

МIНIСТЕРСТВО ОСВIТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**Розрахункова графічна робота**

з дисципліни “ООП ”

Варіант №10

|  |  |  |
| --- | --- | --- |
| Виконав  студент II курсу  групи КП-92  Мовчан Максим Олександрович  (*прізвище, ім’я, по батькові*) |  | Перевірила  26 грудня 2020 р.  Викладач  Заболотня Тетяна Миколаївна  (*прізвище, ім’я, по батькові*) |

Київ 2020

**Постановка задачі**

**Варіант №10**

1. За допомогою шаблона проектування реалізувати механізм збирання коштів на ремонт школи. Кожен класний керівник збирає кошти з батьків свого класу. Кожний завуч збирає гроші з керівників старших, середніх та молодших класів. Директор збирає гроші із завучів.

2. Розробити модуль до програмного забезпечення роботи бухгалтерії, який буде автоматично формувати пакет електронних фінансових документів (акт виконаних робіт, рахунок-фактура тощо), що підтверджують виконання обов’язків компанії перед замовником, тільки у разі наявності оплати останнім послуги чи товару.

3. За допомогою шаблона проектування допоможіть Діду Морозу сформувати подарунковий набір з іграшок, цукерок тощо та його розтиражувати. Всі складові набору повинні бути реалізовані як самостійні об’єкти та скопійовані до кожного наступного повністю.

4. За допомогою шаблону проектування реалізувати перемикання режимів роботи сервера, до якого через мережу мають змогу підключатися різні користувачі. У випадку перевищення припустимого числа підключень, сервер перестає надавати доступ до своїх ресурсів та видає повідомлення про це всім користувачам, які роблять спроби встановити з ним з’єднання. У разі наявності можливості підключення до нього, сервер приймає запит на встановлення зв’язку.

5. User Component містить текстове поле для введення тексту та три кнопки для його збереження до файлу. У відповідність кожній кнопці поставлений окремий спосіб збереження тексту. Перший спосіб – звичайний: зберігаємо текст без внесення жодних змін. Другий спосіб – видаляємо всі зайві пробіли з тексту перед збереженням. Третій спосіб – застосовуємо кодування тексту (або архівацію). За допомогою шаблону проектування реалізувати описаний User Component, якщо відомо, що метод «Зберегти у файл» у нього один (немає жодних override варіантів методу).

**Обґрунтування вибору шаблонів**

**До задачі №1 —** *Компонувальник*

* Дозволяє через єдиний інтерфейс доступитись до всіх підкласів

**До задачі №2** — *Заступник*

* Дозволяє помістити у проміжний клас необхідну логіку (перевірки, etc. )

**До задачі №3** – *Прототип*

* Вводить загальний інтерфейс для всіх об’єктів, що підтримують клонування. Це дозволяє копіювати об’єкти, не прив’язуючись до їхніх конкретних класів.

**До задачі №4 –** *Стан*

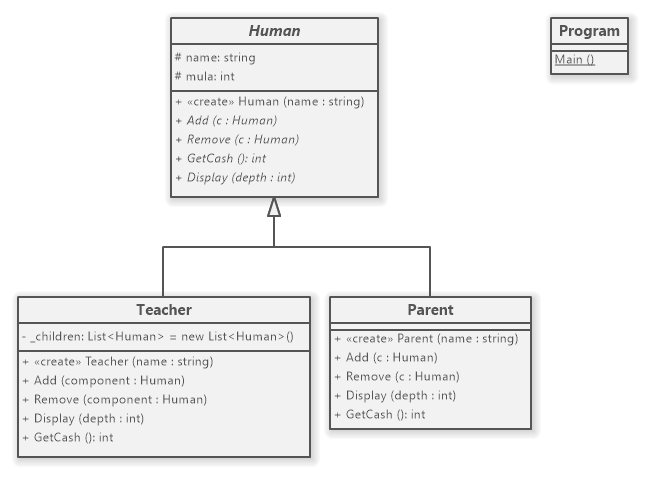
* поведінка об'єкта залежить від його стану і повинна змінюватися під час виконання;
* одна і та сама структура умовного оператора повторюється в кількох операціях, патерн “Стан” передбачає розміщення кожної гілки в окремому класі.

**До задачі №5** – *Команда*

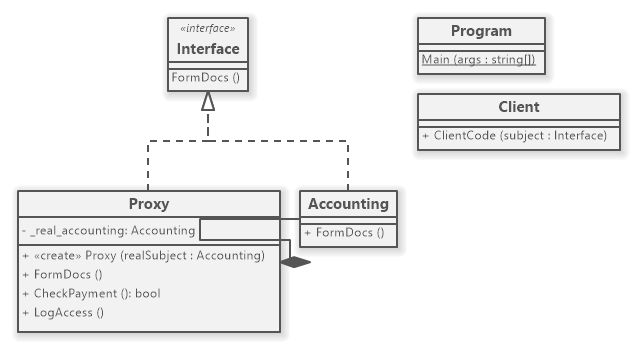
* Дозволяє не надсилати виклики до логіки безпосередньо
* Кожен виклик, що відрізняється від інших, завертається у власний клас (команду).

**UML-діаграми**

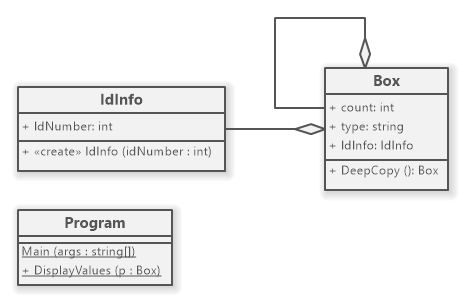
**Завдання 1**



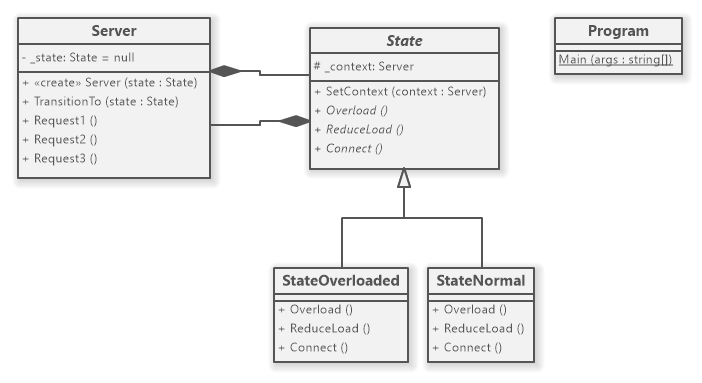
**Завдання 2**



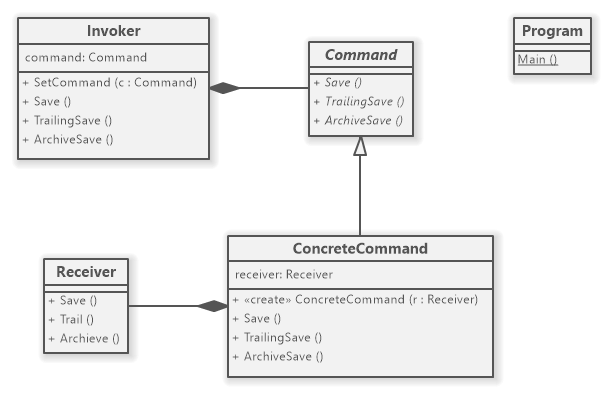
**Завдання3**



**Завдання 4**



**Завдання 5**



**Код завдань**

|  |
| --- |
| **До задачі №1 -** *Компонувальник* |
| using System;  using System.Collections.Generic;  namespace \_1  {  class Program  {  static void Main()  {  Teacher director = new Teacher("Director: ");  Teacher dephead1 = new Teacher("Deputy Head 1: ");  director.Add(dephead1);  Teacher dephead2 = new Teacher("Deputy Head 2: ");  director.Add(dephead2);  Teacher teacher1 = new Teacher("Teacher 1: ");  Teacher teacher2 = new Teacher("Teacher 2: ");  Teacher teacher3 = new Teacher("Teacher 3: ");  dephead1.Add(teacher1);  dephead1.Add(teacher2);  dephead2.Add(teacher3);  Parent parent1\_1 = new Parent("Parent 1: ");  Parent parent2\_1 = new Parent("Parent 1: ");  Parent parent2\_2 = new Parent("Parent 2: ");  Parent parent3\_1 = new Parent("Parent 1: ");  teacher1.Add(parent1\_1);  teacher2.Add(parent2\_1);  teacher2.Add(parent2\_2);  teacher3.Add(parent3\_1);  director.GetCash();  director.Display(2);  }  }  abstract class Human  {  protected string name;  protected int mula;  // Constructor  public Human(string name)  {  this.name = name;  }  public abstract void Add(Human c);  public abstract void Remove(Human c);  public abstract int GetCash();  public abstract void Display(int depth);  }  /// <summary>  /// The 'Composite' class  /// </summary>  class Teacher : Human  {  private List<Human> \_children = new List<Human>();  // Constructor  public Teacher(string name) : base(name)  {  this.mula = 0;  }  public override void Add(Human component)  {  \_children.Add(component);  }  public override void Remove(Human component)  {  \_children.Remove(component);  }  public override void Display(int depth)  {  Console.WriteLine(new String('-', depth) + name + " " + this.mula.ToString());  // Recursively display child nodes  foreach (Human component in \_children)  component.Display(depth + 2);  }  public override int GetCash()  {  this.mula = 0;  foreach (Human component in \_children)  this.mula += component.GetCash();  return this.mula;  }  }  class Parent : Human  {  public Parent(string name) : base(name)  {  Random rnd = new Random();  this.mula = rnd.Next(150);  }  public override void Add(Human c)  {  Console.WriteLine("Impossible operation");  }  public override void Remove(Human c)  {  Console.WriteLine("Impossible operation");  }  public override void Display(int depth)  {  Console.WriteLine(new String('-', depth) + name + " " + mula.ToString());  }  public override int GetCash()  {  return this.mula;  }  }  } |

|  |
| --- |
| **До задачі №2** - *Заступник* |
| using System;  namespace \_2 {  public interface Interface {  void FormDocs();  }  class Accounting: Interface {  public void FormDocs() { Console.WriteLine("Accounting: Forming docs."); }  }  class Proxy: Interface {  private Accounting \_real\_accounting;  public Proxy(Accounting realSubject) {  this.\_real\_accounting = realSubject;  }  public void FormDocs() {  if (this.CheckPayment()) {  this.\_real\_accounting.FormDocs();  this.LogAccess();  }  }  public bool CheckPayment() {  Console.WriteLine("Proxy: Checking payment.");  return true;  }  public void LogAccess() {  Console.WriteLine("Proxy: Logging the time of request.");  }  }  public class Client {  public void ClientCode(Interface subject) { subject.FormDocs(); }  }  class Program {  static void Main(string[] args) {  Client client = new Client();  Console.WriteLine("Without proxy");  Accounting realSubject = new Accounting();  client.ClientCode(realSubject);  Console.WriteLine();  Console.WriteLine("With proxy");  Proxy proxy = new Proxy(realSubject);  client.ClientCode(proxy);  }  }  } |

|  |
| --- |
| **До задачі №3** – *Прототип* |
| using System;  namespace \_3 {  public class Box {  public int count;  public string type;  public IdInfo IdInfo;  public Box DeepCopy() {  Box clone = (Box) this.MemberwiseClone();  clone.IdInfo = new IdInfo(IdInfo.IdNumber);  clone.type = String.Copy(type);  return clone;  }  }  public class IdInfo {  public int IdNumber;  public IdInfo(int idNumber) { this.IdNumber = idNumber; }  }  class Program {  static void Main(string[] args) {  Box p1 = new Box();  p1.count = 42;  p1.type = "Candy";  p1.IdInfo = new IdInfo(666);  Box p2 = p1.DeepCopy();  Console.WriteLine("Original values of box1, box2: ");  Console.WriteLine(" box1 instance values: ");  DisplayValues(p1);  Console.WriteLine(" box2 instance values:");  DisplayValues(p2);  p1.count = 32;  p1.type = "Toy";  p1.IdInfo.IdNumber = 7878;  Console.WriteLine("\nValues of box1, box2 after changes to box1:");  Console.WriteLine(" box1 values: ");  DisplayValues(p1);  Console.WriteLine(" box2 values: ");  DisplayValues(p2);  }  public static void DisplayValues(Box p) {  Console.WriteLine(" Count: {0:d}, Type: {1:s}", p.count, p.type);  Console.WriteLine(" ID#: {0:d}", p.IdInfo.IdNumber);  }  }  } |

|  |
| --- |
| **До задачі №4 -** *Стан* |
| using System;  namespace \_4 {  class Server {  private State \_state = null;  public Server(State state) { this.TransitionTo(state); }  public void TransitionTo(State state) {  Console.WriteLine($"Context: Transition to {state.GetType().Name}.");  this.\_state = state;  this.\_state.SetContext(this);  }  public void Request1() { this.\_state.Overload(); }  public void Request2() { this.\_state.ReduceLoad(); }  public void Request3() { this.\_state.Connect(); }  }  abstract class State {  protected Server \_context;  public void SetContext(Server context) { this.\_context = context; }  public abstract void Overload();  public abstract void ReduceLoad();  public abstract void Connect();  }  class StateNormal: State {  public override void Overload() {  Console.WriteLine("State switched to \"Overloaded\"");  this.\_context.TransitionTo(new StateOverloaded());  }  public override void ReduceLoad() {  Console.WriteLine("StateNormal tried to reduce load.");  }  public override void Connect() {  Console.WriteLine("Connection successful");  }  }  class StateOverloaded: State {  public override void Overload() {  Console.Write("StateOverloaded tried to overload.");  }  public override void ReduceLoad() {  Console.WriteLine("State switched to \"Normal\"");  this.\_context.TransitionTo(new StateNormal());  }  public override void Connect() { Console.WriteLine("Server overloaded"); }  }  class Program {  static void Main(string[] args) {  var server = new Server(new StateNormal());  server.Request1();  server.Request3();  server.Request2();  server.Request3();  }  }  } |

|  |
| --- |
| **До задачі №5** – *Команда* |
| using System;  namespace \_5  {  abstract class Command  {  public abstract void Save();  public abstract void TrailingSave();  public abstract void ArchiveSave();  }  // конкретная команда  class ConcreteCommand : Command  {  Receiver receiver;  public ConcreteCommand(Receiver r)  {  receiver = r;  }  public override void Save()  {  receiver.Save();  }  public override void TrailingSave()  {  receiver.Trail();  receiver.Save();  }  public override void ArchiveSave()  {  receiver.Archieve();  receiver.Save();  }  }  // получатель команды  class Receiver  {  public void Save()  {  Console.WriteLine("Saved");  }  public void Trail()  {  Console.WriteLine("Trailed spaces");  }  public void Archieve()  {  Console.WriteLine("Archiving text");  }  }  // инициатор команды  class Invoker  {  Command command;  public void SetCommand(Command c)  {  command = c;  }  public void Save()  {  command.Save();  }  public void TrailingSave()  {  command.TrailingSave();  }  public void ArchiveSave()  {  command.ArchiveSave();  }  }  class Program  {  static void Main()  {  Invoker invoker = new Invoker();  Receiver receiver = new Receiver();  ConcreteCommand command = new ConcreteCommand(receiver);  invoker.SetCommand(command);  invoker.Save();  Console.WriteLine("---");  invoker.TrailingSave();  Console.WriteLine("---");  invoker.ArchiveSave();  }  }  } |

**Приклади результатів**

|  |
| --- |
| **До задачі №1 -** *Компонувальник* |
| Teacher director = new Teacher("Director: ");  Teacher dephead1 = new Teacher("Deputy Head 1: ");  director.Add(dephead1);  Teacher dephead2 = new Teacher("Deputy Head 2: ");  director.Add(dephead2);  Teacher teacher1 = new Teacher("Teacher 1: ");  Teacher teacher2 = new Teacher("Teacher 2: ");  Teacher teacher3 = new Teacher("Teacher 3: ");  dephead1.Add(teacher1);  dephead1.Add(teacher2);  dephead2.Add(teacher3);  Parent parent1\_1 = new Parent("Parent 1: ");  Parent parent2\_1 = new Parent("Parent 1: ");  Parent parent2\_2 = new Parent("Parent 2: ");  Parent parent3\_1 = new Parent("Parent 1: ");  teacher1.Add(parent1\_1);  teacher2.Add(parent2\_1);  teacher2.Add(parent2\_2);  teacher3.Add(parent3\_1);  director.GetCash();  director.Display(2); |
| --Director: 283  ----Deputy Head 1: 148  ------Teacher 1: 119  --------Parent 1: 119  ------Teacher 2: 29  --------Parent 1: 10  --------Parent 2: 19  ----Deputy Head 2: 135  ------Teacher 3: 135  --------Parent 1: 135 |

|  |
| --- |
| **До задачі №2** - *Заступник* |
| Client client = new Client();  Console.WriteLine("Without proxy");  Accounting realSubject = new Accounting();  client.ClientCode(realSubject);  Console.WriteLine();  Console.WriteLine("With proxy");  Proxy proxy = new Proxy(realSubject);  client.ClientCode(proxy); |
| Without proxy  Accounting: Forming docs.  With proxy  Proxy: Checking payment.  Accounting: Forming docs.  Proxy: Logging the time of request. |

|  |
| --- |
| **До задачі №3** – *Прототип* |
| Box p1 = new Box();  p1.count = 42;  p1.type = "Candy";  p1.IdInfo = new IdInfo(666);  Box p2 = p1.DeepCopy();  Console.WriteLine("Original values of box1, box2: ");  Console.WriteLine(" box1 instance values: ");  DisplayValues(p1);  Console.WriteLine(" box2 instance values:");  DisplayValues(p2);  p1.count = 32;  p1.type = "Toy";  p1.IdInfo.IdNumber = 7878;  Console.WriteLine("\nValues of box1, box2 after changes to box1:");  Console.WriteLine(" box1 values: ");  DisplayValues(p1);  Console.WriteLine(" box2 values: ");  DisplayValues(p2); |
| Original values of box1, box2:  box1 instance values:  Count: 42, Type: Candy  ID#: 666  box2 instance values:  Count: 42, Type: Candy  ID#: 666  Values of box1, box2 after changes to box1:  box1 values:  Count: 32, Type: Toy  ID#: 7878  box2 values:  Count: 42, Type: Candy  ID#: 666 |

|  |
| --- |
| **До задачі №4 -** *Стан* |
| var server = new Server(new StateNormal());  server.Request1();  server.Request3();  server.Request2();  server.Request3(); |
| Original values of box1, box2:  box1 instance values:  Count: 42, Type: Candy  ID#: 666  box2 instance values:  Count: 42, Type: Candy  ID#: 666  Values of box1, box2 after changes to box1:  box1 values:  Count: 32, Type: Toy  ID#: 7878  box2 values:  Count: 42, Type: Candy  ID#: 666 |

|  |
| --- |
| **До задачі №5** – *Команда* |
| Invoker invoker = new Invoker();  Receiver receiver = new Receiver();  ConcreteCommand command = new ConcreteCommand(receiver);  invoker.SetCommand(command);  invoker.Save();  Console.WriteLine("---");  invoker.TrailingSave();  Console.WriteLine("---");  invoker.ArchiveSave(); |
| Saved  ---  Trailed spaces  Saved  ---  Archiving text  Saved |

**Висновки:**

У даній роботі було розроблено програми на основі шаблонів проектування. Були застосовані такі шаблони, як: Декоратор, Проксі, Стан, Фабрика, Ланцюг обов’язків.

Декоратор було використано, як «обґортка», таким чином було реалізовано медичний огляд. Шаблон проксі або замінник було використано для перевірки наявності реєстрації.

Шаблон стану було застосовано для розроблення заміни стану об’єкту при досягненні границі потрібних балів. Шаблон фабрики задіяно для реалізації декількох видів взуття на декількох виробництвах. Паттерн ланцюг обов’язків застосовано для визначення пріоритету при поході до магазину.

Таким чином, було на практиці досліджено шаблони проектування.