|  |
| --- |
| Національний лісотехнічний університет України |
| (повне найменування вищого навчального закладу) |
| Інститут деревооброблювальних технологій і дизайну |
| (повне найменування інституту, назва факультету (відділення)) |
| Кафедра інформаційних технологій |
| (повна назва кафедри, циклової комісії) |

**КУРСОВИЙ ПРОЕКТ**

|  |  |
| --- | --- |
| з дисципліни | «Методи та засоби ООАП» |

|  |  |
| --- | --- |
| на тему: | Шаблони проектування Builder, Composite Pattern |
| та Guarded Suspension | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Студента(ки) | | 4 | | курсу | | | КН-41 | групи | |
| Напрям підготовки | | |  | | КН | | | | | |
| Манжула Давид | | | | | | | | | |
| (прізвище та ініціали) | | | | | | | | | |
| Керівник | Завідувач кафедри інформаційних | | | | | | | | |
| технологій, проф, д.т.н.Соколовський Я.І | | | | | | | | | |
| (посада, вчене звання, науковий ступінь, прізвище та ініціали) | | | | | | | | | |
| Національна шкала | | |  | | | | | | |
| Кількість балів: | | |  | | | Оцінка: ECTS | | |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Члени комісії |  |  | Соколовський Я.І |
|  | (підпис) |  | (вчене звання, науковий ступінь, прізвище та ініціали) |
|  |  |  | Процик Ю.С. |
|  | (підпис) |  | (вчене звання, науковий ступінь, прізвище та ініціали) |
|  |  |  | Яркун В.І. |
|  | (підпис) |  | (вчене звання, науковий ступінь, прізвище та ініціали) |
|  |  |  | Нечепуренко А.В. |
|  | (підпис) |  | (вчене звання, науковий ступінь, прізвище та ініціали) |

м.Львів – 2020 рік

**Вступ**

В даній курсовій роботі потрібно розробити програмне забезпечення шаблонів проектування Builder (Будівельник), Composite Pattern (Компонувальник) і Guarded Suspension (Охоронна призупинка), відповідно до прикладів заданих у книжках Марка Гранда і Фрімена. Буде проведено опис шаблонів, а також буде розроблено UML-діаграми відповідно до кожного шаблону.

**Зміст**

[Технічне завдання 5](#_Toc500453933)

[1. Стан проблемної області 6](#_Toc500453934)

[1.1 Опис шаблону Builder (Будівельник) 6](#_Toc500453935)

[1.1.1 Призначення шаблону 6](#_Toc500453936)

[1.1.2 Застосування шаблону 6](#_Toc500453937)

[1.1.3 Структура шаблону 6](#_Toc500453938)

[1.1.4 Зв'язані шаблони відносно даного шаблону 6](#_Toc500453939)

[1.2 Опис шаблону Guarded Suspension (Охоронна призупинка) 7](#_Toc500453940)

[1.2.1 Призначення шаблону 7](#_Toc500453941)

[1.2.2 Застосування шаблону 7](#_Toc500453942)

[1.2.3 Зв'язані шаблони відносно даного шаблону 8](#_Toc500453943)

[1.3 Опис шаблону Composite (Компонувальник) 8](#_Toc500453944)

[1.3.1 Призначення шаблону 8](#_Toc500453945)

[1.3.2 Застосування шаблону 8](#_Toc500453946)

[1.3.3 Структура шаблону 8](#_Toc500453947)

[1.3.4 Опис шаблону 9](#_Toc500453948)

[2. Проектування програмного забезпечення. UML діаграми. 10](#_Toc500453949)

[2.1 Опис шаблону Builder (Будівельник) 10](#_Toc500453950)

[2.1.1 Структура шаблону 10](#_Toc500453951)

[2.2 Опис шаблону Guarded Suspension (Охоронна призупинка) 11](#_Toc500453952)

[2.2.1 Структура шаблону 11](#_Toc500453953)

[2.3 Опис шаблону Composite (Компонувальник) 12](#_Toc500453954)

[2.3.1 Структура шаблону 12](#_Toc500453955)

[3. Реалізація програмного забезпечення 13](#_Toc500453956)

[3.1 Представлення шаблону Builder (Будівельник) 13](#_Toc500453957)

[3.1.1 Класи 13](#_Toc500453958)

[3.2 Представлення шаблону Guarded Suspension (Охоронна призупинка) 14](#_Toc500453959)

[3.2.1 Класи Клас для роботи з чергою пацієнтів додавання та видалення пацієнтів. 14](#_Toc500453960)

[3.3 Представлення шаблону Composite (Компонувальник) 15](#_Toc500453961)

[3.3.1 Класи 15](#_Toc500453962)

[4. Аналіз результатів роботи програмного забезпечення 17](#_Toc500453963)

[4.1. Аналіз результатів роботи ПЗ на основі шаблону Builder 17](#_Toc500453964)

[4.2. Аналіз результатів роботи ПЗ на основі шаблону Composite Pattern 18](#_Toc500453965)

[4.3. Аналіз результатів роботи ПЗ на основі шаблону Guarded Suspension 20](#_Toc500453966)

[Висновки 21](#_Toc500453967)

[Список використаних джерел 22](#_Toc500453968)

[Додаток 1 23](#_Toc500453969)

[Додаток 2 27](#_Toc500453970)

[Додаток 3 29](#_Toc500453971)

# Технічне завдання

Міністерство освіти і науки України

Національний лісотехнічний університет України

Кафедра інформаційних технологій

**Завдання для курсового проектування**

з дисципліни «Методи та засоби ООАП»

Студент: **Манжула Давид**

1. Навести опис технічного завдання для шаблонів Builder (Будівельник), Guarded Suspension (Охоронна призупинка) з розділу «Контекст» (Марк Гранд).
2. Навести опис технічного завдання для шаблону Composite Pattern (Компонувальник) (Е.Фрімен).
3. Навести опис шаблонів проектування згідно прийнятої схеми:

* Назва та класифікація патерну;
* Призначення;
* Інша назва (за наявності);
* Застосування;
* Структура;
* Учасники;
* Відношення;
* Результати;
* Реалізація;
* Відомі застосування;
* Інші зв’язані шаблони відносно даного шаблону.

1. Відповідно до п. 1 і 2 розробити необхідні діаграми UML (діаграми варіантів використання, діаграми класів, діаграми взаємодій, діаграми станів) для ілюстрації шаблонів та проектування програмної системи.
2. Розробити програмне забезпечення згідно пункту 1 і пункту 2 з використанням вказаних шаблонів у C# та проаналізувати результати роботи програмної системи з використанням шаблонів.

**Календарний план виконання курсового проекту**

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Назва етапу роботи | Строк виконання | Відмітка про виконання |
| 1 | Деталізація технічного завдання; побудова UML діаграм шаблонів. |  |  |
| 2 | Програмна реалізація поставленого завдання. |  |  |
| 3 | Налагодження та тестування розробленого застосунку. |  |  |
| 4 | Оформлення пояснювальної записки та презентації. |  |  |

Дата видачі завдання \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Керівник роботи \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (посада, П. І. Б.)

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (П. І. Б.)

# 1. Стан проблемної області

## 1.1 Опис шаблону Builder (Будівельник)

## 1.1.1 Призначення шаблону

Шаблон Builder дозволяє об’єкту клієнта створювати складний об’єкт, задаючи для нього тільки тип і вміст. Клієнт позбавляється від інформації про деталі конструювання об’єкта.

## 1.1.2 Застосування шаблону

Завдання вмістимості і побудови певного представлення даних не залежить один від одного. Представлення даних продукту може змінюватись, ніяк не впливаючи на об’єкти, презентуючи вміст. Об’єкти будівельники можуть працювати з різними об’єктами, що показують вміст, не потрибуючи при цьому будь, яких змін.

Шаблон Builder забезпечує більш точний контроль під час конструювання, а ніж інші шаблони (наприклад Factory Method), даючи можливість об’єкту курівника крок за кроком контролювати весь процес створення об’єкта продукту. Другі шаблони просто створюють зразу весь об’єкт.

### 1.1.3 Структура шаблону

На рисунку 1.1.3.1 зображена структура шаблону Builder:

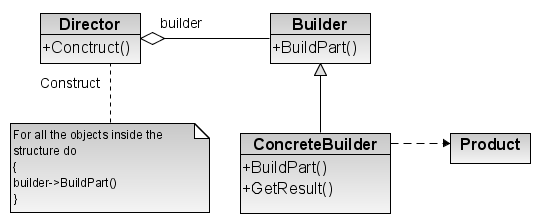


Рисунок 1.1.3.1 Структура шаблону Builder

### 1.1.4 Зв'язані шаблони відносно даного шаблону

Interface. Шаблон Builder використовує шаблон Interface для відкриття класу об’єкта ProductIF.

Composite. Об’єкт, створений за допомогою шаблона Builder, зазвичай є основним.

Factory Method. Шаблон Builder використовує шаблон Factory Method для прийняття рішень, який клас конкретного будівельника вписати.

Template Method. При реалізації класу Abstract Builder часто використовується шаблон Template Method.

Null Object. Шаблон Null може використовуватись шаблоном Builder для отримання даних з незадіяних методів.

Visitor. Шаблон Visitor дозволяє об’єкту клієнта бути більш тісно зв’язними з побудовою нового складного об’єкта, це допускає шаблон Builder. Замість опису вмістимого створюваних об’єктів за допомогою багатьох звернень до методів інформації, представляється цілком, як комплексна структура даних.

## 1.2 Опис шаблону Guarded Suspension (Охоронна призупинка)

### 1.2.1 Призначення шаблону

Якщо існує умова, яка забороняє методу виконувати його функції, то даний шаблон призупиняє виконання цього методу до того моменту, поки умова не перестане діяти.

### 1.2.2 Застосування шаблону

Даний шаблон застосовується коли:

* Потрібно вирішувати проблеми обєкта, стан якого не дозволяє виконати деякі операції.
* Використовувати синхронізувати методи чи команди.

### 1.2.3 Зв'язані шаблони відносно даного шаблону

Balking. Шаблон Balking представляє другу стратегію обробки звернень до методів об’єкта, стан яких не дозволяє виконати ці методи.

Two-Phase Termination. Реалізація шаблону Two-Phase Termination зазвичай представляє генерацію і обробку виключених ситуацій Interrupte-Exception. В його реалізацію, як правило, присутня взаємодія з шаблоном Guarded Suspension.

## 1.3 Опис шаблону Composite (Компонувальник)

## 1.3.1 Призначення шаблону

Шаблон Compositeдозволяє створювати складні об’єкти за принципом рекурсивного об’єднання подібних об’єктів у вигляді дерева. Крім того, цей шаблон дає змогу керувати цими об’єктами дерева, але при наявності у цих об’єктів загального інтерфейсу чи суперкласу.

## 1.3.2 Застосування шаблону

Шаблон застосовується якщо :

* Потрібно получити доступ до головного об’єкта, у структурованому вигляді дерева.
* Клієнтські об’єкти класу можуть розглянути його просто як клас.
* Якщо клієнт викликатимете метод об’єкта ComponentIF.
* Якщо об’єкт ComponentIF потрібно використати, як об’єкт.

### 1.3.3 Структура шаблону

На рисунку 1.3.3.1 показано структуру шаблону Composite.

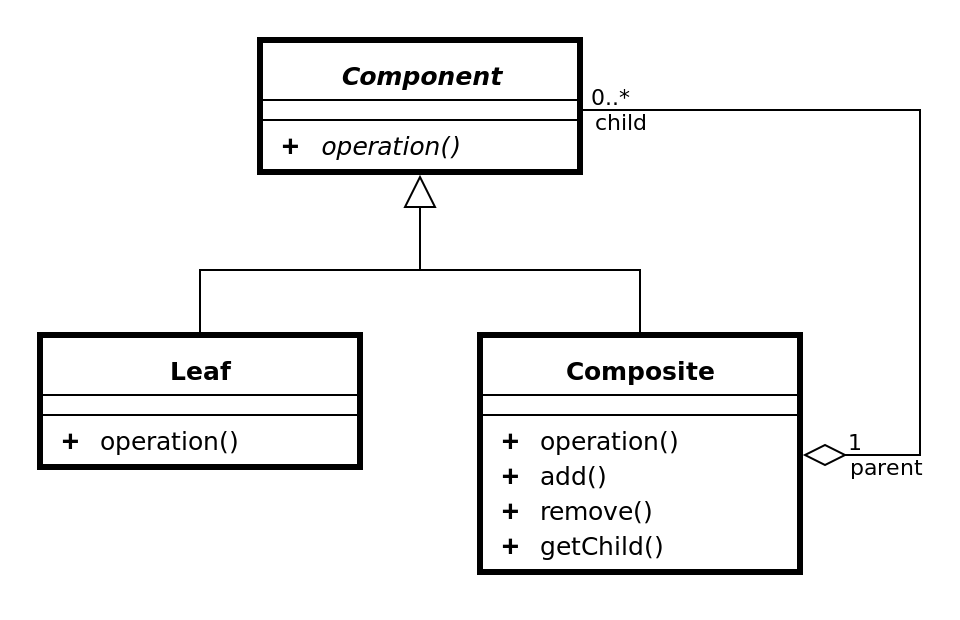


Рисунок 1.3.3.1 Структура шаблону Composite

### 1.3.4 Опис шаблону

Шаблони проектування, які зв'язані з шаблоном CompositePattern:

Chain of Responsibility. Шаблон Chain of Responsibility може використовуватися разом з шаблоном Composite.

Low Coupling/High Cohesion. Шаблон Low Coupling/High Cohesion не рекомендує розміщати спеціальні методи в класі загального значення.

Visitor. Можна використовувати шаблон Visitor для інкапсуляції в одному класі операцій.

# 2. Проектування програмного забезпечення. UML діаграми.

## 2.1 Опис шаблону Builder (Будівельник)

### 2.1.1 Структура шаблону

На рисунку 2.1.1.1 можна побачити UML-діаграму шаблону Builder. На діаграмі зображено, які інтерфейси і класи були використані у програмі.



Рисунок 2.1.1.1 UML-діаграма шаблону

## 2.2 Опис шаблону Guarded Suspension (Охоронна призупинка)

### 2.2.1 Структура шаблону

На рисунку 2.2.1.1 можна побачити UML-діаграму класів які представляють шаблон GuardedSuspension.

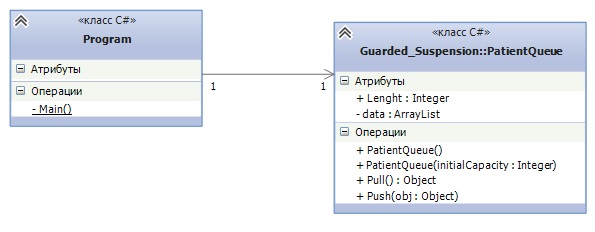


Рисунок 2.2.1.1 UML-діаграма шаблону GuardedSuspension

## 2.3 Опис шаблону Composite (Компонувальник)

## 2.3.1 Структура шаблону

На рисунку 2.3.1.1 можна побачити UML-діаграму шаблону Composite.

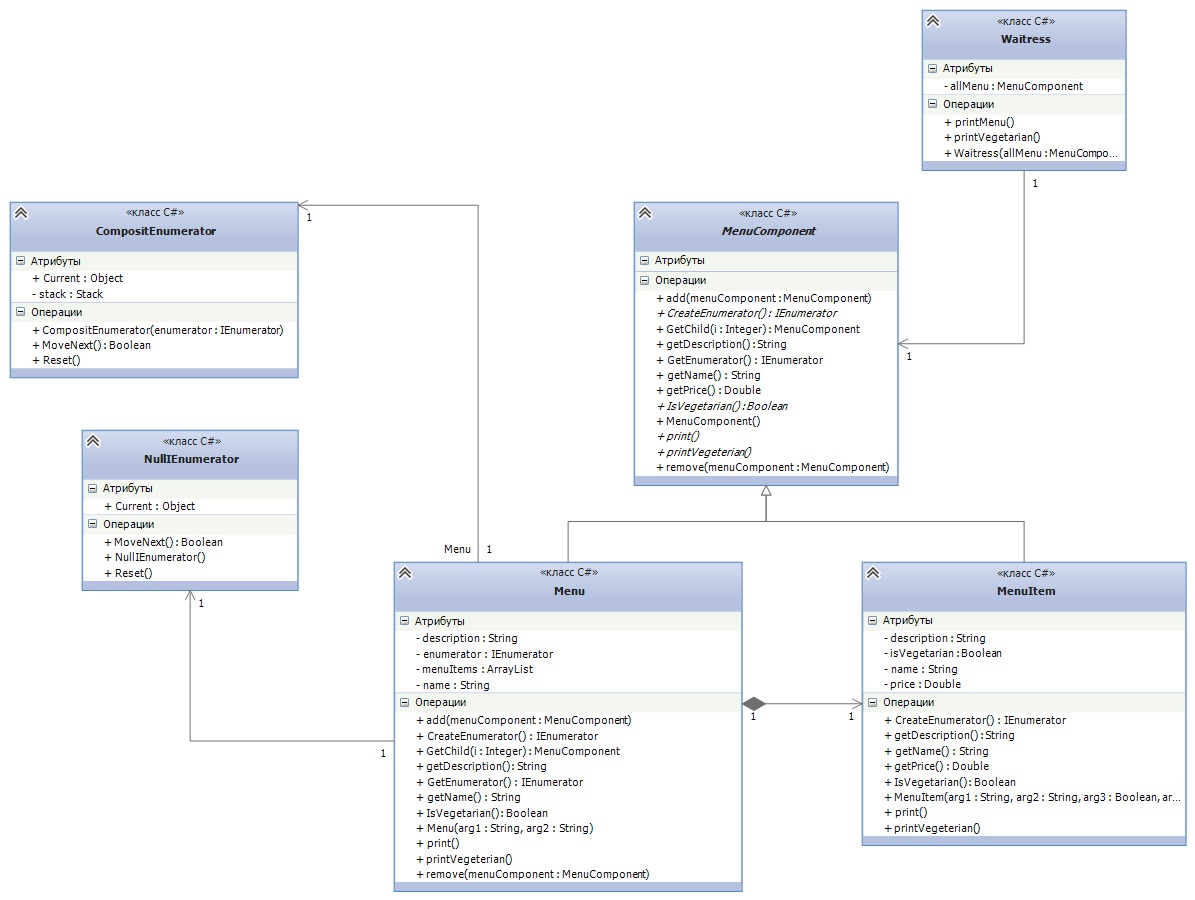


Рисунок 2.3.1.1 UML-діаграма шаблону

# 3. Реалізація програмного забезпечення

## 3.1 Представлення шаблону Builder (Будівельник)

## 3.1.1 Класи

Класи для перетворення повідомлень.

class PROFSMsg: OutboundMessageIF

{

public String Message { get; set; }//текст повідомлення

public String Receiver { get; set; }//електронна адреса одержувача

public String Sender { get; set; }//електронна адреса відправника

public override string ToString()

{

string str = "Receiver: ";

str += Receiver + System.Environment.NewLine;

str += "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" + System.Environment.NewLine;

str += "Sender: ";

str += Sender + System.Environment.NewLine;

str += "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" + System.Environment.NewLine;

str += Message + System.Environment.NewLine;

str += "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_";

return str;

}

}

class MAPIMsg: OutboundMessageIF

{

public String Message { get; set; }//текст повідомлення

public String Receiver { get; set; }//електронна адреса одержувача

public String Sender { get; set; }//електронна адреса відправника

public override string ToString()

{

string str = "Повідомлення адресовано до: ";

str += Receiver + System.Environment.NewLine;

str += "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";

str += Message + System.Environment.NewLine;

str += "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";

str += System.Environment.NewLine;

str += "Повідомлення було надіслано від: ";

str += Sender;

return str;

}

}

Клас для повідомлень.

class MIMEMsg {

public String Message { get; set; }//текст повідомлення в форматі MIME

public int Lenght { get { return Message.Length; } }

public MIMEMsg(string arg1, string arg2, string arg3)

{

Message =arg1+"/ "+arg2+"/ "+System.Environment.NewLine+arg3;

}

public override string ToString()

{

return Message;

} }

## 3.2 Представлення шаблону Guarded Suspension (Охоронна призупинка)

### 3.2.1 Класи Клас для роботи з чергою пацієнтів додавання та видалення пацієнтів.

public class PatientQueue

{

private ArrayList data;

public int Lenght { get { return data.Count; } }

/// <summary>

/// Конструктор створює обєкт <c>PatientQueue</c>.

/// </summary>

public PatientQueue()

{

data = new ArrayList();

}

/// <summary>

/// Параметризований конструктор

/// створює обєкт <c>PatientQueue</c>.

/// </summary>

/// <param name="initialCapacity">

/// Ініціалізація пропускної здатності внутрішньої структури даних для черги.

/// </param>

public PatientQueue(int initialCapacity)

{

data = new ArrayList(initialCapacity);

}

/// <summary>

/// Поміщає обєкт в кінець черги

/// </summary>

/// <param name="obj">

/// Обєкт для розміщення в черзі

/// </param>

public void Push(object obj)

{

lock (data)

{

data.Add(obj);

Monitor.Pulse(data);

}

}

/// <summary>

/// Дістає та видаляє обєкт з початку черги

/// </summary>

/// <returns>

/// Повертає видобутий з черги обєкт

/// </returns>

/// <remarks>

/// Якщо черга порожня даний метод чекає поки інший

/// потік помістить обєкт в чергу і

/// після цього повертає цей обєкт

/// </remarks>

public object Pull()

{

lock (data)

{

while (data.Count == 0)

{

Monitor.Wait(data);

}

}

object obj = data[0];

data.RemoveAt(0);

return obj;

}

}

## 3.3 Представлення шаблону Composite (Компонувальник)

### 3.3.1 Класи

Даний клас описує пункт меню.

class MenuItem : MenuComponent

{

string name;

string description;

bool isVegetarian;

double price;

public MenuItem(string arg1, string arg2, bool arg3, double arg4)

{

name = arg1;

description = arg2;

isVegetarian = arg3;

price = arg4;

}

public override string getName() { return name; }

public override string getDescription() { return description; }

public override bool IsVegetarian() { return isVegetarian; }

public override double getPrice(){return price;}

public override void print()

{

Console.Write(" " + getName());

if (isVegetarian) Console.Write(" (вегетерiанська страва)");

Console.WriteLine(", " + getPrice());

Console.WriteLine(" - - - " + getDescription());

}

public override void printVegeterian()

{

if (isVegetarian)

{

Console.Write(" " + getName());

Console.Write(" (вегетерiанська страва)");

Console.WriteLine(", " + getPrice());

Console.WriteLine(" - - - " + getDescription());

}

}

public override IEnumerator CreateEnumerator()

{

return new NullIEnumerator();

}

}

Клас меню описує ціле меню і всі страви які в ньому знаходяться.

class Menu : MenuComponent

{

IEnumerator enumerator = null;

ArrayList menuItems = new ArrayList();

string name;

string description;

public Menu(string arg1, string arg2)

{

name = arg1;

description = arg2;

}

public override IEnumerator CreateEnumerator()

{

if (enumerator == null)

enumerator = new CompositEnumerator(menuItems.GetEnumerator());

return enumerator;

}

public override void add(MenuComponent menuComponent)

{

menuItems.Add(menuComponent);

}

public override void remove(MenuComponent menuComponent)

{

menuItems.Remove(menuComponent);

}

public override MenuComponent GetChild(int i)

{

return (MenuComponent)menuItems[i];

}

public override string getName()

{

return name;

}

public override string getDescription()

{

return description;

}

public override void print() {

Console.Write("\n "+getName());

Console.Write(", "+ getDescription());

Console.WriteLine("\n----------------------------------------------");

enumerator = menuItems.GetEnumerator();

while (enumerator.MoveNext())

{

MenuComponent component = (MenuComponent)enumerator.Current;

component.print();

}

}

public override void printVegeterian() {

enumerator = menuItems.GetEnumerator();

while (enumerator.MoveNext())

{

MenuComponent component = (MenuComponent)enumerator.Current;

if (component.IsVegetarian())

component.printVegeterian();

}

}

public override IEnumerator GetEnumerator() {

enumerator= new CompositEnumerator(menuItems.GetEnumerator());

return enumerator;

}

public override bool IsVegetarian() {

enumerator = menuItems.GetEnumerator();

bool flag = false;

while (enumerator.MoveNext())

{

MenuComponent component = (MenuComponent)enumerator.Current;

if (component.IsVegetarian())

{

flag = true;

break;

}

}

return flag;

} }

# 4. Аналіз результатів роботи програмного забезпечення

### 4.1. Аналіз результатів роботи ПЗ на основі шаблону Builder

**4.1.2 Користувацький інтерфейс**

На рисунку 4.1.2.1 показано користувацький інтерфейс програми, де знаходяться кнопки для перетворення інформації та вказівки параметрів.

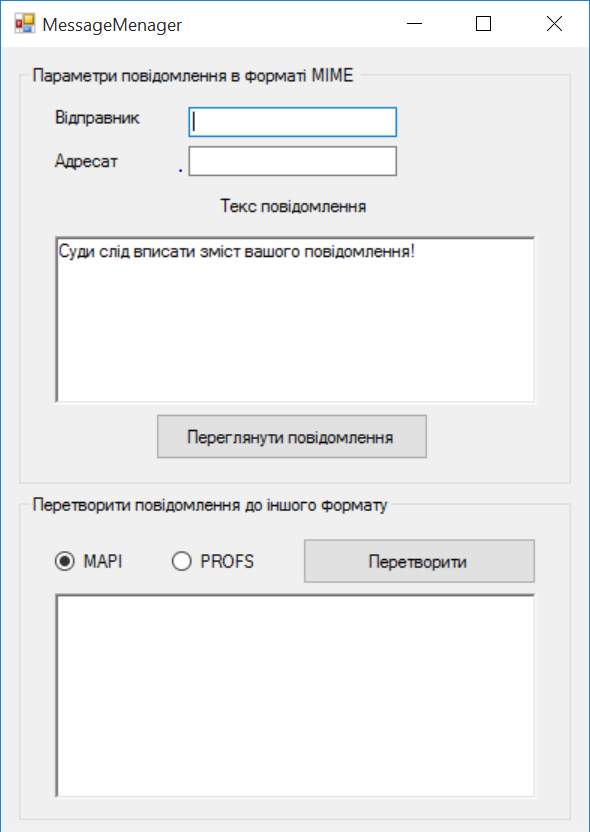


Рисунок 4.1.2.1 Інтерфейс програми

На рисунку 4.1.2.2 зображено, як виглядає перетворення повідомлення у різні формати.

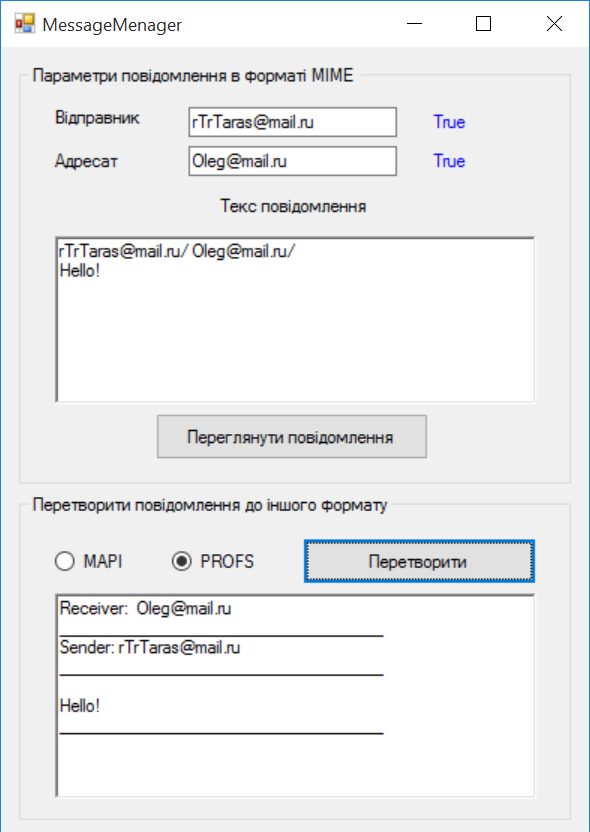


Рисунок 4.1.2.2 Інтерфейс програми

### 4.2. Аналіз результатів роботи ПЗ на основі шаблону Composite Pattern

На рисунку 4.2.2.1 зображено користувацький інтерфейс програми, що демонструє роботу черги.

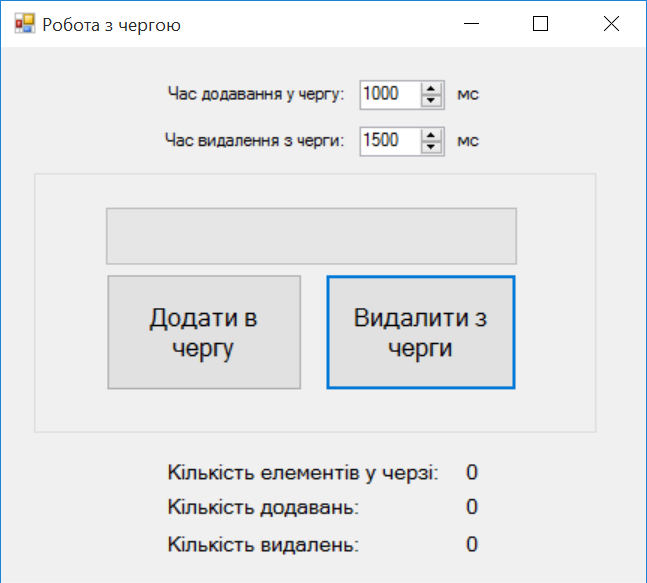


Рисунок 4.2.2.1 Інтерфейс програми

На рисунку 4.2.2.2 зображено роботу черги, додавання та віднімання черги, а також часові параметри.

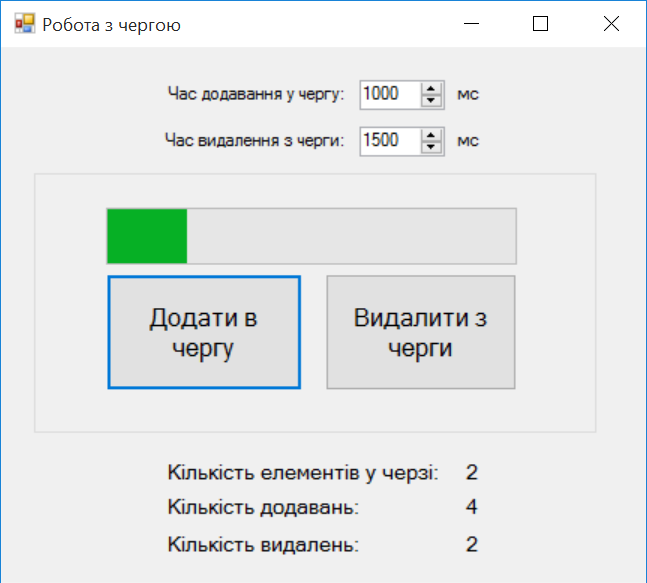


Рисунок 4.2.2.2 Інтерфейс програми

### 4.3. Аналіз результатів роботи ПЗ на основі шаблону Guarded Suspension

На рисунку 4.3.1 зображено користувацький інтерфейс з результатом роботи програми:

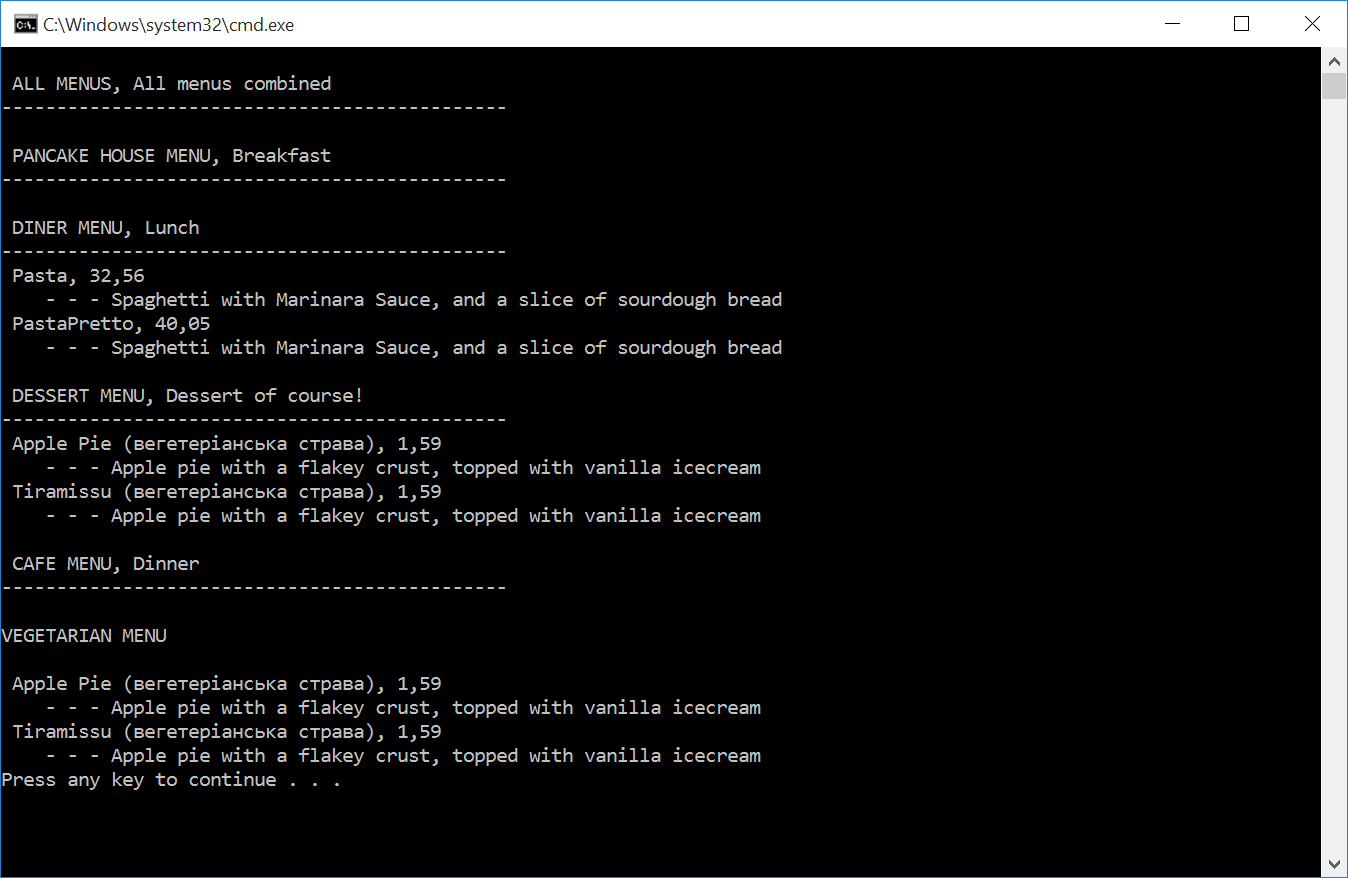


Рисунок 4.3.1 Користувацький інтерфейс програми

# Висновки

В даному курсовому проекті було розроблено програмне забезпечення відповідного до шаблонів проектування, які були задані у завданні для виконання роботи – Builder (Будівельник), Guarded Suspension (Охоронна призупинка) і Composite (Компонувальник). Також було описано застосування шаблонів і пов’язані шаблони із вказаними.

# Список використаних джерел

1. Соколовський Я.І. Методи та засоби ООАП. Конспект лекцій. Електронний варіант. – 2017.
2. Яркун В.І., Нечепуренко А.В., Герасимчук О.П.. Методичні вказівки до виконання лабораторних робіт. Електронний варіант. - 2016р.
3. Соколовський Я.І., Герасимчук О.П., Яркун В.І. Методичні вказівки та завдання до курсового проектування з дисципліни "Методи та засоби ООАП" для студентів напряму "Комп’ютерні науки". Електронний варіант. Львів, НЛТУ України. – 2017. – с. 36.
4. Марк Гранд /  Шаблоны проектирования в JAVA. Каталог популярных шаблонов проектирования, проиллюстрированных при помощи UML = Patterns in Java, Volume 1. A Catalog of Reusable Design Patterns Illustrated with UML. —  М.: [«Новое знание»](https://ru.wikipedia.org/w/index.php?title=%D0%9D%D0%BE%D0%B2%D0%BE%D0%B5_%D0%B7%D0%BD%D0%B0%D0%BD%D0%B8%D0%B5_(%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE)&action=edit&redlink=1), 2004. — С. 560.
5. Фримен Эр., Фримен Эл., Сьерра К., Бейтс Б. Паттерны проектирования (2011).
6. Э.Гамма, Р.Хелм, Р.Джонсон, Дж.Влиссидес Приемы объектно-ориентированного проектирования. Паттерны проектирования = Design Patterns: Elements of Reusable Object-Oriented Software. — СПб: [«Питер»](https://ru.wikipedia.org/wiki/%D0%9F%D0%B8%D1%82%D0%B5%D1%80_(%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE)), 2007. — С. 366.
7. [Шевчук А.](http://padabum.com/search.php?author=%D0%A8%D0%B5%D0%B2%D1%87%D1%83%D0%BA%20%D0%90.) [Охрименко Д.](http://padabum.com/search.php?author=%D0%9E%D1%85%D1%80%D0%B8%D0%BC%D0%B5%D0%BD%D0%BA%D0%BE%20%D0%94.) Design Patterns via C#. Приемы объектно-ориентированного проектирования . – СПб: Питер. – 2015. 288с.
8. Сергей Тепляков / Паттерны проектирования на платформе .NET. - СПб.: Питер, 2015. – 320с.: ил.
9. К. Ларман / Применение UML 2.0 и шаблонов проектирования. Введение в объектно-ориентированный анализ, проектирование и итеративную разработку. 2007. – 727с.
10. Г.Буч объектно-ориентированное проектирование с примерами применения. 2008. – 718с.
11. Литвин В.В., Шаховська Н.Б. Проектування інформаційних систем. Львів: Магнолія-2006, 2011. — 380 с.
12. Андрій Будай Дизайн патерни - просто, як двері, Львів: - 2012, 90 ст.

## Додаток 1

Лістинг програми – Builder

MAPIMsg.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Builder

{

class MAPIMsg: OutboundMessageIF

{

public String Message { get; set; }//текст повідомлення

public String Receiver { get; set; }//електронна адреса одержувача

public String Sender { get; set; }//електронна адреса відправника

public override string ToString()

{

string str = "Повідомлення адресовано до: ";

str += Receiver + System.Environment.NewLine;

str += "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";

str += Message + System.Environment.NewLine;

str += "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";

str += System.Environment.NewLine;

str += "Повідомлення було надіслано від: ";

str += Sender;

return str;

}

}

}

MessageBuilder.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Builder

{

abstract class MessageBuilder

{

//перетворений обєкт повідомлення

public OutboundMessageIF messageObject { get; set; }

//отримати обєкт будівельника

public static MessageBuilder GetInstance(String format)

{

MessageBuilder builder = null;

if (format.Equals("MAPI"))

builder = new MAPIBuilder();

else if (format.Equals("PROFS"))

builder = new PROFSBuilder();

return builder;

}

//абстрактні методи для побудови обєкту повідомлення

public abstract void From(string Value);

public abstract void To(string Value);

public abstract void PlainText(string Value);

}

class MAPIBuilder : MessageBuilder

{

public MAPIBuilder()

{

messageObject = new MAPIMsg();

}

public override void From(string msg)

{

messageObject.Sender = msg;

}

public override void To(string msg)

{

messageObject.Receiver = msg;

}

public override void PlainText(string msg)

{

messageObject.Message = msg;

}

}

class PROFSBuilder : MessageBuilder

{

public PROFSBuilder()

{

messageObject = new PROFSMsg();

}

public override void From(string msg)

{

messageObject.Sender =msg;

}

public override void To(string msg)

{

messageObject.Receiver = msg;

}

public override void PlainText(string msg)

{

messageObject.Message = msg;

}

}

}

MIMEMsg.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Builder

{

class MIMEMsg

{

public String Message { get; set; }//текст повідомлення в форматі MIME

public int Lenght { get { return Message.Length; } }

public MIMEMsg(string arg1, string arg2, string arg3)

{

Message =arg1+"/ "+arg2+"/ "+System.Environment.NewLine+arg3;

}

public override string ToString()

{

return Message;

}

}

}

MIMEParser.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Builder

{

class MIMEParser

{

public MIMEMsg objectToParse { get; set; }

public MessageBuilder builder { get; set; }

public MIMEParser(MIMEMsg ob)

{

objectToParse = ob;

}

public OutboundMessageIF parse(string format)

{

builder = MessageBuilder.GetInstance(format);

string []param=objectToParse.Message.Split('/');

builder.From(param[0]);

builder.To(param[1]);

builder.PlainText(param[2]);

return builder.messageObject;

}

}

}

OutboundMessageIF.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Builder

{

interface OutboundMessageIF

{

String Message { get; set; }//текст повідомлення

String Receiver { get; set; }//електронна адреса одержувача

String Sender { get; set; }//електронна адреса відправника

}

}

PROFSMsg.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Builder

{

class PROFSMsg: OutboundMessageIF

{

public String Message { get; set; }//текст повідомлення

public String Receiver { get; set; }//електронна адреса одержувача

public String Sender { get; set; }//електронна адреса відправника

public override string ToString()

{

string str = "Receiver: ";

str += Receiver + System.Environment.NewLine;

str += "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" + System.Environment.NewLine;

str += "Sender: ";

str += Sender + System.Environment.NewLine;

str += "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" + System.Environment.NewLine;

str += Message + System.Environment.NewLine;

str += "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_";

return str;

}

}

}

Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Windows.Forms;

namespace Builder

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}

## Додаток 2

Лістинг програми – Guarded Suspension

Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Windows.Forms;

namespace Guarded\_Suspension

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}

Queue.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Collections;

using System.Threading;

namespace Guarded\_Suspension

{

/// <summary>

/// A queue class that handles a attempt to remove an object

/// from an empty queue by waiting for another thread to put

/// an object in the queue.

/// </summary>

public class PatientQueue

{

private ArrayList data;

public int Lenght { get { return data.Count; } }

/// <summary>

/// Конструктор створює обєкт <c>PatientQueue</c>.

/// </summary>

public PatientQueue()

{

data = new ArrayList();

}

/// <summary>

/// Параметризований конструктор

/// створює обєкт <c>PatientQueue</c>.

/// </summary>

/// <param name="initialCapacity">

/// Ініціалізація пропускної здатності внутрішньої структури даних для черги.

/// </param>

public PatientQueue(int initialCapacity)

{

data = new ArrayList(initialCapacity);

}

/// <summary>

/// Поміщає обєкт в кінець черги

/// </summary>

/// <param name="obj">

/// Обєкт для розміщення в черзі

/// </param>

public void Push(object obj)

{

lock (data)

{

data.Add(obj);

Monitor.Pulse(data);

}

}

/// <summary>

/// Дістає та видаляє обєкт з початку черги

/// </summary>

/// <returns>

/// Повертає видобутий з черги обєкт

/// </returns>

/// <remarks>

/// Якщо черга порожня даний метод чекає поки інший

/// потік помістить обєкт в чергу і

/// після цього повертає цей обєкт

/// </remarks>

public object Pull()

{

lock (data)

{

while (data.Count == 0)

{

Monitor.Wait(data);

}

}

object obj = data[0];

data.RemoveAt(0);

return obj;

}

}

}

## Додаток 3

Лістинг програми - Composite

CompositIterator.cs

using System;

using System.Collections.Generic;

using System.Collections;

using System.Linq;

using System.Text;

namespace CompositePattern

{

class CompositEnumerator : IEnumerator

{

Stack stack = new Stack();

public CompositEnumerator(IEnumerator enumerator)

{

stack.Push(enumerator);

}

public object Current

{

get

{

if (MoveNext())

{

IEnumerator enumerator = (IEnumerator)stack.Peek();

MenuComponent component = (MenuComponent)enumerator.Current;

if (component is Menu)

{

stack.Push(((Menu)component).CreateEnumerator());

}

return component;

}

else return null;

}

}

public bool MoveNext()

{

if (stack.Count == 0)

return false;

else

{

IEnumerator enumerator = (IEnumerator)stack.Peek();

if (!enumerator.MoveNext())

{

stack.Pop();

return MoveNext();

}

else return true;

}

}

public void Reset()

{

throw new ApplicationException();

}

}

}

Menu.cs

using System;

using System.Collections.Generic;

using System.Collections;

using System.Linq;

using System.Text;

namespace CompositePattern

{

class Menu : MenuComponent

{

IEnumerator enumerator = null;

ArrayList menuItems = new ArrayList();

string name;

string description;

public Menu(string arg1, string arg2)

{

name = arg1;

description = arg2;

}

public override IEnumerator CreateEnumerator()

{

if (enumerator == null)

enumerator = new CompositEnumerator(menuItems.GetEnumerator());

return enumerator;

}

public override void add(MenuComponent menuComponent)

{

menuItems.Add(menuComponent);

}

public override void remove(MenuComponent menuComponent)

{

menuItems.Remove(menuComponent);

}

public override MenuComponent GetChild(int i)

{

return (MenuComponent)menuItems[i];

}

public override string getName()

{

return name;

}

public override string getDescription()

{

return description;

}

public override void print()

{

Console.Write("\n "+getName());

Console.Write(", "+ getDescription());

Console.WriteLine("\n----------------------------------------------");

enumerator = menuItems.GetEnumerator();

while (enumerator.MoveNext())

{

MenuComponent component = (MenuComponent)enumerator.Current;

component.print();

}

}

public override void printVegeterian()

{

enumerator = menuItems.GetEnumerator();

while (enumerator.MoveNext())

{

MenuComponent component = (MenuComponent)enumerator.Current;

if (component.IsVegetarian())

component.printVegeterian();

}

}

public override IEnumerator GetEnumerator()

{

enumerator= new CompositEnumerator(menuItems.GetEnumerator());

return enumerator;

}

public override bool IsVegetarian()

{

enumerator = menuItems.GetEnumerator();

bool flag = false;

while (enumerator.MoveNext())

{

MenuComponent component = (MenuComponent)enumerator.Current;

if (component.IsVegetarian())

{

flag = true;

break;

}

}

return flag;

}

}

}

MenuComponent.cs

using System;

using System.Collections.Generic;

using System.Collections;

using System.Linq;

using System.Text;

namespace CompositePattern

{

abstract class MenuComponent

{

public virtual void add(MenuComponent menuComponent)

{

throw new ApplicationException("Даний метод не підтримується!");

}

public virtual void remove(MenuComponent menuComponent)

{

throw new ApplicationException("Даний метод не підтримується!");

}

public virtual MenuComponent GetChild(int i)

{

throw new ApplicationException("Даний метод не підтримується!");

}

public virtual string getName()

{

throw new ApplicationException("Даний метод не підтримується!");

}

public virtual string getDescription()

{

throw new ApplicationException("Даний метод не підтримується!");

}

public virtual double getPrice()

{

throw new ApplicationException("Даний метод не підтримується!");

}

public abstract bool IsVegetarian();

//{

// throw new ApplicationException("Даний метод не підтримується!");

// //return false;

//}

public abstract void print();

public abstract void printVegeterian();

public virtual IEnumerator GetEnumerator() { return null; }

public abstract IEnumerator CreateEnumerator();

}

}

MenuItem.cs

using System;

using System.Collections.Generic;

using System.Collections;

using System.Linq;

using System.Text;

namespace CompositePattern

{

class MenuItem : MenuComponent

{

string name;

string description;

bool isVegetarian;

double price;

public MenuItem(string arg1, string arg2, bool arg3, double arg4)

{

name = arg1;

description = arg2;

isVegetarian = arg3;

price = arg4;

}

public override string getName() { return name; }

public override string getDescription() { return description; }

public override bool IsVegetarian() { return isVegetarian; }

public override double getPrice(){return price;}

public override void print()

{

Console.Write(" " + getName());

if (isVegetarian) Console.Write(" (вегетерiанська страва)");

Console.WriteLine(", " + getPrice());

Console.WriteLine(" - - - " + getDescription());

}

public override void printVegeterian()

{

if (isVegetarian)

{

Console.Write(" " + getName());

Console.Write(" (вегетерiанська страва)");

Console.WriteLine(", " + getPrice());

Console.WriteLine(" - - - " + getDescription());

}

}

public override IEnumerator CreateEnumerator()

{

return new NullIEnumerator();

}

}

}

NullIEnumerator.cs

using System;

using System.Collections.Generic;

using System.Collections;

using System.Linq;

using System.Text;

namespace CompositePattern

{

class NullIEnumerator: IEnumerator

{

public object Current { get { return null; } }

public bool MoveNext()

{

return false;

}

public void Reset()

{

throw new ApplicationException();

}

}

}

Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace CompositePattern

{

class Program

{

static void Main(string[] args)

{

MenuComponent pancakeHouseMenu = new Menu("PANCAKE HOUSE MENU","Breakfast");

MenuComponent dinerMenu = new Menu("DINER MENU", "Lunch");

MenuComponent cafeMenu = new Menu("CAFE MENU", "Dinner");

MenuComponent dessertMenu = new Menu("DESSERT MENU","Dessert of course!");

MenuComponent allMenus = new Menu("ALL MENUS", "All menus combined");

allMenus.add(pancakeHouseMenu);

allMenus.add(dinerMenu);

allMenus.add(cafeMenu);

dinerMenu.add(new MenuItem( "Pasta",

"Spaghetti with Marinara Sauce, and a slice of sourdough bread",

false, 32.56));

dinerMenu.add(new MenuItem("PastaPretto",

"Spaghetti with Marinara Sauce, and a slice of sourdough bread",

false, 40.05));

dinerMenu.add(dessertMenu);

dessertMenu.add(new MenuItem("Apple Pie",

"Apple pie with a flakey crust, topped with vanilla icecream",

true,1.59));

dessertMenu.add(new MenuItem("Tiramissu",

"Apple pie with a flakey crust, topped with vanilla icecream",

true, 1.59));

Waitress waitress = new Waitress(allMenus);

waitress.printMenu();

waitress.printVegetarian();

}

}

}

Waitress.cs

using System;

using System.Collections.Generic;

using System.Collections;

using System.Linq;

using System.Text;

namespace CompositePattern

{

class Waitress

{

MenuComponent allMenu;

public Waitress(MenuComponent allMenu)

{

this.allMenu = allMenu;

}

public void printMenu()

{

allMenu.print();

}

public void printVegetarian()

{

Console.WriteLine("\nVEGETARIAN MENU\n");

allMenu.printVegeterian();

}

}

}