НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КПІ»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра програмного забезпечення комп’ютерних систем

ЛАБОРАТОРНА РОБОТА №5

*з дисципліни*

*«Об’єктно-орієнтоване програмування»*

ТЕМА: «Реалізація складних поведінкових шаблонів

проектування»

Підготував: студент групи КП-51

Бабенко Валерій Павлович

Перевірила:

Заболотня Тетяна Миколаївна

Київ 2017

# Мета роботи

# Ознайомлення з основними характеристиками шаблонів «Ланцюжок обов’язків», «Команда» та «Посередник», запам’ятовування поширених ситуацій, коли використання цих шаблонів є доцільним, набуття вмінь та навичок реалізації шаблонів під час створення програмного коду.

# Постановка задачі

Варіант 2

1. За допомогою шаблону проектування змоделювати роботу викладачів, які перевіряють екзаменаційні роботи студентів з певного предмету. Кожна робота перевіряється тільки одним викладачем. Якщо перший викладач зайнятий перевіркою роботи, вона передається для перевірки другому і т.д. У разі, якщо були зайняті всі викладачі і екзаменаційна робота залишилась неперевіреною, вона надходить на перевірку повторно до тих пір, поки її не перевірить якийсь викладач.
2. Для роботи в інформаційній системі користувач повинен заповнити додаткову інформацію про себе у своєму профілі в системі. Система пропонує ввести дані щодо ПІБ та дати народження користувача. Крім того, система надає можливість відмінити збереження цих даних. За допомогою шаблону проектування реалізувати поведінку такої системи.

# Вибір структурного шаблону програмування

1. «перевіряється тільки одним викладачем», всі роботи повинні бути перевіреними, «оброби сам чи передай іншому» - це шаблон «**Ланцюжок обов’язків**».
2. «відмінити збереження цих даних» – це шаблон «**Команда**».

# Лістинг програми

|  |
| --- |
| Program.cs |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading;  using System.Threading.Tasks;  namespace OOP2\_Lab5  {  class Program  {  static void Main(string[] args)  {  #region Testing  Work work1 = new Work("some text", "Babenko");  Work work2 = new Work("some text", "Ivanov");  Work work3 = new Work("some text", "Flac");  Work work4 = new Work("some text", "Yella");  Work work5 = new Work("some text", "Trash");  AbstractTeacher teacher1 = new Teacher1();  AbstractTeacher teacher2 = new Teacher2();  AbstractTeacher teacher3 = new Teacher3();  AbstractTeacher chain = new Chain();  teacher1.Successor = teacher2;  teacher2.Successor = teacher3;  teacher3.Successor = chain;  chain.Successor = teacher1;  teacher1.testWork(work1);  teacher1.testWork(work2);  teacher1.testWork(work3);  teacher1.testWork(work4);  teacher1.testWork(work5);  while (true)  {  if (work1.Mark != 0 && work2.Mark != 0 && work3.Mark != 0 && work4.Mark != 0 && work5.Mark != 0)  {  Thread.Sleep(500);  Console.WriteLine("Testing has ended");  break;  }  }  #endregion  Console.WriteLine(" ");  #region Registration  System sys = new System();  sys.createProfile(1);  sys.createProfile(11);  User valera = new User();  User behrang = new User();  FillingFieldsCommand command = new FillingFieldsCommand();  valera.setExecutor(command, sys);  valera.execute(command, "Valera Babenko Pavlovich", new DateTime(), 1);  valera.unExecute(command, 1);  valera.setExecutor(command, sys);  valera.execute(command, "Behrang Behvandi Abdulahovich", new DateTime(), 11);  sys.display();  #endregion  Console.WriteLine(" ");  }  }  } |

|  |
| --- |
| Registration.cs |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Collections;  namespace OOP2\_Lab5  {  //a part of the receiver  class Profile  {  private int id;  public int Id  {  get { return id; }  }  private string fullname;  public string Fullname  {  get { return fullname; }  }  private DateTime birthdayDate;  public DateTime BirthdayDate  {  get { return birthdayDate; }  }  public Profile(int id)  {  this.id = id;  }  public void setData(string fullname, DateTime birthdayDate)  {  this.fullname = fullname;  this.birthdayDate = birthdayDate;  }  public void cleanData()  {  this.fullname = null;  this.birthdayDate = new DateTime();  }  public void display()  {  Console.Write("Id: "+ id);  if (fullname != null)  {  Console.Write("; Full name: " + fullname + "; Birthday: "+ birthdayDate);  }  Console.WriteLine();  }  }  //invoker  class User  {  public void setExecutor(FillingFieldsCommandAbstract t, AbstractSystem e)  {  t.\_Executor = e;  }  public void execute(FillingFieldsCommandAbstract t, string fullname, DateTime birthdate, int id)  {  t.execute(fullname, birthdate, id);  }  public void unExecute(FillingFieldsCommandAbstract t, int id)  {  t.unExecute(id);  }  }  //command  abstract class FillingFieldsCommandAbstract  {  protected AbstractSystem executor;  public AbstractSystem \_Executor  {  set { executor = value; }  }  abstract public void execute(string fullname, DateTime birthdate, int id);  abstract public void unExecute(int id);  }  //concrete command  class FillingFieldsCommand : FillingFieldsCommandAbstract  {  override public void execute(string fullname, DateTime birthdate, int id)  {  if (executor != null)  {  executor.fillFields(fullname, birthdate, id);  }  else  {  throw new Exception();  }  }  override public void unExecute(int id)  {  if (executor != null)  {  executor.cleanFields(id);  }  else  {  throw new Exception();  }  }  }  //receiver; executor  abstract class AbstractSystem  {  abstract public void createProfile(int id);  abstract public void fillFields(string fullname, DateTime birthdate, int id);  abstract public void cleanFields(int id);  abstract public void display();  }  class System : AbstractSystem  {  private List<Profile> list;  public System()  {  list = new List<Profile>();  }    override public void createProfile(int id)  {  Profile profile = new Profile(id);  list.Add(profile);  }  override public void fillFields(string fullname, DateTime birthdate, int id)  {  foreach(Profile prof in list)  {  if (prof.Id == id)  {  prof.setData(fullname, birthdate);  return;  }  }  throw new Exception();  }  override public void cleanFields(int id)  {  foreach (Profile prof in list)  {  if (prof.Id == id)  {  prof.cleanData();  return;  }  }  throw new Exception();  }  override public void display()  {  Console.WriteLine("There're profiles:");  foreach (Profile prof in list)  {  prof.display();  }  }  }  } |

|  |
| --- |
| Examination.cs |
| using System.Threading;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace OOP2\_Lab5  {  public class Work  {  private int mark;  public int Mark  {  set { mark = value; }  get { return mark; }  }  private string text;  private string author;  public string Author  {  get { return author; }  }  public Work(string text, string author)  {  this.text = text;  this.author = author;  this.mark = 0;  }  }  abstract public class AbstractTeacher  {  public Thread testing;  protected AbstractTeacher \_successor;  public AbstractTeacher Successor  {  set { \_successor = value; }  }  abstract public void testWork(Work work);  }  public class Teacher1 : AbstractTeacher  {  override public void testWork(Work work)  {  if (testing == null || testing.IsAlive == false)  {  testing = new Thread(delegate ()  {  Thread.Sleep(1000);  Random rand = new Random();  work.Mark = rand.Next(1, 5);  Console.WriteLine("1st teacher: The work was tested. " + work.Author + " had " + work.Mark + " points");  });  testing.Start();  }  else  {  Console.WriteLine("Out of 1st!");  if (\_successor != null)  {  \_successor.testWork(work);  }  else  {  throw new ApplicationException("ChainOfResponsibility object exhausted all successors without call being handled.");  }  }  }  }  public class Teacher2 : AbstractTeacher  {  override public void testWork(Work work)  {  if (testing == null || testing.IsAlive == false)  {  testing = new Thread(delegate ()  {  Thread.Sleep(2000);  Random rand = new Random();  work.Mark = rand.Next(1, 5);  Console.WriteLine("2nd teacher: The work was tested. " + work.Author + " had " + work.Mark + " points");  });  testing.Start();  }  else  {  Console.WriteLine("Out of 2nd!");  if (\_successor != null)  {  \_successor.testWork(work);  }  else  {  throw new ApplicationException("ChainOfResponsibility object exhausted all successors without call being handled.");  }  }  }  }  public class Teacher3 : AbstractTeacher  {  override public void testWork(Work work)  {  if (testing == null || testing.IsAlive == false)  {  testing = new Thread(delegate ()  {  Thread.Sleep(3000);  Random rand = new Random();  work.Mark = rand.Next(1, 5);  Console.WriteLine("3rd teacher: The work was tested. " + work.Author + " had " + work.Mark + " points");  });  testing.Start();  }  else  {  Console.WriteLine("Out of 3rd!");  if (\_successor != null)  {  \_successor.testWork(work);  }  else  {  throw new ApplicationException("ChainOfResponsibility object exhausted all successors without call being handled.");  }  }  }  }  //this class holds a work to prevent overflowing the stack  public class Chain : AbstractTeacher  {  override public void testWork(Work work) {  Thread.Sleep(100);  \_successor.testWork(work);  }  }  }  } |

