

Informatikos fakultetas

**T120B516 Objektinis programų projektavimas**

**2-ojo laboratorinio darbo ataskaita**

|  |  |  |
| --- | --- | --- |
|  | Studentai: | Ignas Damijonaitis,  Lukas Raila,  Marius Blažys  IFF-4/1 |
| Dėstytojas: | Andrej Ušaniov |

Kaunas 2017

Turinys

[1. State 3](#_Toc500075198)

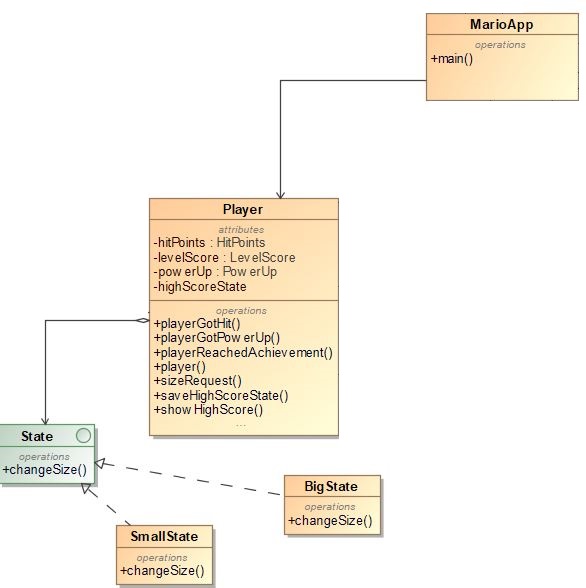
[2. Chain of Responsibility 5](#_Toc500075199)

[3. Memento 7](#_Toc500075200)

[4. Išvados 10](#_Toc500075201)

# State

UML diagrama:



Pav 1 State pattern

Kodo fragmentai:

Player.cs:

public class Player

{

private int hitPoints;

private int levelScore;

private int powerUp;

private HitPoints \_hitPoints;

private LevelScore \_levelScore;

private PowerUp \_powerUp;

GunFactory gunFactory;

Gun gun;

private IState \_state;

private int highScoreState;

public Player(int newHitPoints, int newLevelScore, int newPowerUp, IState state)

{

hitPoints = newHitPoints;

levelScore = newLevelScore;

powerUp = newPowerUp;

\_hitPoints = new HitPoints();

\_levelScore = new LevelScore();

\_powerUp = new PowerUp();

this.\_state = state;

}

public IState IState

{

get { return \_state; }

set

{

\_state = value;

Console.WriteLine("Player's size: " + \_state.GetType().Name);

}

}

public void sizeRequest()

{

\_state.changeSize(this);

}

}

IState.cs:

public interface IState

{

void changeSize(Player player);

}

SmallState.cs:

public class SmallState : IState

{

public void changeSize(Player player)

{

player.IState = new BigState();

}

}

BigState.cs:

public class BigState : IState

{

public void changeSize(Player player)

{

player.IState = new SmallState();

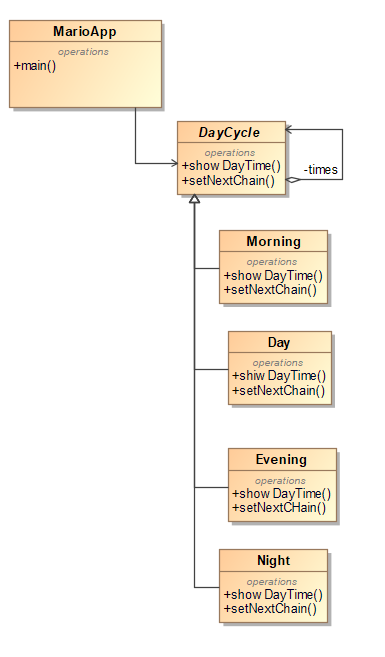
}

}

Šablono tikslas – keisit objekto elgesį, priklausomai nuo būsenos. Šiame pavyzdyje keičiasi žaidėjo dydis per interfeisą „Istate“. „Player“ klasė tiesiogiai neimplementuoja būsenos elgsenos. Tai „Player“ klasę padaro nepriklausomą nuo to, kaip konkreti elgsena yra implementuojama. „SmallState“ ir „BigState“ klasės implementuoja „Istate“ interfeisą. Taip žaidėjas eidamas per trąsą, tai padidėja, tai pamažėja, priklausomai nuo atliktų pasiekimų.

# Chain of Responsibility

UML diagrama:



Pav 2 Chain of Responsibility pattern

Kodo fragmentai:

DayCycle.cs:

abstract class DayCycle

{

protected DayCycle successor;

public void setNextChain(DayCycle successor)

{

this.successor = successor;

}

public abstract void showDayTime(int time);

}

Morning.cs:

class Morning : DayCycle

{

public override void showDayTime(int time)

{

if (time >= 6 && time < 12)

{

Console.WriteLine("It's Morning now!");

}

else if (successor != null)

{

successor.showDayTime(time);

}

}

}

Day.cs:

class Day : DayCycle

{

public override void showDayTime(int time)

{

if (time >= 12 && time < 18)

{

Console.WriteLine("It's Day now!");

}

else if (successor != null)

{

successor.showDayTime(time);

}

}

}

Evening.cs:

class Evening : DayCycle

{

public override void showDayTime(int time)

{

if (time >= 18 && time < 24)

{

Console.WriteLine("It's Evening now!");

}

else if (successor != null)

{

successor.showDayTime(time);

}

}

}

Night.cs:

class Night : DayCycle

{

public override void showDayTime(int time)

{

if (time >= 0 && time < 6)

{

Console.WriteLine("It's Night now!");

}

else if (successor != null)

{

successor.showDayTime(time);

}

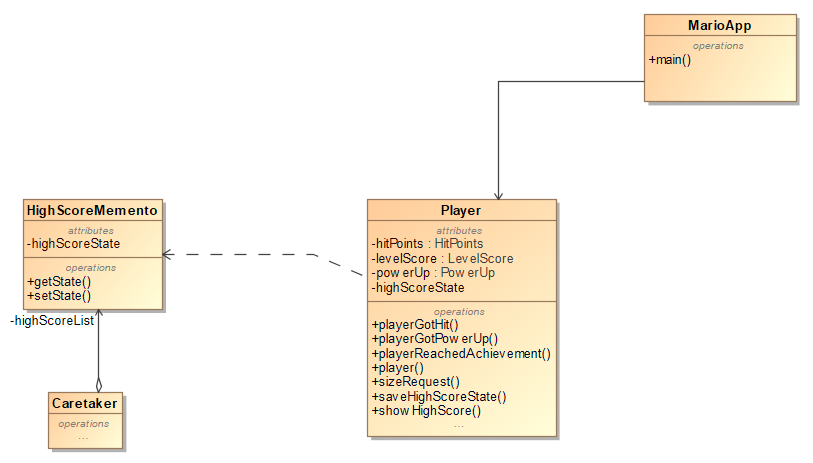
}

}

Chain of Responsibility šablonas leidžia atskirti užklausos gavėją nuo siuntėjo, kad jie nepriklausytų vienas nuo kito ir kad daugiau negu vienas objektas galėtų įvykdyti užklausą. Realizuotu atveju pagal duotą valandą, tikrinama koks dienos metas yra tuo metu žaidime. Šablonas išskaido objektų kūrimą į atskiras klases, kurios gali kisti nepriklausomai.

# Memento

UML diagrama:



Pav 3 Memento pattern

Kodo fragmentai:

Player.cs:

public class Player

{

private int hitPoints;

private int levelScore;

private int powerUp;

private HitPoints \_hitPoints;

private LevelScore \_levelScore;

private PowerUp \_powerUp;

GunFactory gunFactory;

Gun gun;

private IState \_state;

private int highScoreState;

public Player(int newHitPoints, int newLevelScore, int newPowerUp, IState state)

{

hitPoints = newHitPoints;

levelScore = newLevelScore;

powerUp = newPowerUp;

\_hitPoints = new HitPoints();

\_levelScore = new LevelScore();

\_powerUp = new PowerUp();

this.\_state = state;

}

public int HighScoreState

{

get { return highScoreState; }

set

{

highScoreState = value;

Console.WriteLine("High score: " + highScoreState);

}

}

public HighScoreMemento showHighScore()

{

return (new HighScoreMemento(highScoreState));

}

public void saveHighScoreState(HighScoreMemento memento)

{

Console.WriteLine("Restoring high score...");

HighScoreState = memento.HighScoreState;

}

}

HighScoreMemento:

public class HighScoreMemento

{

private int highScoreState;

public HighScoreMemento(int state)

{

this.highScoreState = state;

}

public int HighScoreState

{

get { return highScoreState; }

}

}

Caretaker.cs:

class Caretaker

{

private HighScoreMemento \_memento;

public HighScoreMemento HighScoreMemento

{

set { \_memento = value; }

get { return \_memento; }

}

}

Memento šablonas nepažeidžiant enkapsuliacijos, užfiksuoja objekto būseną, kad tas objektas galėtų vėliau į tą būseną grįžti. Realizuotame pavyzdyje yra saugomas žaidėjo rekordas ir žaidimo metu yra surenkamas naujas rekordas, todėl jeigu naujasis rekordas nepralenkia senojo, rekordo būsena grįžta į pradinę. Šablonas implementuotas trimis objektais – „Player“ (Originator), „HighScoreMemento“ (Memento) ir „Caretaker“. „Player‘ – objektas turintis tą būseną. „Caretaker“ – keičia žaidėjo rekordą, bet gali ir panaikinti pakeitimą.

# Išvados