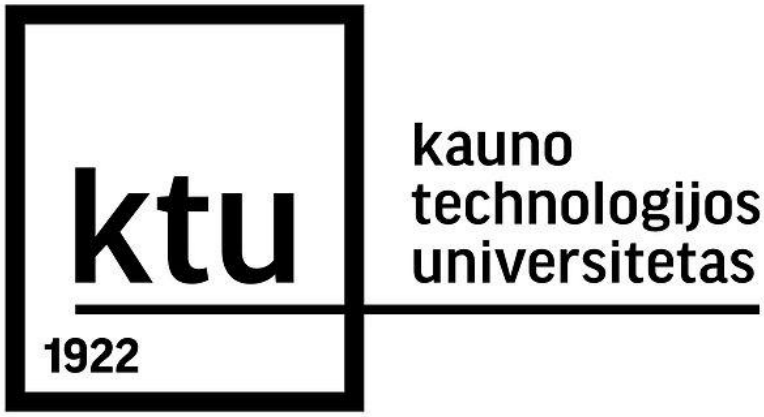


INFORMATIKOS FAKULTETAS

T120B516 Objektinis programų projektavimas

Projekto ataskaita 2

Studentai: Ignas Savickas IFF-4/2
Jonas Ausevičius IFF-4/2

Dėstytojai: Lekt. dr. Andrej Ušianov
Lekt. Kęstutis Valinčius

KAUNAS 2017

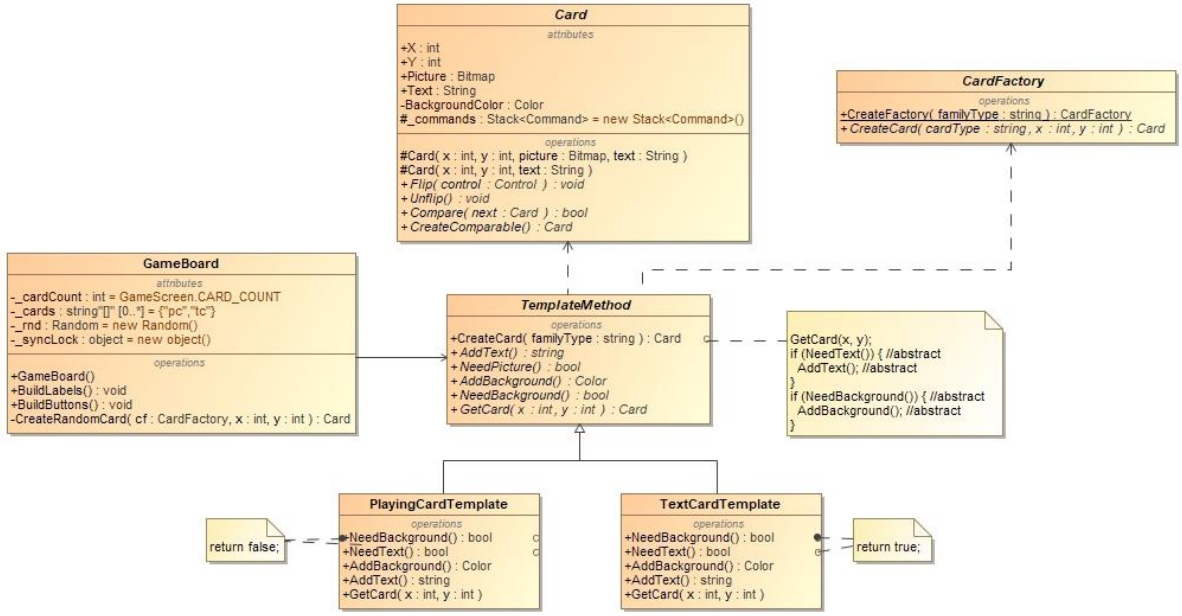
Turinys

1. Projektavimo šablonai (3 laboratorinis)

Kiekvienam laboratoriniam darbui buvo pasirinkti atskiri projektavimo šablonai. Pirmajam – Template Method, Flyweight, State.

1.1. Template Method

1.1.1. Klasių diagrama



pav. 1. Template Method projektavimo šablono pritaikymo klasė

1.1.2. Esminis kodas

```
public abstract class TemplateMethod
{
    public virtual Card CreateCard(int x, int y)
    {
        Card card = GetCard(x, y);

        if (NeedText())
        {
            card.Text = card.Text + AddText();
        }
        if (NeedBackground())
        {
            card.BackgroundColor = AddBackground();
        }
        return card;
    }
```

```
    protected abstract Card GetCard(int x, int y);

    protected abstract bool NeedText();

    protected abstract string AddText();

    protected abstract bool NeedBackground();

    protected abstract Color AddBackground();

}


public class PlayingCardTemplate : TemplateMethod
{
    private readonly string[] _playingCards = { "h", "s", "d", "c" };
    private readonly System.Random _rnd = new System.Random();
    private readonly object _syncLock = new object();

    protected override Card GetCard(int x, int y)
    {
        var cf = CardFactory.CreateFactory("pc");
        int index;
        lock (_syncLock)
        {
            index = _rnd.Next(0, _playingCards.Length);
        }
        string cardType = _playingCards[index];
        return cf.CreateCard(cardType, x, y);
    }

    protected override bool NeedText()
    {
        return false;
    }

    protected override string AddText()
    {
        return null;
    }

    protected override bool NeedBackground()
    {
        return false;
    }

    protected override Color AddBackground()
    {
        return Color.White;
    }
}


public class TextCardTemplate : TemplateMethod
{
    private readonly string[] _textCards = { "l", "n" };
    private readonly System.Random _rnd = new System.Random();
    private readonly object _syncLock = new object();

    protected override Card GetCard(int x, int y)
    {
```

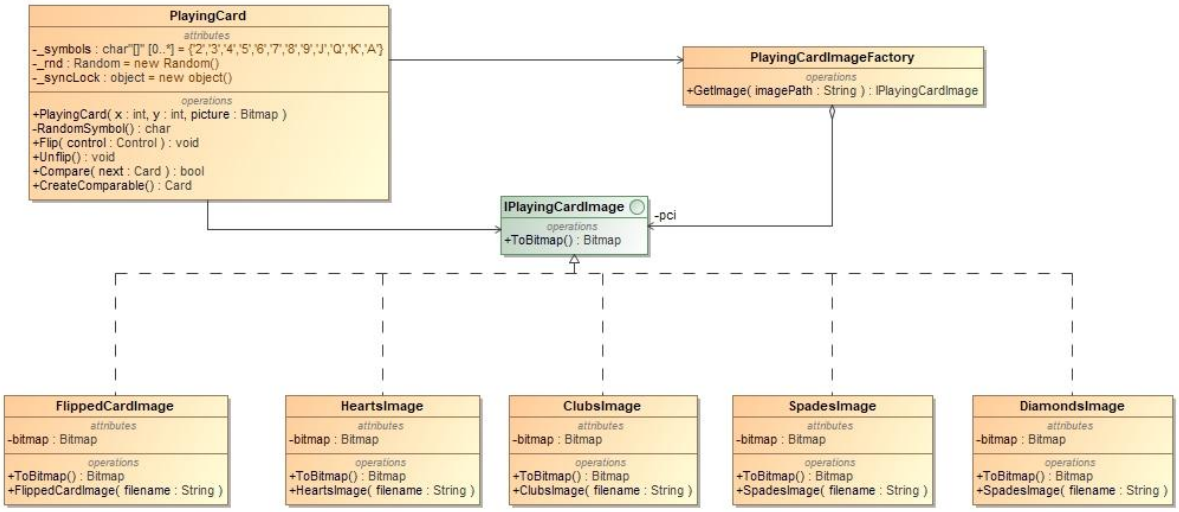
```
            var cf = CardFactory.CreateFactory("tc");
            int index;
            lock (_syncLock)
            {
                index = _rnd.Next(0, _textCards.Length);
            }
            string cardType = _textCards[index];
            return cf.CreateCard(cardType, x, y);
        }

        protected override bool NeedText()
        {
            return true;
        }

        protected override string AddText()
        {
            return Random.GetLetter() + Random.GetNumber();
        }

        protected override bool NeedBackground()
        {
            return true;
        }

        protected override Color AddBackground()
        {
            return Random.GetColor();
        }
}
```

1.1.3. Naudojimo pagrindimas

Sukuriamas bendras kortos kūrimo algoritmas, kurio žingsnių eiliškumas yra nekeičiamas. Kiekvienas žingsnis yra keičiamas klasėje, kuri paveldi abstrakčią klasę. Taip klientas negalės keisti kortos kūrimo algoritmo. Tam tikrą korta bus sukuriama pasinaudojant tam tikrą algoritmą.

1.2. Flyweight

1.2.1. Klasių diagrama



pav. 2. Flyweight projektavimo šablono pritaikymo klasės

1.2.2. Esminis kodas

```
public class PlayingCardImageFactory
{
    private static readonly Hashtable Hash = new Hashtable();

    public static IPlayingCardImage GetImage(string image)
    {
        IPlayingCardImage pci = (IPlayingCardImage) Hash[image];

        if (pci != null) return pci;
        if (image.Equals("clubs"))
        {
            pci = new ClubsImage();
        }
        if (image.Equals("diamonds"))
        {
            pci = new DiamondsImage();
        }
        if (image.Equals("hearts"))
        {
            pci = new HeartsImage();
        }
        if (image.Equals("spades"))
        {
            pci = new SpadesImage();
        }
        if (image.Equals("flipped_card"))
        {
            pci = new FlippedCardImage();
        }

        Hash.Add(image, pci);

        return pci;
```

```
    }
}


public interface IPlayingCardImage
{
    Bitmap ToBitmap();
}


public class ClubsImage : IPlayingCardImage
{
    private readonly Bitmap _bitmap;

    public ClubsImage()
    {
        _bitmap = new Bitmap("../../Resources/clubs.png");
    }

    public Bitmap ToBitmap()
    {
        return _bitmap;
    }
}
```

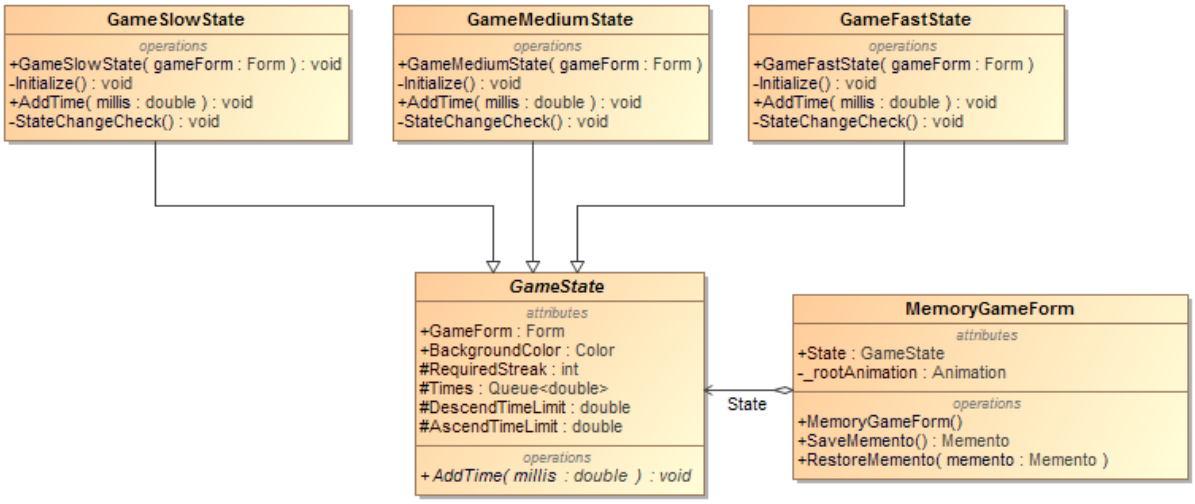
Visos kitos klasės, kurios implementuoja IPlayingCardImage, yra labai panašios į ClubsImage klasė. Skiriasi tik paveiksliuko vieta.

1.2.3. Naudojimo pagrindimas

Gaunant naują paveiksliuką programa atsimina jo vietą ir kai kitą kartą reikalingas toks pat paveiksliukas, tai programa nekuria naujo, o pasiima seną. Taip tausojama kompiuterio atmintis ir greičiau užkraunami paveiksliukai žaidimo paleidimo metu.

1.3. State

1.3.1. Klasių diagrama



pav. 3. State šablono pritaikymo klasių lentelės

1.3.2. Esminis kodas

```
public partial class MemoryGameForm : System.Windows.Forms.Form
{
    public GameState State;
    private readonly Stopwatch _objectivesWatch;
    private Animation _rootAnimation;

    public MemoryGameForm()
    {
        InitializeComponent();

        _rootAnimation = new LowAnimation();
        var medAnimation = new MediumAnimation();
        var highAnimation = new HighAnimation();

        _rootAnimation.SetSuccessor(medAnimation);
        medAnimation.SetSuccessor(highAnimation);

        State = new GameSlowState(this);
        _objectivesWatch = new Stopwatch();
        _objectivesWatch.Start();
        ConstructGameScreen();
    }

    protected override void OnPaintBackground(PaintEventArgs e)
    {
        base.OnPaintBackground(e);
        BackColor = State.BackgroundColor;
    }

    public void CompleteObjective(Card card)
    {
        string text = card.Text;
        _rootAnimation.ProcessRequest(new AnimationRequest(text));
```

```
            State.AddTime(_objectivesWatch.Elapsed.TotalMilliseconds);
            _objectivesWatch.Restart();
        }
    }

    public abstract class GameState
    {
        public MemoryGameForm GameForm;

        public Color BackgroundColor { get; protected set; }

        protected int RequiredStreak;
        protected Queue<double> Times;
        // if Sum(times) >= this, then decrease game speed state
        protected double DescendTimeLimit;
        // if Sum(times) when Count(times) == requiredStreak <= this, then increase game
    speed state
        protected double AscendTimeLimit;

        public abstract void AddTime(double millis);
    }

    class GameSlowState : GameState
    {
        public GameSlowState(GameState state) : this(state.GameForm)
        {
        }

        public GameSlowState(MemoryGameForm gameForm)
        {
            GameForm = gameForm;
            Times = new Queue<double>();
            Initialize();
        }

        private void Initialize()
        {
            BackgroundColor = Color.BlanchedAlmond;
            RequiredStreak = 5;
            DescendTimeLimit = Int32.MaxValue;
            AscendTimeLimit = GameScreen.CARD_COUNT * GameScreen.CARD_COUNT * 1000 * 1.5;
        }

        public override void AddTime(double millis)
        {
            while (Times.Count >= RequiredStreak)
                Times.Dequeue();
            Times.Enqueue(millis);

            StateChangeCheck();
        }

        private void StateChangeCheck()
        {
            if (Times.Count >= RequiredStreak && Times.Sum() <= AscendTimeLimit)
            {
                GameForm.State = new GameMediumState(this);
            }
        }
    }

    class GameMediumState : GameState
```

```
{
    public GameMediumState(GameState state) : this(state.GameForm)
    {
    }

    public GameMediumState(MemoryGameForm gameForm)
    {
        GameForm = gameForm;
        Times = new Queue<double>();
        Initialize();
    }

    private void Initialize()
    {
        BackgroundColor = Color.BurlyWood;
        RequiredStreak = 5;
        DescendTimeLimit = GameScreen.CARD_COUNT * GameScreen.CARD_COUNT * 1000 * 1.4;
        AscendTimeLimit = GameScreen.CARD_COUNT * GameScreen.CARD_COUNT * 1000 * 1.1;
    }

    public override void AddTime(double millis)
    {
        while (Times.Count >= RequiredStreak)
            Times.Dequeue();
        Times.Enqueue(millis);

        StateChangeCheck();
    }

    private void StateChangeCheck()
    {
        if (Times.Count >= RequiredStreak && Times.Sum() <= AscendTimeLimit)
        {
            GameForm.State = new GameFastState(this);
        }
        else if (Times.Sum() >= DescendTimeLimit)
        {
            GameForm.State = new GameSlowState(this);
        }
    }
}
```

1.3.3. Naudojimo pagrindimas

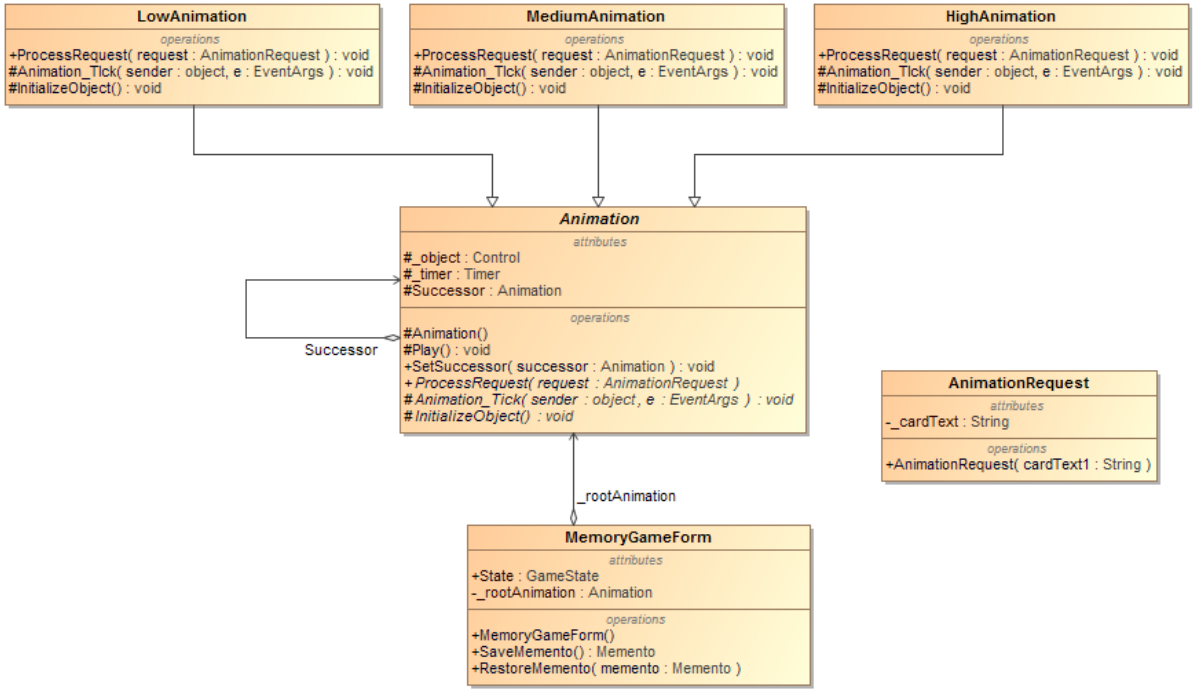
State šablonas leido keisti formos būseną (šiuo atveju jos spalvą) automatiškai, pagal tam tikras sąlygas, kai atliekamas veiksmas. Formoje nereikėjo nurodyti galimų konkrečių „state“ klasių, ar atlikti skaičiavimus patikrinti į kurią ir kada reikia keisti – nėra artimo susiejimo.

2. Projektavimo šablonai (4 laboratorinis)

Antrajame laboratoriniame darbe buvo naudojami 4 projektavimo šablonai: Chain of responsibility, Interpreter, Mementor ir Null Object.

2.1. Chain of responsibility

2.1.1. Klasių diagrama



pav. 4. Chain of responsibility projektavimo šablono klasių diagrama

2.1.2. Esminis kodas

```
public partial class MemoryGameForm : System.Windows.Forms.Form
{
    public GameState State;
    private readonly Stopwatch _objectivesWatch;
    private Animation _rootAnimation;

    public MemoryGameForm()
    {
        InitializeComponent();

        _rootAnimation = new LowAnimation();
        var medAnimation = new MediumAnimation();
        var highAnimation = new HighAnimation();

        _rootAnimation.SetSuccessor(medAnimation);
        medAnimation.SetSuccessor(highAnimation);

        State = new GameSlowState(this);
        _objectivesWatch = new Stopwatch();
        _objectivesWatch.Start();
        ConstructGameScreen();
    }

    public void CompleteObjective(Card card)
    {
        string text = card.Text;
        _rootAnimation.ProcessRequest(new AnimationRequest(text));
        State.AddTime(_objectivesWatch.Elapsed.TotalMilliseconds);
        _objectivesWatch.Restart();
    }
}
```

```
abstract class Animation
{
    protected Control _object;
    protected readonly Timer _timer;
    protected Animation Successor;

    protected Animation()
    {
        InitializeObject();

        _timer = new Timer();
        _timer.Tick += Animation_Tick;
        _timer.Interval = 10;
    }

    protected void Play()
    {
        _object?.Dispose();
        InitializeObject();

        _timer.Stop();
        _timer.Start();

        System.Windows.Forms.Form.ActiveForm?.Controls.Add(_object);
    }

    public void SetSuccessor(Animation successor)
    {
        Successor = successor;
    }

    public abstract void ProcessRequest(AnimationRequest request);

    protected abstract void Animation_Tick(object sender, EventArgs e);

    protected abstract void InitializeObject();
}

class AnimationRequest
{
    private string _cardText;

    public AnimationRequest(string cardText1)
    {
        this._cardText = cardText1;
    }

    public string CardText
    {
        get { return _cardText; }
        set { _cardText = value; }
    }
}

class LowAnimation : Animation
{
    public override void ProcessRequest(AnimationRequest request)
    {
        if (Int32.TryParse(request.CardText, out var num) && num < 6)
        {
            Play();
```

```
        }
        else
        {
            Successor?.ProcessRequest(request);
        }
    }

    protected override void Animation_Tick(object sender, EventArgs e)
    {
        _object.Size += new Size(50, 50);
        _object.Location = new Point(_object.Location.X - 25, _object.Location.Y-25);
        if (_object.Size.Width > GameScreen.Instance.SCREEN_WIDTH * 1.5)
        {
            _timer.Stop();
            _object.Dispose();
        }
    }

    protected override void InitializeObject()
    {
        _object = new Control
        {
            Location = new Point(GameScreen.Instance.SCREEN_WIDTH / 2,
GameScreen.Instance.SCREEN_HEIGHT / 2),
            Size = new Size(1, 1),
            BackColor = Color.Aquamarine
        };
    }
}

class MediumAnimation : Animation
{
    public override void ProcessRequest(AnimationRequest request)
    {
        if (Int32.TryParse(request.CardText, out var _))
        {
            Play();
        }
        else
        {
            Successor?.ProcessRequest(request);
        }
    }

    protected override void Animation_Tick(object sender, EventArgs e)
    {
        _object.Size += new Size(30, 30);
        _object.Location = new Point(_object.Location.X - 15, _object.Location.Y-15);
        if (_object.Size.Width > GameScreen.Instance.SCREEN_WIDTH * 1.5)
        {
            _timer.Stop();
            _object.Dispose();
        }
    }

    protected override void InitializeObject()
    {
        _object = new Control
        {
            Location = new Point(GameScreen.Instance.SCREEN_WIDTH / 2,
GameScreen.Instance.SCREEN_HEIGHT / 2),
            Size = new Size(1, 1),
```

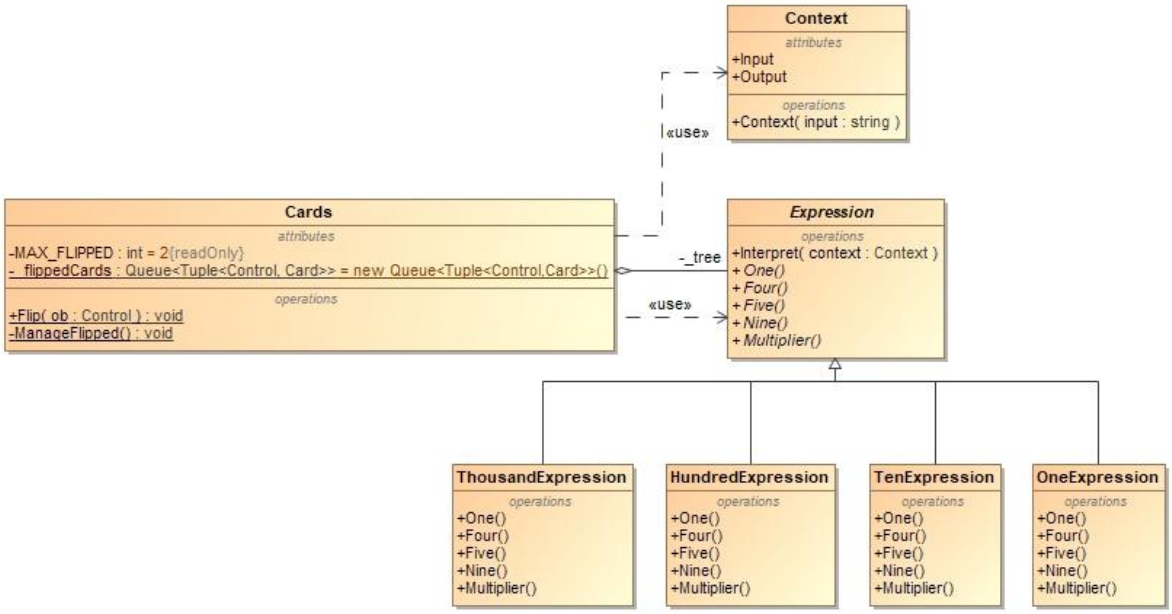
```
            BackColor = Color.DarkCyan
        };
    }
}
```

2.1.3. Naudojimo pagrindimas

Chain of responsibility šablonas leido atskirti formos klasę nuo konkrečių funkcijas atliekančių klasių. Mūsų projekte šablonas naudojamas iš formos į eilę klasių (turinčias skirtingus parametrus ir veiksmus) pasiųsti surastų kortų tekstą – tekstas keliauja tarp klasių kol atitinka vieną iš jų reikalavimų (tam tikras skaičius, raidė ir t.t.). Formoje nereikėjo aprašyti reikalavimų ir veiksmų, panaudotas šablonas gali būti nesunkiai praplečiamas pridedant naują klasią į eilę.

2.2. Interpreter

2.2.1. Klasių diagrama



pav. 5. Interpreter projektavimo šablono pritaikymo klasės

2.2.2. Esminis kodas

```
public class Context
{
    public string Input { get; set; }
    public int Output { get; set; }

    public Context(string input)
    {
        Input = input;
    }
}
```

```
public abstract class Expression
{
    public void Interpret(Context context)
    {
        if (context.Input.Length == 0)
        {
            return;
        }
        if (context.Input.StartsWith(Nine()))
        {
            context.Output += 9 * Multiplier();
            context.Input = context.Input.Substring(2);
        }
        else if (context.Input.StartsWith(Four()))
        {
            context.Output += 4 * Multiplier();
            context.Input = context.Input.Substring(2);
        }
        else if (context.Input.StartsWith(Five()))
        {
            context.Output += 5 * Multiplier();
            context.Input = context.Input.Substring(1);
        }
        while (context.Input.StartsWith(One()))
        {
            context.Output += Multiplier();
            context.Input = context.Input.Substring(1);
        }
    }

    public abstract string One();
    public abstract string Four();
    public abstract string Five();
    public abstract string Nine();
    public abstract int Multiplier();
}


public class HundredExpression : Expression
{
    public override string One()
    {
        return "C";
    }

    public override string Four()
    {
        return "CD";
    }

    public override string Five()
    {
        return "D";
    }

    public override string Nine()
    {
        return "CM";
    }

    public override int Multiplier()
```

```
    {
        return 100;
    }
}


public class TenExpression : Expression
{
    public override string One()
    {
        return "X";
    }

    public override string Four()
    {
        return "XL";
    }

    public override string Five()
    {
        return "L";
    }

    public override string Nine()
    {
        return "XC";
    }

    public override int Multiplier()
    {
        return 10;
    }
}
```

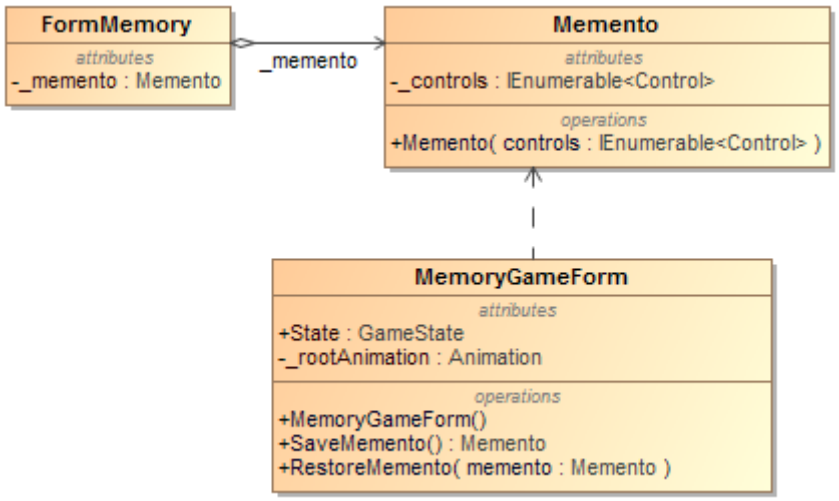
Kitų dvi klasių principas panašus į dvi paskutines klases. Skiriasi tik tas, kad tūkstančių išraiškoje grąžinama tik vientas „M“ ir daugiklis 1000, o vieneto išraiškoje – vienetai romėniškoje numeracijoje.

2.2.3. Naudojimo pagrindinis

Gaunant romėnišką skaičių programa paverčia skaičių į arabišką panaudodama šį šabloną. Nereikia naudoti išorinių bibliotekų. Atversti skaičiai sudedami ir atspausdinami komandinėje eilutėje.

2.3. Memento

2.3.1. Klasių diagrama



pav. 6. Memento projektavimo šablono klasių diagrama

2.3.2. Esminis kodas

```
public partial class MemoryGameForm : System.Windows.Forms.Form
{

    public GameState State;
    private readonly Stopwatch _objectivesWatch;
    private Animation _rootAnimation;

    public MemoryGameForm()
    {
        InitializeComponent();

        _rootAnimation = new LowAnimation();
        var medAnimation = new MediumAnimation();
        var highAnimation = new HighAnimation();

        _rootAnimation.SetSuccessor(medAnimation);
        medAnimation.SetSuccessor(highAnimation);

        State = new GameSlowState(this);
        _objectivesWatch = new Stopwatch();
        _objectivesWatch.Start();
        ConstructGameScreen();
    }

    public Memento SaveMemento()
    {
        var copy = new Control[Controls.Count];
        Controls.CopyTo(copy, 0);
        return new Memento(copy);
    }
```

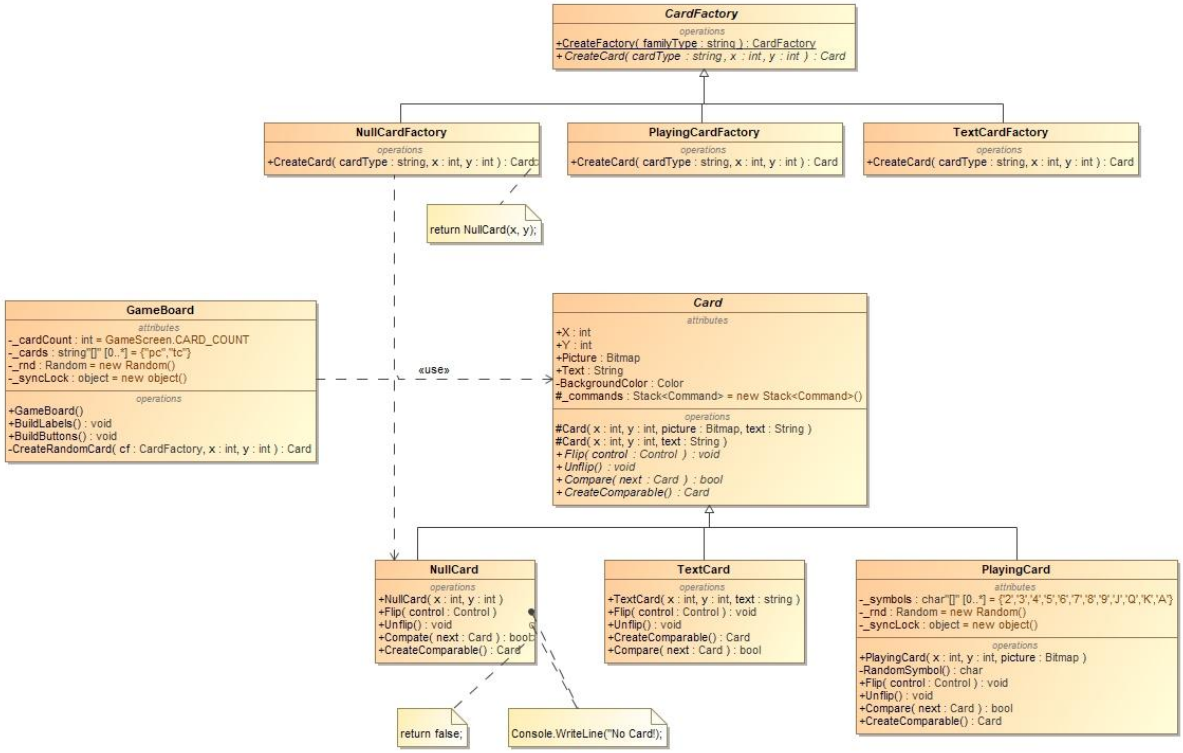
```
    public void RestoreMemento(Memento memento)
    {
        this.Controls.Clear();
        this.Controls.AddRange(memento.Controls.ToArray());
    }
}

public class Memento
{
    private IEnumerable<Control> _controls;

    public Memento(IEnumerable<Control> controls)
    {
        _controls = controls;
    }

    public IEnumerable<Control> Controls
    {
        get { return _controls; }
        set { _controls = value; }
    }
}

public static class FormMemory
{
    private static Memento _memento;

    public static Memento Memento
    {
        get { return _memento; }
        set { _memento = value; }
    }
}

public static class GameControls
{
    public static void InfoButton_Click(object sender, EventArgs e)
    {
        var activeForm = System.Windows.Forms.Form.ActiveForm as MemoryGameForm;
        FormMemory.Memento = activeForm.SaveMemento();
        activeForm.Controls.Clear();
        activeForm.ConstructInfoScreen();
    }

    public static void BackButton_Click(object sender, EventArgs e)
    {
        var activeForm = System.Windows.Forms.Form.ActiveForm as MemoryGameForm;
        activeForm.Controls.Clear();
        activeForm.RestoreMemento(FormMemory.Memento);
    }
}
```

2.3.3. Naudojimo pagrindimas

Memento šablonas naudojamas išsaugoti formos elementus prieš pakeičiant formos langą. Išsaugomi visi elementai ir jų stadijos – jei buvo atversta korta, ji atmintyje ir liks atversta. Šablonas suteikė galimybę išsaugoti objekto vidinę būseną, sukurta būsenų klasė gali būti panaudota saugojimui keletų būsenų.

2.4. Null Object

2.4.1. Klasių diagrama



pav. 7. Null Object projektavimo šablono pritaikymo klasės

2.4.2. Esminis kodas

```
public abstract class CardFactory
            {
    public static CardFactory CreateFactory(string familyType)
    {
        if (familyType.Equals("pc"))
        {
            return new PlayingCardFactory();
        }
        if (familyType.Equals("tc"))
        {
            return new TextCardFactory();
        }
        return new NullCardFactory();
    }

    public abstract Card CreateCard( string cardType, int x, int y);
}


public class NullCardFactory : CardFactory
{
```

```
    public override Card CreateCard(string cardType, int x, int y)
    {
        return new NullCard(x, y);
    }
}


public abstract class Card
{
    public int X { get; set; }

    public int Y { get; set; }

    public Bitmap Picture { get; set; }

    public string Text { get; set; }

    public Color BackgroundColor { get; set; }

    protected readonly Stack<Command> _commands = new Stack<Command>();

    protected Card(int x, int y, Bitmap picture, string text)
    {
        X = x;
        Y = y;
        Picture = picture;
        Text = text;
    }

    protected Card(int x, int y, string text)
    {
        X = x;
        Y = y;
        Text = text;
    }

    public abstract void Flip(Control control);

    public abstract void Unflip();

    public abstract bool Compare(Card next);

    public abstract Card CreateComparable();
}


public class NullCard : Card
{
    public NullCard(int x, int y) : base(x, y, Properties.Resources._null, "null")
    {
    }

    public override void Flip(Control control)
    {
        Console.WriteLine(@"No card!");
    }

    public override void Unflip()
    {
        Console.WriteLine(@"No card!");
    }
```

```
    public override bool Compare(Card next)
    {
        return false;
    }

    public override Card CreateComparable()
    {
        return (Card)this.MemberwiseClone();
    }
}
```

2.4.3. Naudojimo pagrindimas

Jeigu atsitiktų taip, kad atitinkamas tekstas nebūtų sugeneruotas, tai būtų sukuriama tuščia gamykla, kurioje būtų sukuriama tuščia korta arba tiesiog tuščia korta būtų sukurta tam tikrose gamyklose. Taip atsitikus tokios kortos būtų pažymėtos raudonu „X“ ir nupieštos žaidime. Paspaudus ant tokios kortos, veiksmas būtų atšauktas ir komandinėje eilutėje būtų išvesta „No Card!”. Panaudojus šį šabloną, žaidimas nenustotų veikti, tik išvestų klaidą.

2.5. Dependency Injection

2.5.1. Esminis kodas

```
abstract class Animation
{
    protected Control _object;
    protected readonly Timer _timer;
    protected Animation Successor;

    protected Animation()
    {
        InitializeObject();

        _timer = new Timer();
        _timer.Tick += Animation_Tick;
        _timer.Interval = 10;
    }

    protected void Play()
    {
        _object?.Dispose();
        InitializeObject();

        _timer.Stop();
        _timer.Start();

        System.Windows.Forms.Form.ActiveForm?.Controls.Add(_object);
    }

    public void SetSuccessor(Animation successor)
    {
        Successor = successor;
    }
```

```
    public abstract void ProcessRequest(AnimationRequest request);

    protected abstract void Animation_Tick(object sender, EventArgs e);

    protected abstract void InitializeObject();
}


public partial class MemoryGameForm : System.Windows.Forms.Form
{
    public GameState State;
    private readonly Stopwatch _objectivesWatch;
    private Animation _rootAnimation;

    public MemoryGameForm()
    {
        InitializeComponent();

        _rootAnimation = new LowAnimation();
        var medAnimation = new MediumAnimation();
        var highAnimation = new HighAnimation();

        _rootAnimation.SetSuccessor(medAnimation);
        medAnimation.SetSuccessor(highAnimation);

        State = new GameSlowState(this);
        _objectivesWatch = new Stopwatch();
        _objectivesWatch.Start();
        ConstructGameScreen();
    }

}
```

3. Išvados

- Template Method

  Kuriant kortas buvo iškilusi problema, kad kiekvienas kortos tipas reikalauja skirtingų savybių. Tam, kad išspręstume šią problemą, panaudojome „Template Method“ projektavimo šabloną. Šablone yra iškviečiama tam tikra kortos gamykla, kuri grąžina kortą, ir, kad pridėtume skirtingas savybes, reikalingas algoritmas, kuris tinka visoms kortoms. Algoritmas yra visoms kortoms vienodas tik skiriasi žingsnių atlikimas. Gautas rezultatas yra, kad kortos generavimas yra paprastesnis yra užtenka iškviesti kortos šablono metodą.

- Flyweight

  Šis šablonas pasirinktas tam, kad paveikslėlių krovimas žaidime būtų greitesnis nei buvo anksčiau. Kai reikia užkrauti žaidimų kortos foną, iš „Flyweight“ projektavimo šablono gauname paveikslėlį. Šablono viduje tikrinama, ar paveikslėlis jau nuskaitytas ar ne. Jeigu ne, tai jį nuskaito ir įsideda į sąrašą, kad kitą kartą nereikėtų iš naujo nuskaityti paveikslėlį. Kai toks paveikslėlis bus sąraše, tai jį tiesiog iš sąrašo ir gaus. Pritaikius šabloną, žaidimo užsikrovimas pagreitėjo ir yra naudojama mažiau atminties.

- State

  Kortų žaidimui buvo reikalinga saugoti žaidimo stadiją, kuri keičiama gana dažnai ir norint ją pakeisti žaidėjas turėjo pasiekti tam tikrą kartelę – kuo geresnis žaidėjas tuo ryškesnė žaidimo formos spalva. Norint neapsunkinti žaidimo formos klasę jos stadijos keitimo logika ir pačių stadijų savybėmis, buvo panaudotas „State“ projektavimo šablonas, taip atskiriant pagrindinę žaidimo logiką nuo konkrečios jos dalies.

- Chain of responsibility

  Buvo sugalvota atlikti tam tikrą veiksmą priklausomai nuo atverstos kortos parametrų. Veiksmas ar jo reikalavimai turėtų būti nesunkiai keičiamas, jo logika gali skirtis nuo kitų veiksmų. Panaudotas „Chain of Responsibility“ atskirti siunčiamą užklausą nuo konkretaus veiksmo, duoda šansą keliems veiksmas pamatyti užklausą ir nuspręsti ar ji atitinka reikalavimus, šablonas lengviau konfiguruojamas ir palaikomas nei sąlygos sakinys.

- Interpreter

  Kilo problema, kaip paversti romėnišką skaičių į arabišką. Problemai spręsti panaudojome „Interpreter“ projektavimo šabloną. Jis išskaido romėnišką skaičių į tūkstančius, šimtus, dešimtis ir vienetus. Kiekviename žingsnyje vis pridedamas gautas rezultatas prie galutinio rezultato. Suradus dvi kortas su romėniškais skaičiais ir vienodomis spalvomis, skaičių suma yra atspausdinama komandinėje eilutėje. Naudojant šį šabloną, galima mokintis romėniškus skaitmenis.

- Memento

  Žaidime buvo sukurtas naujas langas žaidimo informacijai atvaizduoti. Langas turėjo būti toje pačioje formoje kaip ir žaidimas, tačiau nesinorėjo žaidimo leisti iš naujo pereinant į informacijos langą ir atgal. Naudojamas „Memento“ šablonas išsaugoti formos būseną jai keičiantis tarp langų – navigacijos metu formos duomenys išsaugomi ir juos pakeičia naujo lango duomenys, sugrįžus atgal – išsaugoti duomenys gražinami formai. Formos informacija saugojama tam skirtoje vietoje, ją sugražinti galima bet kuriuo metu.

- Null Object

  Kai yra kuriama korta, kurios simbolis yra neapibrėžtas, yra sukuriamas *null* objektas. Tokio objekto atvaizdavimas žaidime negalimas (žaidimas sustoja). Kad taip neatsitiktų buvo pritaikytas „Null Object“ šablonas. Kai kortos simbolis yra netinkamas, yra sukuriama korta, kurią galima pavaizduoti žaidime tik ji nieko nedarys. Taip žaidimas nenustoja veikti ir kai yra pavaizduojama netinkama korta.

4. Apibendrinimas

Laboratorinio darbo metu šablonai „Template“, „Flyweight“, „State“, „Chain of responsibility“, „Interpreter“, „Memento“ ir „Null Object“ panaudoti tolimesniam kortų žaidimų vystymui. Keli iš jų buvo panaudoti perkuriant ar patobulinant buvusias klases, metodus ir žaidimo logiką („Template“, „Flyweight“, „Interpreter“ ir „Null Object“), kiti – pridedant papildomas funkcijas žaidimui („Memento“, „State“, „Chain of responsibility“). Šie šablonai buvo projektuojami taip, kad būtų galima nesunkiai prijungti papildomą žaidimo logiką – daugumai iš jų prideda abstrakcijos, užtenka tik sukurti konkretų objektą pagal abstrakčias klases.