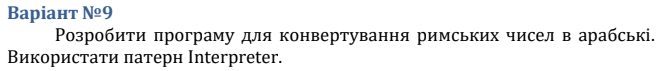


using System;  
using System.Collections;  
using System.Collections.Generic;  
  
namespace lab\_8\_Marushchak  
{  
 abstract class Employee  
 {  
 public string Name { get; set; }  
 public double Salary { get; set; }  
   
 public string Status { get; set; }  
  
 public Employee(string name, double salary)  
 {  
 Name = name;  
 Salary = salary;  
 }  
 }  
   
 class SelfEmployee : Employee  
 {  
 public SelfEmployee(string name, double salary) : base(name, salary)  
 {  
 Status = "Self employeed";  
 }  
   
 }  
   
 class SalariedEmployee: Employee  
 {  
 public SalariedEmployee(string name, double salary) : base(name, salary)  
 {  
 Status = "Salaried";  
 }  
 }  
   
 class UnemployedEmployee : Employee  
 {  
 public UnemployedEmployee(string name) : base(name, 0)  
 {  
 Status = "Unemployeed";  
 }  
 }  
  
 class EmployeeSalaryComparer : IComparer<Employee>  
 {  
 public int Compare(Employee x, Employee y)  
 {  
 return x.Salary.CompareTo(y.Salary);  
 }  
 }  
  
 abstract class Iterator : IEnumerable<Employee>  
 {  
 public abstract Employee CurrentItem();  
 public abstract bool MoveNext();  
  
 public abstract IEnumerator<Employee> GetEnumerator();  
  
 IEnumerator IEnumerable.GetEnumerator()  
 {  
 return GetEnumerator();  
 }  
 }  
   
 abstract class Aggregate  
 {  
 public abstract Iterator CreateIterator();  
 }  
  
 class ConcreteAggregate : Aggregate  
 {  
 private List<Employee> \_items = new List<Employee>();  
  
 public override Iterator CreateIterator()  
 {  
 return new ConcreteIterator(this);  
 }  
  
 public int Count  
 {  
 get { return \_items.Count; }  
 }  
  
 public Employee this[int index]  
 {  
 get { return \_items[index]; }  
 set { \_items.Insert(index, value); }  
 }  
  
 public void AddEmployee(Employee employee)  
 {  
 \_items.Add(employee);  
 }  
  
 public void SortBySalary()  
 {  
 \_items.Sort(new EmployeeSalaryComparer());  
 }  
 }  
  
 class ConcreteIterator : Iterator  
 {  
 private ConcreteAggregate \_aggregate;  
 private int \_current;  
  
 public ConcreteIterator(ConcreteAggregate aggregate)  
 {  
 this.\_aggregate = aggregate;  
 \_current = -1;  
 }  
   
 public override bool MoveNext()  
 {  
 \_current++;  
 return \_current < \_aggregate.Count;  
 }  
   
 public override Employee CurrentItem()  
 {  
 return \_aggregate[\_current];  
 }  
  
 public override IEnumerator<Employee> GetEnumerator()  
 {  
 for (int i = 0; i < \_aggregate.Count; i++)  
 {  
 yield return \_aggregate[i];  
 }  
 }  
 }  
  
 internal class Program  
 {  
 public static void Main(string[] args)  
 {  
 ConcreteAggregate a = new ConcreteAggregate();  
 a.AddEmployee(new SelfEmployee("Ivan", 1700));  
 a.AddEmployee(new SelfEmployee("Taras", 1250));  
 a.AddEmployee(new SalariedEmployee("Mykola", 3500));  
 a.AddEmployee(new UnemployedEmployee("Petro"));  
   
  
 Iterator i = a.CreateIterator();  
  
 while (i.MoveNext())  
 {  
 Employee employee = i.CurrentItem();  
 Console.WriteLine($"Name: {employee.Name}; " +  
 $"Salary: {employee.Salary}; " +  
 $"Status: {employee.Status}");  
 }  
  
 Console.WriteLine(new string('\*', 50));  
   
 a.SortBySalary();  
   
  
 foreach (Employee employee in i)  
 {  
 Console.WriteLine($"Name: {employee.Name}; " +  
 $"Salary: {employee.Salary}; " +  
 $"Status: {employee.Status}");  
 }  
  
 Console.ReadKey();  
 }  
 }  
}





using System;  
using System.Collections.Generic;  
  
namespace lab\_8\_Mediator  
{  
 interface IChatMediator  
 {  
 void SendMessage(string message, User sender, string receiverName);  
 }  
  
 abstract class User  
  
 {  
 protected IChatMediator \_chatMediator;  
 public string Name { get; }  
  
 protected User(IChatMediator chatMediator,string name)  
 {  
 \_chatMediator = chatMediator;  
 Name = name;  
 }  
  
 public abstract void ReceiveMessage(string message);  
 }  
   
 class ChatMediator : IChatMediator  
 {  
 private List<User> \_users = new List<User>();  
  
 public void AddUser(User user)  
 {  
 \_users.Add(user);  
 }  
   
 public void SendMessage(string message, User sender, string receiverName)  
 {  
 User receiver = \_users.Find(usr => usr.Name== receiverName);  
  
 if (receiver != null)  
 {  
 receiver.ReceiveMessage($"Message from: {sender.Name}, message: {message}");  
 }  
 else  
 {  
 Console.WriteLine($"User \*{receiverName}\* is not found");  
 }  
 }  
 }  
   
 class ConctreateUser : User  
 {  
 public ConctreateUser(IChatMediator chatMediator, string name) : base(chatMediator, name)  
 {  
 }  
  
 public override void ReceiveMessage(string message)  
 {  
 Console.WriteLine($"{Name} received message; {message}");  
 }  
  
 public void SendMessage(string message, string receiverName)  
 {  
 \_chatMediator.SendMessage(message,this,receiverName);  
 }  
 }  
  
 internal class Program  
 {  
 public static void Main(string[] args)  
 {  
 ChatMediator mediator = new ChatMediator();  
  
 ConctreateUser user1 = new ConctreateUser(mediator, "Ivan");  
 ConctreateUser user2 = new ConctreateUser(mediator, "Petro");  
  
 mediator.AddUser(user1);  
 mediator.AddUser(user2);  
  
 user1.SendMessage("Hello, Boris","Boris");  
 user1.SendMessage("Hello, Petro", "Petro");  
 user2.SendMessage("Hi, Ivan", "Ivan");  
 }  
 }  
}



**Висновок:** на даній лабораторній роботі використовував шаблон проектування Iterator, Mediator, Interpreter.