Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Алтайский государственный технический университет им. И. И. Ползунова»

Факультет информационных технологий

Кафедра прикладной математики

Отчет защищен с оценкой

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_ С. М. Старолетов

(подпись) (и. о., фамилия)

« » 2017 г.

дата

Комплексный отчет

по дисциплине «Архитектурное проектирование и паттерны программирования»

ЛР 09.03.04.16.000О

Студент группы ПИ-42 В.А.Овсянников

(и. о., фамилия)

Преподаватель доцент, к.ф.-м.н. С. М. Старолетов

должность, ученое звание (и. о., фамилия)

Барнаул 2017

**Постановка задачи**

**Предметная область программы**

Эмуляция игры для NES Battle City (Танчики). Игра являет собой поле, на котором расположены танк игрока, вражеские танки, военная база (Орёл). Возможны другие объекты. Танки бывают разных типов. В зависимости от типа они обладают различными характеристиками. Танки умеют стрелять и передвигаться. Игрок управляет своим танком с клавиатуры. Выигрывает сторона, уничтожившая все объекты противников.

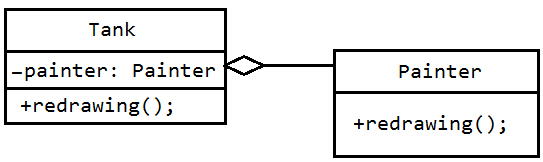
**Используемые паттерны:**

**Делегирование**

***Проблема***: внутри объекта выполняет какое-либо действие, но клиентский код должен оставаться неизменным

***Решение***: делегировать действие другому классу.

**Диаграмма классов:**

****

**Реализация:**

***Tank.cs***

public void redrawing()

{

painter.redrawing(anim);

}

***Painter.cs***

override public void redrawing(int anim)

{

if (tank != null)

{

setImage(tank.DirectionGun, anim);

Canvas.SetBottom(image, tank.CurrentY);

Canvas.SetLeft(image, tank.CurrentX);

}

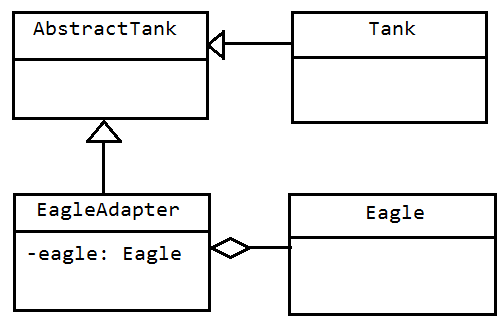
}

**Адаптер**

***Проблема***: Есть независимый класс Eagle, но его объекты нужно обрабатывать как объекты класса Tank во многих случаях (обработка коллизий, повреждение снарядом и т.д.).

***Решение***: создадим класс EagleAdapter, который будет наследоваться от общего с классом Tank супер-класса AbstractTank. EagleAdapter будет содержать в себе ссылку на объект класса Eagle.

## Диаграмма классов:



## Реализация:

***AbstractTank.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace tanki

{

abstract public class AbstractTank

{

protected double currentX;

protected double currentY;

protected Performance performance;

protected string directionGun;

protected int anim;

protected string name;

public string Name

{

get { return name; }

}

public int Anim { get { return anim; } }

public string DirectionGun

{

get { return directionGun; }

set { directionGun = value; }

}

public double CurrentX

{

get { return currentX; }

set { currentX = value; }

}

public double CurrentY

{

get { return currentY; }

set { currentY = value; }

}

public Performance Performance

{

get { return performance; }

}

abstract public void move(List<AbstractTank> tanks);//движение танка

abstract public void addBullet(List<Charge> bullets);//добавляет новый встрел

}

}

***Tank.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows.Controls;

using System.Windows.Media.Imaging;

using System.Media;

using System.Windows.Media;

using System.Windows.Threading;

namespace tanki

{

public class Tank : AbstractTank

{

//protected Field field;

protected string direction;

public Tank(string tankName, double start\_x, double start\_y, string \_directionGun)

{

name = tankName;

directionGun = \_directionGun;

direction = Settings.stop;

currentX = start\_x;

currentY = start\_y;

FactoryTank myFactory = getFactory(tankName);

performance = myFactory.getPerformance();

}

public FactoryTank getFactory(string name)

{

FactoryTank factory = null;

switch (name)

{

case Settings.nameTankBTR:

factory = new FactoryBTR(this);

break;

case Settings.nameTankHeavy:

factory = new FactoryHeavy(this);

break;

case Settings.nameTankLight:

factory = new FactoryLight(this);

break;

default:

break;

}

return factory;

}

public string Direction

{

get { return direction; }

set { direction = value; }

}

//основные методы

override public void move(List<AbstractTank> tanks)//движение танка

{

switch (direction)

{

case Settings.up:

movingUp(tanks);

anim++;

break;

case Settings.down:

movingDown(tanks);

anim++;

break;

case Settings.left:

movingLeft(tanks);

anim++;

break;

case Settings.right:

movingRight(tanks);

anim++;

break;

default:

break;

}

}

override public void addBullet(List<Charge> bullets)//добавляет новый встрел

{

if (!Performance.Gun.Blast)

{

Sounds.soundOfBlast.Stop();

Performance.Gun.addBullet(bullets);

Sounds.soundOfBlast.Play();

Performance.Gun.Blast = true;

}

}

//вспомогательные методы

public void directionChoice(string dir)//устанавливает направление танка

{

direction = dir;

directionGun = dir;

}

public void movingLeft(List<AbstractTank> tanks)//движение влево

{

if (isEmpty(tanks, CurrentX - Performance.Speed, CurrentY))

CurrentX -= Performance.Speed;

}

public void movingRight(List<AbstractTank> tanks)//движение вправо

{

if (isEmpty(tanks, CurrentX + Performance.Speed, CurrentY))

CurrentX += Performance.Speed;

}

public void movingDown(List<AbstractTank> tanks)////движение вниз

{

if (isEmpty(tanks, CurrentX, CurrentY - Performance.Speed))

CurrentY -= Performance.Speed;

}

public void movingUp(List<AbstractTank> tanks)//движение вверх

{

if (isEmpty(tanks, CurrentX, CurrentY + Performance.Speed))

CurrentY += Performance.Speed;

}

public bool isEmpty(List<AbstractTank> tanks, double x1, double y1)//проверяет может танк переместиться

{

foreach (AbstractTank tank in tanks)

if (tank != this)

{

if (Math.Abs(tank.CurrentX - x1) <= Settings.sizeTank && Math.Abs(tank.CurrentY - y1) <= Settings.sizeTank)

return false;

}

if (x1 < 0 || x1 > Settings.width - Settings.sizeTank || y1 < 0 || y1 > Settings.height - Settings.sizeTank)

return false;

return true;

}

}

}

***Eagle.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace tanki

{

public class Eagle

{

protected double currentX;

protected double currentY;

public Eagle()

{

currentX = Settings.posEagleX;

currentY = 0;

}

public double CurrentX

{

get { return currentX; }

}

public double CurrentY

{

get { return currentY; }

}

}

}

***EagleAdapter.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace tanki

{

class EagleAdapter : AbstractTank

{

private Eagle eagle;

public EagleAdapter()

{

eagle = new Eagle();

currentX = eagle.CurrentX;

currentY = eagle.CurrentY;

performance = new PerformanceLight(this);

name = Settings.nameEagle;

}

override public void move(List<AbstractTank> tanks)

{

}

override public void addBullet(List<Charge> bullets)

{

}

}

}

**Пример:**

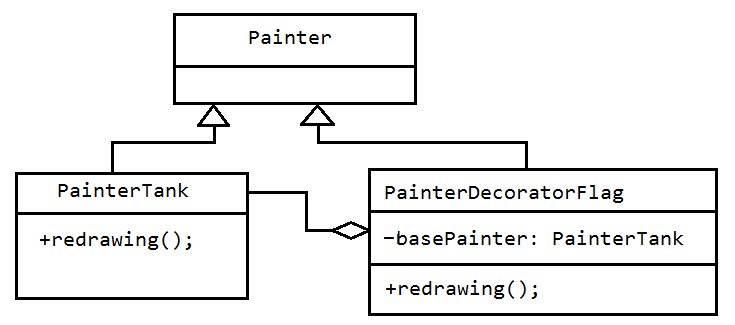


**Декоратор**

***Проблема***: необходимо наделить текущий класс новым поведением без изменения кода класса. Например, обозначить танки на поле разными флагами.

***Решение***: Ввести класс PainterDecoratorFlag, который будет наследоваться от того же базового класса Painter, что и отрисовщик танка PainterTank и хранить ссылку на оригинальный отрисовщик внутри себя.

**Диаграмма классов:**

****

**Реализация:**

***Painter.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows.Controls;

using System.Windows.Media.Imaging;

namespace tanki

{

abstract class Painter

{

protected Tank tank;

protected string source;

protected Field field;

protected Image image;

public string Source

{

get { return source; }

set { source = value; }

}

abstract public void setTank(Tank \_tank);

abstract public void deleteImage();

abstract public void setImage(string direction, int a);

abstract public void redrawing(int anim);

}

}

***PainterTank.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows.Controls;

using System.Windows.Media.Imaging;

namespace tanki

{

class PainterTank : Painter

{

public PainterTank(Field \_field, string \_source)

{

field = \_field;

source = \_source;

image = new Image();

field.Canvas.Children.Add(image);

}

override public void setTank(Tank \_tank)

{

tank = \_tank;

}

override public void setImage(string direction, int a)

{

int anim = a % 2;

BitmapImage bi3 = new BitmapImage();

string path = source + direction + anim.ToString() + Settings.expansion;

bi3.BeginInit();

bi3.UriSource = new Uri(path, UriKind.Relative);

bi3.EndInit();

image.Source = bi3;

}

override public void redrawing(int anim)

{

if (tank != null)

{

setImage(tank.DirectionGun, anim);

Canvas.SetBottom(image, tank.CurrentY);

Canvas.SetLeft(image, tank.CurrentX);

}

}

override public void deleteImage()

{

field.Canvas.Children.Remove(image);

}

}

}

***PainterDecoratorFlag.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows.Controls;

using System.Windows.Media.Imaging;

namespace tanki

{

class PainterDecoratorFlag : Painter

{

private PainterTank basePainter;

public PainterDecoratorFlag(Field \_field, string \_source, string \_sourceFlag)

{

basePainter = new PainterTank(\_field, \_source);

field = \_field;

source = \_sourceFlag;

image = new Image();

field.Canvas.Children.Add(image);

}

override public void setTank(Tank \_tank)

{

tank = \_tank;

basePainter.setTank(\_tank);

}

override public void setImage(string direction, int a)

{

basePainter.setImage(direction, a);

BitmapImage bi3 = new BitmapImage();

bi3.BeginInit();

bi3.UriSource = new Uri(source, UriKind.Relative);

bi3.EndInit();

image.Source = bi3;

}

override public void redrawing(int anim)

{

basePainter.redrawing(anim);

setImage(tank.DirectionGun, anim);

Canvas.SetBottom(image, tank.CurrentY);

Canvas.SetLeft(image, tank.CurrentX);

}

override public void deleteImage()

{

basePainter.deleteImage();

field.Canvas.Children.Remove(image);

}

}

}

**Пример:**

****

**Композит**

***Проблема***: Необходимо управлять множеством объектов как одним объектом. Например, можно создать танковый взвод и управлять им как одним танком.

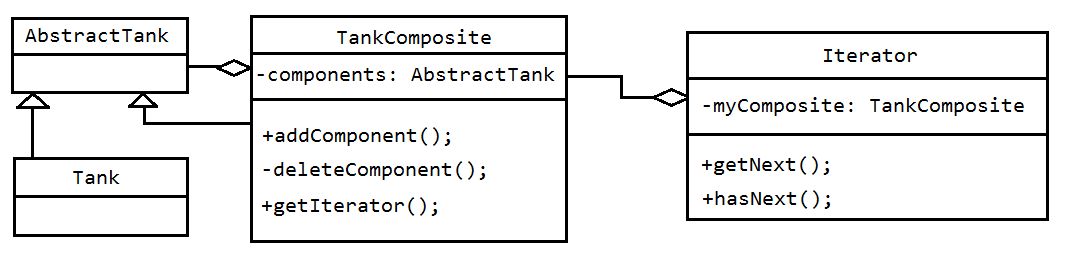
***Решение***: создать класс TankComposite, который будет наследоваться от общего супер-класса AbstractTank и будет содержать в себе коллекцию объектов базового класса, а также переопределит его методы.

**Итератор**

***Проблема***: необходим последовательный перебор элементов коллекции (в данном случае, нашего композита).

***Решение***: создать независимый класс, который будет содержать ссылку на конкретный композит и предоставит методы для доступа к его элементам.

**Диаграмма классов:**



**Реализация:**

***AbsrtactTank.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace tanki

{

abstract public class AbstractTank

{

protected double currentX;

protected double currentY;

protected Performance performance;

protected Painter painter;

protected string name;

protected string directionGun;

protected string direction;

protected int anim;

public string Name

{

get { return name; }

}

public string DirectionGun

{

get { return directionGun; }

set { directionGun = value; }

}

public string Direction

{

get { return direction; }

set { direction = value; }

}

public Painter Painter

{

get { return painter; }

set { painter = value; }

}

public double CurrentX

{

get { return currentX; }

set { currentX = value; }

}

public double CurrentY

{

get { return currentY; }

set { currentY = value; }

}

public Performance Performance

{

get { return performance; }

}

abstract public void death(List<AbstractTank> tankList);

abstract public void move(List<AbstractTank> tanks);//движение танка

abstract public void redrawing();//перерисовка танка

abstract public void addBullet(List<Charge> bullets);//добавляет новый встрел

abstract public void directionChoice(string dir);

abstract public void directionGunChoice(string dir);

abstract public void comparer(AbstractTank tank, double x1, double y1, ref bool flag);

abstract public void comparer(Charge bullet, double x1, double y1, double x3, double y3, ref bool flag);

}

}

***Tank.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows.Controls;

using System.Windows.Media.Imaging;

using System.Media;

using System.Windows.Media;

using System.Windows.Threading;

namespace tanki

{

public class Tank : AbstractTank

{

//protected Field field;

public Tank(string tankName, double start\_x, double start\_y, string \_directionGun)

{

name = tankName;

directionGun = \_directionGun;

direction = Settings.stop;

currentX = start\_x;

currentY = start\_y;

switch (tankName)

{

case Settings.nameTankBTR:

performance = new Performance(new Gun(this,Settings.damageBullet,Settings.speedBullet), Settings.speedTankBTR, Settings.armourLight);

painter = new PainterTank(Settings.sourceTankBTR);

break;

case Settings.nameTankLight:

performance = new Performance(new Gun(this,Settings.damageBullet,Settings.speedBullet), Settings.speedTankLight, Settings.armourLight);

painter = new PainterTank(Settings.sourceTankLight);

break;

case Settings.nameTankHeavy:

performance = new Performance(new Gun(this,Settings.damageBullet,Settings.speedBullet), Settings.speedTankHeavy, Settings.armourHeavy);

painter = new PainterTank(Settings.sourceTankHeavy);

break;

default:

break;

}

painter.setTank(this);

}

public Tank(string tankName, double start\_x, double start\_y, string \_directionGun, string decorator)

{

name = tankName;

directionGun = \_directionGun;

direction = Settings.stop;

currentX = start\_x;

currentY = start\_y;

switch (tankName)

{

case Settings.nameTankBTR:

performance = new Performance(new Gun(this, Settings.damageBullet, Settings.speedBullet), Settings.speedTankBTR, Settings.armourLight);

painter = new PainterDecoratorFlag(Settings.sourceTankBTR, decorator);

break;

case Settings.nameTankLight:

performance = new Performance(new Gun(this, Settings.damageBullet, Settings.speedBullet), Settings.speedTankLight, Settings.armourLight);

painter = new PainterDecoratorFlag(Settings.sourceTankLight, decorator);

break;

case Settings.nameTankHeavy:

performance = new Performance(new Gun(this, Settings.damageBullet, Settings.speedBullet), Settings.speedTankHeavy, Settings.armourHeavy);

painter = new PainterDecoratorFlag(Settings.sourceTankHeavy, decorator);

break;

default:

break;

}

painter.setTank(this);

}

//основные методы

override public void move(List<AbstractTank> tanks)//движение танка

{

switch (direction)

{

case Settings.up:

movingUp(tanks);

anim++;

break;

case Settings.down:

movingDown(tanks);

anim++;

break;

case Settings.left:

movingLeft(tanks);

anim++;

break;

case Settings.right:

movingRight(tanks);

anim++;

break;

default:

break;

}

}

override public void redrawing()//перерисовка танка

{

painter.redrawing(anim);

}

override public void addBullet(List<Charge> bullets)//добавляет новый встрел

{

if (!Performance.Gun.Blast)

{

Sounds.soundOfBlast.Stop();

performance.Gun.addBullet(bullets);

Sounds.soundOfBlast.Play();

performance.Gun.Blast = true;

}

}

//вспомогательные методы

override public void directionChoice(string dir)//устанавливает направление танка

{

direction = dir;

//directionGun = dir;

}

override public void directionGunChoice(string dir)//устанавливает направление танка

{

direction = dir;

directionGun = dir;

}

override public void death(List<AbstractTank> tankList)//обрабатывает смерть танка

{

if (performance.HealthPoints <= 0)

{

//tankList[i].death();

tankList.Remove(this);

Sounds.bang1.Stop();

Sounds.bang1.Play();

Painter.deleteImage();

}

// Painter.deleteImage();

}

public void movingLeft(List<AbstractTank> tanks)//движение влево

{

if (isEmpty(tanks, CurrentX - Performance.Speed, CurrentY))

CurrentX -= Performance.Speed;

}

public void movingRight(List<AbstractTank> tanks)//движение вправо

{

if (isEmpty(tanks, CurrentX + Performance.Speed, CurrentY))

CurrentX += Performance.Speed;

}

public void movingDown(List<AbstractTank> tanks)////движение вниз

{

if (isEmpty(tanks, CurrentX, CurrentY - Performance.Speed))

CurrentY -= Performance.Speed;

}

public void movingUp(List<AbstractTank> tanks)//движение вверх

{

if (isEmpty(tanks, CurrentX, CurrentY + Performance.Speed))

CurrentY += Performance.Speed;

}

public bool isEmpty(List<AbstractTank> tanks, double x1, double y1)//проверяет может танк переместиться

{

/\*

foreach (AbstractTank tank in tanks)

if (tank != this)

{

if (Math.Abs(tank.CurrentX - x1) <= Settings.sizeTank && Math.Abs(tank.CurrentY - y1) <= Settings.sizeTank)

return false;

}\*/

bool flag = true;

foreach (AbstractTank tank in tanks)

tank.comparer(this, x1, y1, ref flag);

if (!flag)

return false;

if (x1 < 0 || x1 > Settings.width - Settings.sizeTank || y1 < 0 || y1 > Settings.height - Settings.sizeTank)

return false;

return true;

}

override public void comparer(AbstractTank tank, double x1, double y1, ref bool flag)

{

if (tank != this)

if (Math.Abs(this.CurrentX - x1) < Settings.sizeTank && Math.Abs(this.CurrentY - y1) < Settings.sizeTank)

flag = false;

}

override public void comparer(Charge bullet, double x1, double y1, double x3, double y3, ref bool flag)

{

if(this!=bullet.Tank)

{

double x2 = currentX;

double y2 = currentY;

double x4 = x2 + Settings.sizeTank;

double y4 = y2 + Settings.sizeTank;

if ((Math.Abs(x2 - x1) <= Settings.sizeBullet ||

Math.Abs(x3 - x4) <= Settings.sizeBullet) &&

(Math.Abs(y2 - y1) <= Settings.sizeBullet ||

Math.Abs(y3 - y4) <= 2\* Settings.sizeBullet))

{

Console.WriteLine(performance.HealthPoints);

performance.HealthPoints -= bullet.Tank.Performance.Gun.Damage;

flag = false;

//return;

}

}

//return true;

}

}

}

***TankComposite.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace tanki

{

public class TankComposite : AbstractTank

{

private List<AbstractTank> components;

public TankComposite()

{

name = "Compsite";

components = new List<AbstractTank>();

performance = new Performance(200);

currentX = 1000;

currentY = 1000;

}

public List<AbstractTank> Components

{

get { return components; }

}

public void addComponent(AbstractTank component)

{

components.Add(component);

}

public void deleteComponent(AbstractTank component)

{

components.Remove(component);

}

override public void move(List<AbstractTank > tankList)

{

foreach (AbstractTank component in components)

component.move(tankList);

}

override public void death(List<AbstractTank> tankList)

{

for (int i = 0; i < components.Count;i++ )

components[i].death(components);

if (components.Count < 1)

{

tankList.Remove(this);

}

}

override public void addBullet(List<Charge> bullets)

{

foreach (AbstractTank component in components)

component.addBullet(bullets);

}

override public void redrawing()

{

foreach (AbstractTank component in components)

component.redrawing();

}

public Iterator getIterator()

{

return new Iterator(this);

}

override public void directionChoice(string dir)//устанавливает направление танка

{

direction = dir;

foreach (AbstractTank tank in components)

{

tank.Direction = dir;

}

}

override public void directionGunChoice(string dir)//устанавливает направление танка

{

direction = dir;

directionGun = dir;

foreach (AbstractTank tank in components)

{

tank.Direction = dir;

tank.DirectionGun = dir;

}

}

override public void comparer(AbstractTank tank, double x1, double y1, ref bool flag)

{

foreach (AbstractTank component in components)

component.comparer(tank, x1, y1, ref flag);

}

override public void comparer(Charge bullet, double x1, double y1, double x3, double y3, ref bool flag)

{

foreach (AbstractTank component in components)

component.comparer( bullet, x1, y1, x3,y3,ref flag);

//return true;

}

}

}

***Itertor.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace tanki

{

public class Iterator

{

private TankComposite myComposite;

private int index;

public Iterator(TankComposite composite)

{

myComposite = composite;

index = -1;

}

public bool hasNext()

{

if (index < myComposite.Components.Count - 1)

return true;

return false;

}

public AbstractTank getNext()

{

if (hasNext())

return myComposite.Components[++index];

return null;

}

public AbstractTank getCurrent()

{

if (hasNext())

return myComposite.Components[index];

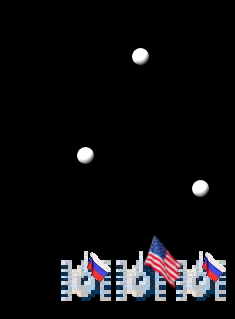
return null;

}

}

}

**Пример**

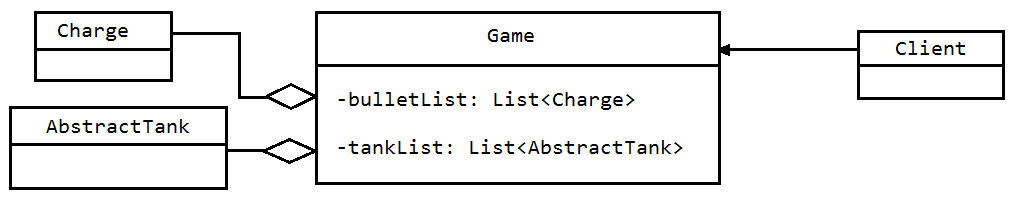


**Фасад**

***Проблема***: в системе много взаимодействующих классов. При добавлении нового придётся менять код клиента.

***Решение***: класс Game предоставляет клиенту набор методов, которые управляют основными процессами. Набор методов для клиента всегда остаётся неизменным. При добавлении новых компонентов или функций все изменения происходят внутри класса Game.

**Диаграмма классов:**

****

**Реализация:**

***Game.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows.Controls;

using System.Windows.Input;

namespace tanki

{

class Game

{

private Drawer myDrawer;

private Tank current;

private List<Charge> bulletList;

private List<AbstractTank> tankList;

private KeyStack myKeyStack;

public Game(Canvas mainCanvas)

{

tankList = new List<AbstractTank>();

Sounds.startSounds();

myKeyStack = new KeyStack();

bulletList = new List<Charge>();

myDrawer = new DrawerConsole(mainCanvas);

myDrawer = new DrawerGraphics(mainCanvas,tankList,bulletList);

}

public void newDrawerConsole(Canvas mainCanvas)

{

myDrawer = new DrawerConsole(mainCanvas);

}

public void newDrawerGraphics(Canvas mainCanvas)

{

myDrawer = new DrawerGraphics(mainCanvas, tankList, bulletList);

}

public KeyStack MyKeyStack

{

get { return myKeyStack; }

}

public Drawer Drawer

{

get { return myDrawer; }

set {myDrawer = value; }

}

public Tank Current

{

get { return current; }

set { current = value; }

}

private void shooting()

{

for (int i=0;i<bulletList.Count;i++)

bulletList[i].shooting(tankList, bulletList,myDrawer);

}//стрельба всех выстрелов

private void death()

{

for (int i = 0; i < tankList.Count; i++)

if (tankList[i].Performance.HealthPoints <= 0)

{

myDrawer.deleteTank(i);

tankList.Remove(tankList[i]);

Sounds.bang1.Stop();

Sounds.bang1.Play();

}

}//проверка танка на смерть

private void addBullet(Tank tank)

{

tank.addBullet(bulletList);

}//добавление пули конкретного танка

public void currentBlasted()

{

current.addBullet(bulletList);

myDrawer.addBullet();

}//добавление пули текущего танка

public string directionChoice(Key key)

{

string direction = Settings.stop;

switch (key)

{

case Settings.keyUp:

direction = Settings.up;

break;

case Settings.keyLeft:

direction = Settings.left;

break;

case Settings.keyDown:

direction = Settings.down;

break;

case Settings.keyRight:

direction = Settings.right;

break;

default:

break;

}

return direction;

}//выбор направления движения

public void addPlayer(string tankName, double start\_x, double start\_y, string \_directionGun, string flag)

{

Tank tank = new Tank(tankName, start\_x, start\_y, \_directionGun);

current = tank;

tankList.Add(tank);

myDrawer.addTank(tankName);

}//добавить танк игрока

public void addEnemy(string tankName, double start\_x, double start\_y, string \_directionGun, string flag)

{

Tank tank = new Tank(tankName, start\_x, start\_y, \_directionGun);

tankList.Add(tank);

myDrawer.addTank(tankName);

}//добавить танк врага

public void addPlayer(string tankName, double start\_x, double start\_y, string \_directionGun)

{

Tank tank = new Tank(tankName, start\_x, start\_y, \_directionGun);

current = tank;

tankList.Add(tank);

myDrawer.addTank(tankName);

}//добавить танк игрока

public void addEagle(EagleAdapter eagle)

{

tankList.Add(eagle);

myDrawer.addEagle();

}

public void addEnemy(string tankName, double start\_x, double start\_y, string \_directionGun)

{

Tank tank = new Tank(tankName, start\_x, start\_y, \_directionGun);

tankList.Add(tank);

myDrawer.addTank(tankName);

}//добавить танк врага

public void tanksMoving()

{

foreach (AbstractTank tank in tankList)

tank.move(tankList);

}//обработка движения танков

public bool isSoundPlayer()

{

if (current.Direction != Settings.stop)

return true;

return false;

}//проверяет, звучит ли танк игрока

public bool isSoundEnemy()

{

return false;

}//проверяет, звучит ли вражеский танк

public void currentControl()

{

if (myKeyStack.Keys.Count > 0)

current.directionChoice(directionChoice(myKeyStack.Keys[myKeyStack.Keys.Count - 1]));

else

current.Direction = Settings.stop;

}//задаёт направление текущего танка в соотвествии с нажатой клавишей

public void redrawing()

{

myDrawer.drawing(tankList, bulletList);

}//перерисовка всех танков

public void timerTick()

{

death();

shooting();

currentControl();

tanksMoving();

redrawing();

//if (tanks.Count > 0)

{

if (isSoundPlayer())

{

Sounds.soundEnemy.Stop();

Sounds.soundPlayer.Play();

}

else

{

Sounds.soundPlayer.Stop();

if (isSoundEnemy())

Sounds.soundEnemy.Play();

}

}

}//игровой тик

public void startTheGame()//начало игры

{

addEagle(new EagleAdapter());

//addPlayer(new Tank(Settings.nameTankLight, Settings.posPlayerOneX, Settings.posPlayerY, Settings.up, Settings.sourceFlagRussia));

addPlayer(Settings.nameTankLight, Settings.posPlayerOneX, Settings.posPlayerY, Settings.up);

//addPlayer(new Tank(Settings.nameTankHeavy, Settings.posPlayerOneX, Settings.posPlayerY, Settings.up, Settings.sourceFlagRussia));

addEnemy(Settings.nameTankLight, Settings.posEnemyOneX+100, Settings.posEnemyY, Settings.down, Settings.sourceFlagUSA);

addEnemy(Settings.nameTankHeavy, Settings.posEnemyTwoX, Settings.posEnemyY, Settings.down);

addEnemy(Settings.nameTankBTR, Settings.posEnemyThreeX, Settings.posEnemyY, Settings.down, Settings.sourceFlagUSA);

}//

}

}

**Приспособленец**

***Проблема***: в системе присутствует большое количество одинаковых объектов. Затраты на их хранение велики.

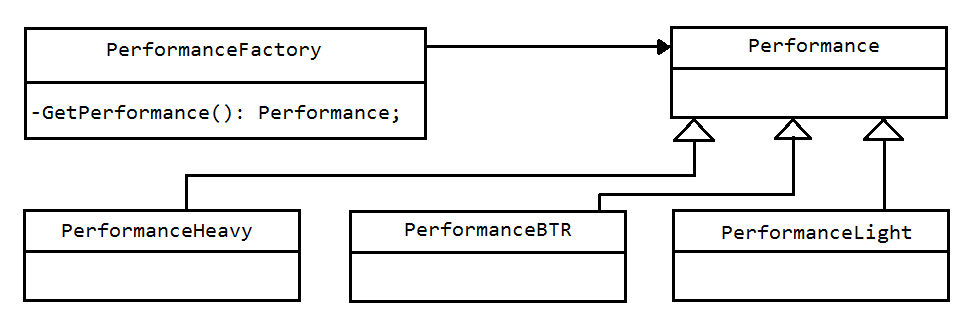
***Решение***: давать объекту Tank уже существующий экземпляр Performance, когда это необходимо, а не создавать новый, или создать новый при отсутствии имеющегося.

**Фабричный метод:**

***Проблема***: в системе есть объекты нескольких подклассов Performance. Неизвестно, какой тип нужно создать в данный момент.

***Решение***: создать дополнительный класс для создания экземпляров.

**Диаграмма классов:**

****

**Реализация:**

***Performance.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace tanki

{

public abstract class Performance

{

protected Gun gun;

protected int speed;

protected int healthPoints;

public int Speed

{

get { return speed; }

set { speed = value; }

}

public int HealthPoints

{

get { return healthPoints; }

set { healthPoints = value; }

}

public Gun Gun

{

get { return gun; }

}

}

}

***PerformanceBTR.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace tanki

{

public class PerformanceBTR : Performance

{

public PerformanceBTR(AbstractTank tank)

{

gun = new Gun(tank, Settings.damageBullet, Settings.speedBullet);

healthPoints = Settings.armourLight;

speed = Settings.speedTankBTR;

}

}

}

***PerformanceHeavy.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace tanki

{

public class PerformanceHeavy : Performance

{

public PerformanceHeavy(AbstractTank tank)

{

gun = new Gun(tank, Settings.damageRocket, Settings.speedRocket);

healthPoints = Settings.armourHeavy;

speed = Settings.speedTankHeavy;

}

}

}

***PerformanceLight.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace tanki

{

public class PerformanceLight : Performance

{

public PerformanceLight(AbstractTank tank)

{

gun = new Gun(tank, Settings.damageBullet, Settings.speedBullet);

healthPoints = Settings.armourLight;

speed = Settings.speedTankLight;

}

}

}

***PerformanceFactory.cs***

using System;

using System.Collections.Generic;

using System.Collections;

using System.Linq;

using System.Text;

namespace tanki

{

public class PerformanceFactory

{

private Hashtable performances = new Hashtable();

public Performance GetPerformance(string key, AbstractTank tank)

{

Performance performance = performances[key] as Performance;

if (performance == null)

{

switch (key)

{

case Settings.nameTankBTR:

performance = new PerformanceBTR(tank);

break;

case Settings.nameTankLight:

performance = new PerformanceLight(tank);

break;

case Settings.nameTankHeavy:

performance = new PerformanceHeavy(tank);

break;

}

performances.Add(key, performance);

}

return performance;

}

}

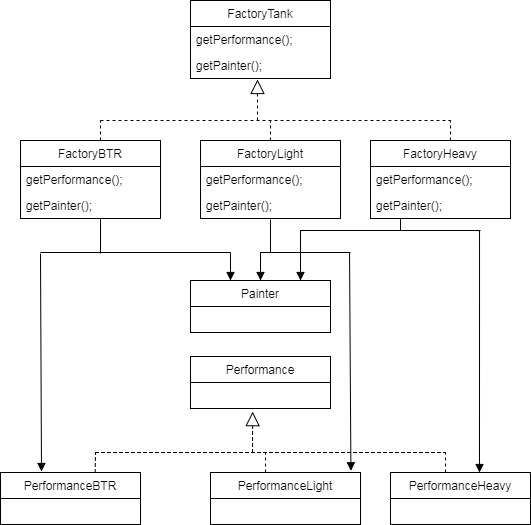
}

**Абстрактная фабрика**

***Проблема***: создаваемый объект Tank требует набор параметров определённого типа

***Решение***: передать создание объекта отдельному классу, Который будет скрывать логику создания объектов от клиента.

**Диаграмма классов:**

****

**Реализация:**

***FactoryTank.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace tanki

{

abstract public class FactoryTank

{

protected AbstractTank tank;

abstract public Performance getPerformance();

}

}

***FactoryPainter.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows.Controls;

namespace tanki

{

abstract public class FactoryPainter

{

//protected Canvas canvas;

abstract public Painter getPainter(Canvas canvas);

}

}

***FactoryPainterBTR.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows.Controls;

namespace tanki

{

class FactoryPainterBTR : FactoryPainter

{

public FactoryPainterBTR()

{

}

override public Painter getPainter(Canvas canvas)

{

return new PainterTank(Settings.sourceTankBTR, canvas);

}

}

}

***FactoryPainterLight.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows.Controls;

namespace tanki

{

class FactoryPainterLight : FactoryPainter

{

public FactoryPainterLight()

{

}

override public Painter getPainter(Canvas canvas)

{

return new PainterTank(Settings.sourceTankLight, canvas);

}

}

}

***FactoryPainterHeavy.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows.Controls;

namespace tanki

{

class FactoryPainterHeavy : FactoryPainter

{

public FactoryPainterHeavy()

{

}

override public Painter getPainter(Canvas canvas)

{

return new PainterTank(Settings.sourceTankHeavy, canvas);

}

}

}

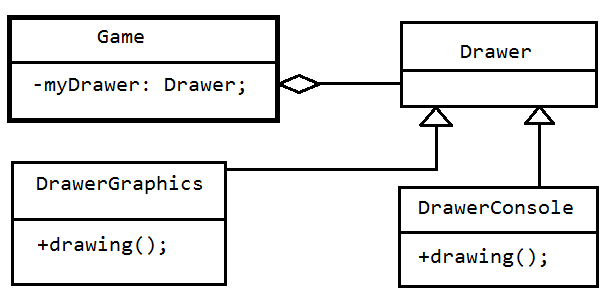
**Мост**

***Проблема***: система должна быть переносима на другие платформы.

***Решение***: отделить реализацию от абстракции. Вынести всю логику игры в отдельные классы. Отрисовку назначить другому классу. Отрисовщик берёт всю необходимую информацию из класса логики и в зависимости от своего типа обрабатывает.

***Результат***: получим систему, визуализация которой не зависит от логики, а зависит только от выбранного отрисовщика.

**Диаграмма классов:**

****

**Реализация:**

***Drawer.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows.Controls;

namespace tanki

{

abstract public class Drawer

{

abstract public void addBullet();

abstract public void addTank(string tankName);

abstract public void addEagle();

abstract public void drawing(List<AbstractTank> tankList, List<Charge> bulletList);

abstract public void deleteTank(int index);

abstract public void deleteBullet();

}

}

***DrawerConsole.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows.Controls;

using System.Windows.Media;

namespace tanki

{

public class DrawerConsole : Drawer

{

private TextBox textBox;

private int n;

private int m;

private string[] eagle = { " ▄▄██▄▄ ", " ▀▀▐▌▀▀ ", " ████████ ", "▀██▄▐▌▄███", "█▄ ▀██▄ ▄█" };

private string[] lightTankUp = { "▀▀▀ ▀▀▀", "▀▀▀█▄▄█▀▀▀", "▀▀▀█▀▀█▀▀▀", "▀▀▀ ▐▌ ▀▀▀", " ▐▌ " };

private string[] lightTankDown = { " ▐▌ ", "▄▄▄ ▐▌ ▄▄▄", "▄▄▄█▄▄█▄▄▄", "▄▄▄█▀▀█▄▄▄", "▄▄▄ ▄▄▄" };

private string[] lightTankRight = { " █ █ █ █ ", " ▄▀█▀█ ▄ ", " █ █▬▬▬▬", " ▀▄█▄█ ▀ ", " █ █ █ █ " };

private string[] lightTankLeft = { " █ █ █ █ ", " ▄ █▀█▀▄ ", "▬▬▬▬█ █ ", " ▀ █▄█▄▀ ", " █ █ █ █ " };

private string[] heavyTankUp = { "▀▀▀ ▀▀▀", "▀▀▀████▀▀▀", "▀▀▀█ █▀▀▀", "▀▀▀█▀▀█▀▀▀", "▀▀▄ ██ ▄▀▀" };

private string[] heavyTankDown = { "▄▄▀ ██ ▀▄▄", "▄▄▄█▄▄█▄▄▄", "▄▄▄█ █▄▄▄", "▄▄▄████▄▄▄", "▄▄▄ ▄▄▄" };

private string[] heavyTankRight = { " █ █ █ █ █", " ▄▀█▀█▀█▄ ", " ██ ███", " ▀▄█▄█▄█▀ ", " █ █ █ █ █" };

private string[] heavyTankLeft = { "█ █ █ █ █ ", " ▄█▀█▀█▀▄ ", "███ ██ ", " ▀█▄█▄█▄▀ ", "█ █ █ █ █ " };

private string[] BTRUp = { "▀██▀▀▀▀██▀", "▄██▄▄▄▄██▄", "███ ██ ███", "▀██▀▀▀▀██▀", "▄▄▄ ▐▌ ▄▄▄" };

private string[] BTRDown = { "▀▀▀ ▐▌ ▀▀▀", "▄██▄▄▄▄██▄", "███ ██ ███", "▀██▀▀▀▀██▀", "▄██▄▄▄▄██▄", };

private string[] BTRRight = { "▀██▀██▀██ ", "▄██▄▄▄▄█▄ ", " ██ ██ █▬▬", "▀██▀▀▀▀█▀ ", "▄██▄██▄██ " };

private string[] BTRLeft = { " ██▀██▀██▀", " ▄█▄▄▄▄██▄", "▬▬█ ██ ██ ", " ▀█▀▀▀▀██▀", " ██▄██▄██▄" };

public DrawerConsole(Canvas mainCanvas)

{

textBox = new TextBox();

textBox.Width = Settings.width;

textBox.Height = Settings.height;

textBox.FontFamily = new FontFamily("Courier new");

textBox.FontSize = 4;

textBox.Background = Brushes.Black;

textBox.Foreground = Brushes.White;

textBox.Focusable = false;

n = Settings.height / 10;

m = Settings.width / 5;

mainCanvas.Children.Add(textBox);

}

override public void addEagle()

{

}

override public void addBullet()

{

}

override public void deleteBullet()

{

}

override public void addTank(string tankName)

{

}

override public void deleteTank(int index)

{

}

override public void drawing(List<AbstractTank> \_tankList, List<Charge> bulletList)

{

textBox.Clear();

char[,] buffer = new char[n, m];

for (int i = 0; i < n; i++)

for (int j = 0; j < m; j++)

buffer[i, j] = ' ';

for (int index = 0; index < \_tankList.Count; index++)

{

string[] tank = null;

int i = Convert.ToInt32(\_tankList[index].CurrentY / 10);

int j = Convert.ToInt32(\_tankList[index].CurrentX / 5);

switch (\_tankList[index].Name)

{

case Settings.nameTankBTR:

switch (\_tankList[index].DirectionGun)

{

case Settings.up:

tank = BTRUp;

break;

case Settings.down:

tank = BTRDown;

break;

case Settings.left:

tank = BTRLeft;

break;

case Settings.right:

tank = BTRRight;

break;

default:

break;

}

break;

case Settings.nameTankLight :

switch (\_tankList[index].DirectionGun)

{

case Settings.up:

tank = lightTankUp;

break;

case Settings.down:

tank = lightTankDown;

break;

case Settings.left:

tank = lightTankLeft;

break;

case Settings.right:

tank = lightTankRight;

break;

default:

break;

}

break;

case Settings.nameTankHeavy:

switch (\_tankList[index].DirectionGun)

{

case Settings.up:

tank = heavyTankUp;

break;

case Settings.down:

tank = heavyTankDown;

break;

case Settings.left:

tank = heavyTankLeft;

break;

case Settings.right:

tank = heavyTankRight;

break;

default:

break;

}

break;

case Settings.nameEagle:

tank = eagle;

break;

default:

break;

}

for (int ii = i; ii < i+5 ; ii++)

for (int jj = j; jj < j + 10; jj++)

buffer[ii, jj] = tank[ii-i][jj - j];

}

foreach (Charge bullet in bulletList)

{

int i = Convert.ToInt32(bullet.CurrentY / 10);

int j = Convert.ToInt32(bullet.CurrentX / 5);

buffer[i, j] = '█';

buffer[i, j + 1] = '█';

}

string str = "";

for (int i = n - 1; i >= 0; i--)

{

for (int j = 0; j < m; j++)

str += buffer[i, j];

str += "\n";

}

textBox.Text = str;

}

}

}

***DrawerGraphics.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows.Controls;

using System.Windows.Media.Imaging;

using System.Windows.Media;

namespace tanki

{

public class DrawerGraphics : Drawer

{

private Canvas myCanvas;

private List<PainterCharge> bulletList;

private List<Painter> tankList;

private List<Charge> bullets;

private List<AbstractTank> tanks;

public DrawerGraphics(Canvas mainCanvas, List<AbstractTank> \_tanks,List<Charge> \_bullets)

{

mainCanvas.Children.Clear();

bulletList = new List<PainterCharge>();

tankList = new List<Painter>();

myCanvas = new Canvas();

myCanvas.Height = Settings.height;

myCanvas.Width = Settings.width;

myCanvas.Background = new SolidColorBrush(Colors.Black);

mainCanvas.Children.Add(myCanvas);

tanks = \_tanks;

bullets = \_bullets;

foreach (AbstractTank tank in tanks)

addTank(tank.Name);

foreach (Charge bullet in bullets)

addBullet();

}

override public void addBullet()

{

if(bulletList.Count<bullets.Count)

{

PainterCharge painter = new PainterCharge( myCanvas);

bulletList.Add(painter);

// canvas.Children.Add(painter.Image);

}

}

override public void addEagle()

{

Painter myPainter = new PainterEagle(myCanvas);

tankList.Add(myPainter);

}

override public void deleteBullet()

{

myCanvas.Children.Remove(bulletList[0].Image);

bulletList.RemoveAt(0);

}

override public void addTank(string tankName)

{

FactoryPainter myFactory = getFactory(tankName);

Painter myPainter = myFactory.getPainter(myCanvas);

tankList.Add(myPainter);

}

override public void deleteTank(int index)

{

myCanvas.Children.Remove(tankList[index].Image);

tankList.RemoveAt(index);

}

override public void drawing(List<AbstractTank> tankList, List<Charge> bulletList)

{

drawTanks(tankList);

drawBullets(bulletList);

}

public void drawTanks(List<AbstractTank> \_tankList)

{

for (int i = 0; i < tankList.Count; i++)

tankList[i].redrawing(\_tankList[i]);

}

public void drawBullets(List<Charge> \_bulletList)

{

for (int i = 0; i < \_bulletList.Count; i++)

bulletList[i].redrawing(\_bulletList[i]);

}

public FactoryPainter getFactory(string name)

{

FactoryPainter factory = null;

switch (name)

{

case Settings.nameTankBTR:

factory = new FactoryPainterBTR();

break;

case Settings.nameTankHeavy:

factory = new FactoryPainterHeavy();

break;

case Settings.nameTankLight:

factory = new FactoryPainterLight();

break;

case Settings.nameEagle:

factory = new FactoryPainterEagle();

break;

default:

break;

}

return factory;

}

}

}

**Пример:**

