**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

**INFORMATIKOS FAKULTETAS**

**T120B516 Objektinis programų projektavimas**

**(Design Patterns)**

**Projektinio darbo ataskaita**

Atliko: Karolina Jašauskaitė, IFF-4/1,

Žygimantas Skinkys, IFF-4/1,

Aidas Balčaitis, IFF-4/1

Kaunas 2017

Turinys

[Įvadas 3](#_Toc497804663)

[Darbo tikslas 3](#_Toc497804664)

[1. Pirmas labaratorinis darbas 3](#_Toc497804665)

[Programavimo šablonai: 3](#_Toc497804666)

[1.1. Singleton 3](#_Toc497804667)

[1.2. Factory 3](#_Toc497804668)

[1.3. Strategy 5](#_Toc497804669)

[1.4. Observer 6](#_Toc497804670)

[2. Antras labaratorinis darbas 6](#_Toc497804671)

[Programavimo šablonai 6](#_Toc497804672)

[2.1. Adapter 6](#_Toc497804673)

[2.2. Decorator 6](#_Toc497804674)

[2.3. Facade 7](#_Toc497804675)

[2.4. Command 7](#_Toc497804676)

[Išvados 7](#_Toc497804677)

# Įvadas

## Darbo tikslas

Kuriant „Pacman” žaidimą išmokti taikyti projektavimo šablonus (angl. „design patterns“) ir susipažinti su jų naudojimo ypatumais.

„Pacman” žaidime vartotojas valdys pagrindinį herojų, kurio tikslas surinkti kuo daugiau taškį renkant “Candy”, ir išvengi priešų “Enemy”.

Žaidime naudojami 5 pagrindiniai objektai: žaidimas, žaidimo lenta, priešai, saldainiai ir žaidėjo valdomas objektas.

# Pirmas labaratorinis darbas

## Programavimo šablonai:

## Singleton

Panaudojimo tikslas:

Panaudojome Singleton programavimo šabloną, kad užtikrintume, kad vienu metu žaidėjas žaidžia tik vieną žaidimą.

Uml diagrama:

Pav. 1 Singleton programavimo šablonas

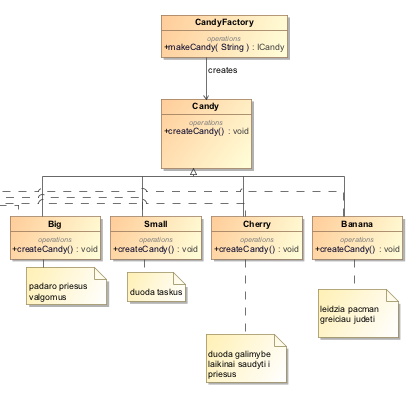
Kodas:

## Factory

Panaudojimo tikslas:

Kadangi turime 4 skirtingų tipų saldainius ( Banana, Big, Small, Cherry), norint juos realizuoti ir panaudoti savo žaidime, pasinaudojome “Factory” programavimo šabloną, kuris pagal gautą atributą sukurs saldainius.

Uml diagrama:



Pav. 2 Factory programavimo šablonas

Kodas:

|  |
| --- |
| CandyFactory.cs |
| using System; namespace Pacman {     public class CandyFactory     {         public CandyFactory()         {             public Candy createCandy(String newCandyType)             {                 Candy newCandy = null;                 if (newCandyType.Equals("Banana"))                 {                     return new Banana();                 }                 else if (newCandyType.Equals("Big"))                 {                     return new Big();                 }                 else if (newCandyType.Equals("Cherry"))                 {                     return new Cherry();                 }                 else if (newCandyType.Equals("Small"))                 {                     return new Small();                 }             }         }     } } |
| Candy.cs |
| using System; namespace Pacman {     public abstract class Candy     {          public abstract Candy CreateCandy();       } } |
| Banana.cs |
| using System; namespace Pacman {     public class Banana : Candy     {         public Banana()         {         }     } } |
| Cherry.cs |
| using System; namespace Pacman {     public class Cherry : Candy     {         public Cherry()         {         }     } } |
| Big.cs |
| using System; namespace Pacman {     public class Big : Candy     {         public Big()         {         }     } } |
| Small.cs |
| using System; namespace Pacman {     public class Small  : Candy      {         public Small()         {         }     } } |

## Strategy

Panaudojimo tikslas:

Uml diagrama:

Pav. 3 Strategy programavimo šablonas

Kodas:

## Observer

Panaudojimo tikslas:

Uml diagrama:

Pav. 4 Observer programavimo šablonas

Kodas:

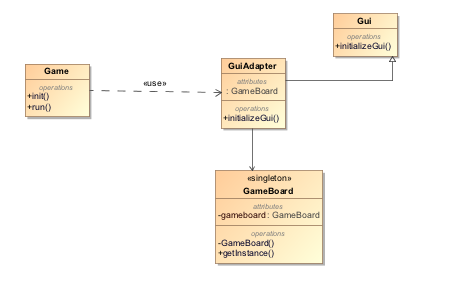
# Antras labaratorinis darbas

## Programavimo šablonai

## Adapter

Panaudojimo tikslas:

Uml diagrama:



Pav. 5 Adapter programavimo šablonas

Kodas:

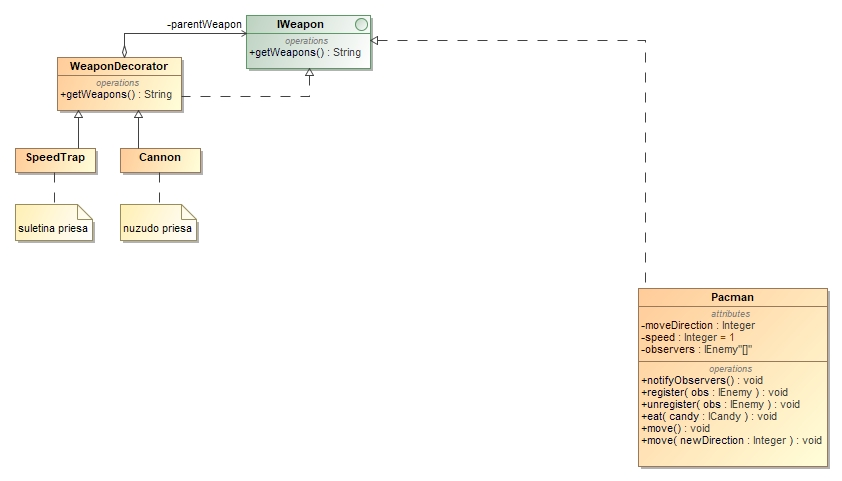
|  |
| --- |
| Gui.cs |
| using System; namespace Pacman {     public class Gui     {         public virtual void InitializeGui()         {             Console.WriteLine("Initializing graphical interface");         }     } } |
| GuiAdapter.cs |
| using System; namespace Pacman {     public class GuiAdapter : Gui     {         private readonly BaseWindowFascade \_windowFascade = new BaseWindowFascade();         public override void InitializeGui()         {             Console.WriteLine("Adapter works");             Console.ReadKey();             \_windowFascade.Run();         }     } } |
| Game.cs |
| using System; namespace Pacman {     public class Game     {         public static void Main(string[] args)         {             var diSetup = new DependencyInjectionSetup().GetScope();             Gui gui = new GuiAdapter();             gui.InitializeGui();         }     }  } |

## Decorator

Panaudojimo tikslas:

Pasirinkome naudoti Decorator programavimo šabloną, nes Pacman gali turėti vieną iš dviejų ginklų. Vietoje to kad naudoti switch case ar if sakinius priklausomai nuo pasirinkto ginklo, panaudojome Decorator programavimo šabloną, kad ginklai būtų pridedami be didesnių sunkumų.

Uml diagrama:



Pav. 6 Decorator programavimo šablonas

Kodas:

|  |
| --- |
| IWeapon.cs |
| using System;  using System.Collections.Generic;  using System.Text;  namespace Pacman  {  interface IWeapon  {  String getWeapons();  }  } |
| WeaponDecorator.cs |
| using System;  using System.Collections.Generic;  using System.Text;  namespace Pacman  {  class WeaponDecorator : IWeapon  {  private IWeapon parentWeapon;  public string getWeapons()  {  return parentWeapon.ToString();  }  }  } |
| SpeedTrap.cs |
| using System;  using System.Collections.Generic;  using System.Text;  namespace Pacman  {  class SpeedTrap : WeaponDecorator  {    }  } |
| Cannon.cs |
| using System;  using System.Collections.Generic;  using System.Text;  namespace Pacman  {  class Cannon : WeaponDecorator  {  }  } |
| Pacman.cs |
| using System;  using System.Collections.Generic;  using System.Text;  namespace Pacman  {  class Pacman  {  private int moveDirection;  private int speed = 1;  private List<IEnemy> observers;  void notifyObservers()  {  foreach(IEnemy obs in observers){  obs.update();  }  }  void register(IEnemy obs)  {  observers.Add(obs);  }  void unregister(IEnemy obs)  {  observers.Remove(obs);  }  void eat(ICandy candy)  {  }  public void move()  {  throw new NotImplementedException();  }  public void move(int newDirection)  {  this.moveDirection = newDirection;  }    }  } |

## Facade

Panaudojimo tikslas:

Uml diagrama:

Pav. 7 Facade programavimo šablonas

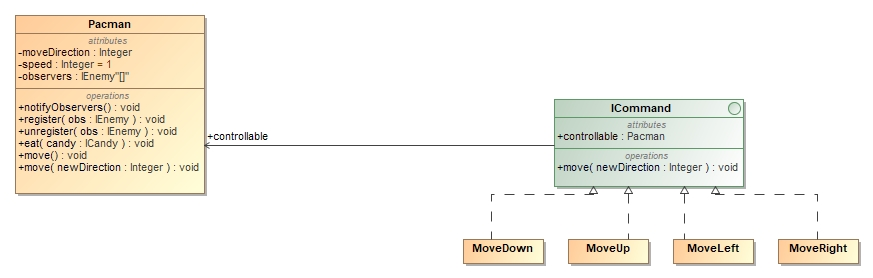
Kodas:

## Command

Panaudojimo tikslas:

Command šabloną panaudojom tam, kad paslėpti komandų apdorojimų logiką. Tai mums leis vėliau lengvai pridėti naujų kamandų ir jas apdoroti objektiškai.

Uml diagrama:



Pav. 8 Command programavimo šablonas

Kodas:

|  |
| --- |
| ICommand.cs |
| using System;  using System.Collections.Generic;  using System.Text;  namespace Pacman  {  interface ICommand  {  Pacman controllable { get; set; }  void move(int newDirection);  }  } |
| MoveDown.cs |
| using System;  using System.Collections.Generic;  using System.Text;  namespace Pacman  {  class MoveDown : ICommand  {  public Pacman controllable { get => throw new NotImplementedException(); set => throw new NotImplementedException(); }  public void move(int newDirection)  {  this.controllable.move(3);  }  }  } |
| MoveUp.cs |
|  |
| MoveLeft.cs |
| using System;  using System.Collections.Generic;  using System.Text;  namespace Pacman  {  class MoveLeft : ICommand  {  public Pacman controllable { get => throw new NotImplementedException(); set => throw new NotImplementedException(); }  public void move(int newDirection)  {  this.controllable.move(4);  }  }  } |
| MoveRight.cs |
|  |

# Išvados

Programos įgyvendinimui panaudoti programavimo šablonai: Observer, Strategy, Factory, Singleton, Facade, Command, Adapter ir Decorator. Kai kuriose vietose šie šablonai padėjo sumažinti programos sudėtingumą, padaryti kodą labiau skaitomą ir lengviau modifikuojamą.

Tačiau, supratome, kad perteklinis šablonų panaudojimas, apsunkina kodo rašymą ir pailgina įgyvendinimą.

1. Observers – šio šablono dėka yra lengviau pridėti bei pašalinti priešus, taip pat keivkieną kartą atnaujinus ekrano vaizdą paskaičiuoti priešų gyvybes.
2. Strategy – leido implementuoti skirtingus žaidimo lygius, kuriuos galime keisti programos veikimo metu.
3. Factory – padėjo lokalizuoti strategijos objekto kūrimą ir paslėpti jo kūrimo sudėtingumą nuo kliento.
4. Singleton – šis šablonas padeda užtikrinti, kad vieno žaidimo metu bus prieinamas tik vienas žaidimo objektas.
5. Facade – padeda paslėpti nereikalingą sudėtingumą ir funkcionalumą nuo kliento, atskleisdamas tik jam reikalingus metodus.
6. Command – leido pakeisti „string“ tipo komandas į „strongly typed“ objektus.
7. Adapter – šis šablonas padeda lengvai praplėsti objekto funkcionalumą. Kadangi nežinome kokį patobulinimą po kokio pasirinks žaidėjas, tai kiekvieno patobulinimo metu, mes tiesiog apgaubiama jau esantį objektą, kad neprarasti senų patobulinimu, bet pridėti ir naujus.
8. Decorator – leido nekeičiant jau esamos „spaceship“ klasės pridėti prie jos papildomą funkcionalumą