НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КПІ»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра програмного забезпечення комп’ютерних систем

ЛАБОРАТОРНА РОБОТА №4

*з дисципліни*

*«Об’єктно-орієнтоване програмування»*

ТЕМА: «Реалізація поведінкових шаблонів проектування»

Підготував: студент групи КП-51

Бабенко Валерій Павлович

Перевірила:

Заболотня Тетяна Миколаївна

Київ 2017

# Мета роботи

# Ознайомлення з основними характеристиками

# шаблонів «Стратегія», «Шаблонний метод» та «Стан», 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

# За допомогою шаблона проектування у грі «Морський бій» реалізувати

# зміну зображення кораблів на ігровому полі в залежності від їх стану:

# цілий, поранений, убитий.

# За допомогою шаблона проектування реалізувати декілька способів

# переміщення нагору на вниз у торговому центрі. Наприклад, можна

# ходити сходами, їздити на ескалаторі та їздити на ліфті. В залежності

# від стану устаткування або від часу доби активувати той чи інший

# спосіб пересування відвідувачів. Наприклад, з 22.00 до 10.00 кожного

# дня ескалатори та ліфти перестають працювати. А під час

# профілактичних робіт з ескалатором має працювати ліфт та навпаки.

# Вибір структурного шаблону програмування

1. «В залежності від їх стану» - це шаблон «**Стан**».
2. Сімейство взаємозамінних алгоритмів, зміна алгоритмів незалежно від клієнта – це шаблон «**Стратегія**».

# Лістинг програми

|  |
| --- |
| Program.cs |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace OOP2\_Lab4  {  class Program  {  static void Main(string[] args)  {  #region  int[] coordsX = { 1, 1, 1 };  int[] coordsY = { 1, 2, 3 };  Ship ship = new Ship(coordsX, coordsY);  ship.display();  ship.hit(1, 1);  ship.display();  ship.hit(1, 3);  ship.display();  ship.hit(1, 2);  ship.display();  #endregion  Console.WriteLine("");  #region  Mall mall = new Mall();  mall.ascent();  mall.displayAccessibleWays();  mall.setWay(new Elevator());  mall.ascent();  mall.setWay(new Escalator());  mall.ascent();  mall.setWay(new Staircase());  mall.ascent();  #endregion  }  }  } |

|  |
| --- |
| Seabattle.cs |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace OOP2\_Lab4  {  class Ship  {  private ShipFragment[] fragments;  private int health;  private int healthStart;  public Ship(int [] coordsX, int [] coordsY)  {  int maxHealth = 4;  if (coordsX.Length <= maxHealth && coordsY.Length <= maxHealth && coordsX.Length == coordsY.Length &&  coordsX.Length >= 0 && coordsY.Length >= 0)  {  healthStart = health = coordsX.Length;  fragments = new ShipFragment[health];  for (int i = 0; i < health; i++)  {  fragments[i] = new ShipFragment(coordsX[i], coordsY[i]);  }  }  else  {  throw new Exception();  }  }  public void hit(int x, int y)  {  int killed = 0;  for (int i = 0; i < fragments.Length; i++)  {  if (x == fragments[i].PosX && y == fragments[i].PosY)  {  fragments[i].setWoundedState();  }  if (fragments[i].getName() == "Wounded")  {  killed++;  }  }  if (killed == healthStart)  {  Console.WriteLine("... and killed");  for (int i = 0; i < fragments.Length; i++)  {  fragments[i].setKilledState();  }  }  }  public void display()  {  for (int i = 0; i < fragments.Length; i++)  {  fragments[i].display();  }  }  }  class ShipFragment  {  private int posX;  private int posY;  public int PosX  {  get { return posX; }  }  public int PosY  {  get { return posY; }  }  private ShipFragmentHealthState currentState;  ShipFragmentStateUnscathed unscathedShipState = new ShipFragmentStateUnscathed();  ShipFragmentStateWounded woundedShipState = new ShipFragmentStateWounded();  ShipFragmentStateKilled killedShipState = new ShipFragmentStateKilled();  public ShipFragment(int x, int y)  {  currentState = unscathedShipState;  posX = x;  posY = y;  }  public void display()  {  currentState.display();  //Console.Write(" "+posX+","+posY);  }  public void setWoundedState()  {  currentState = woundedShipState;  Console.WriteLine("Ship was hit");  }  public void setKilledState()  {  currentState = killedShipState;  }  public string getName()  {  return currentState.getName();  }  }  abstract class ShipFragmentHealthState  {  protected string stateName;  abstract public void display();  public string getName()  {  return stateName;  }  }  class ShipFragmentStateUnscathed : ShipFragmentHealthState  {  public ShipFragmentStateUnscathed()  {  stateName = "Unscathed";  }  override public void display()  {  Console.WriteLine("+");  }  }  class ShipFragmentStateWounded : ShipFragmentHealthState  {  public ShipFragmentStateWounded()  {  stateName = "Wounded";  }  override public void display()  {  Console.WriteLine("~");  }  }  class ShipFragmentStateKilled : ShipFragmentHealthState  {  public ShipFragmentStateKilled()  {  stateName = "Killed";  }  override public void display()  {  Console.WriteLine("x");  }  }  } |

|  |
| --- |
| Mall.cs |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace OOP2\_Lab4  {  class Mall  {  private IAscentStrategy ascentStrategy;  public Mall() {  }  public void setWay(IAscentStrategy strategy)  {  ascentStrategy = strategy;  }  public void displayAccessibleWays()  {  int hours = Time.getHoursNow();    Console.WriteLine("1. Staircase is available!");  if (hours < 10 || hours > 22)  {  Console.WriteLine("2. Elevator and escalator are unavailable");  return;  }  else  {  if (hours > 14 && hours < 15)  {  Console.WriteLine("2. Elevator is unavailable");  }  else  {  Console.WriteLine("2. Elevator is available!");  }  if (hours > 13 && hours < 14)  {  Console.WriteLine("3. Escalator is unavailable");  }  else  {  Console.WriteLine("3. Escalator is available!");  }  }  }  public void ascent()  {  if (ascentStrategy == null)  {  Console.WriteLine("Choose strategy!");  return;  }  ascentStrategy.ascent();  }  }  public interface IAscentStrategy  {  void ascent();  }  public class Staircase : IAscentStrategy  {  public Staircase() { }  public void ascent()  {  Console.WriteLine("Going the staircase...");  }  }  public class Escalator : IAscentStrategy  {  public Escalator()  {  }  public void ascent()  {  int hours = Time.getHoursNow();  if (hours < 10 || hours > 22 || (hours > 13 && hours < 14))  {  Console.WriteLine("Escalator is closed");  return;  }  Console.WriteLine("Waiting on the escalator...");  }  }  public class Elevator : IAscentStrategy  {  public Elevator()  {  }  public void ascent()  {  int hours = Time.getHoursNow();  if (hours < 10 || hours > 22 || (hours > 14 && hours < 15))  {  Console.WriteLine("Elevator is closed");  return;  }  Console.WriteLine("Waiting in the elevator...");  }  }    static class Time  {  static public int getHoursNow()  {  string[] strArr = DateTime.Now.ToShortTimeString().Split(':');  return Convert.ToInt32(strArr[0]);  }  }  } |

