Министерство науки и высшего образования Российской Федерации
Муромский институт (филиал)
Федерального государственного бюджетного образовательного учреждения
высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

Факультет ИТР

Кафедра ПИн

# *ЛАБОРАТОРНАЯ РАБОТА №9*

По Объектно-ориентированному программированию

Руководитель

Привезенцев Д.Г.
(фамилия, инициалы)

(подпись) (дата)

Студент ПИн - 121
(группа)

Ермилов М.В.
(фамилия, инициалы)

(подпись) (дата)

Муром 2022

## Лабораторная работа №9

**Тема:** Изучение и практическое применение поведенческого шаблона проектирования Состояние

Ход работы:

**Задание** :

Разработать иерархию классов с использованием шаблона Состояние согласно вариантам.

Пулемёт.

Атрибуты:

скорострельность, число патронов в магазине, вероятность осечки.

Операции:

Нажать курок, Отпустить курок, Перезарядить, Сменить ствол.

Состояния:

Готовность, Стрельба, Перегрев, Отсутствие патронов

Код по заданию:

```
    namespace lab9
{
    internal static class Program
    {
        /// <summary>
        ///  The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            // To customize application configuration such as set high DPI settings or
default font,
            // see https://aka.ms/applicationconfiguration.
            ApplicationConfiguration.Initialize();
            Application.Run(new FormMain());
        }
    }
}
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Forms;
using Timer = System.Windows.Forms.Timer;

namespace lab9
{
    public partial class FormMain : Form
    {
        private Machinegun machinegun;
        public FormMain()
        {
            InitializeComponent();
            machinegun = new Machinegun();
            machinegun.UpdateState += UpdateEvent;
            UpdateEvent();
            InitializeTimer();
        }

        private void ButtonPullTrigger_Click(object sender, EventArgs e) =>
machinegun.PullTrigger();
        private void ButtonReleaseTrigger_Click(object sender, EventArgs e) =>
machinegun.ReleaseTrigger();
        private void ButtonRecharge_Click(object sender, EventArgs e) =>
machinegun.Recharge();

        private void ButtonFix_Click(object sender, EventArgs e)
            => LabelLog.Text = $"Log: {machinegun.State.Fix()}";
        private void ButtonShoot_Click(object sender, EventArgs e)
            => LabelLog.Text = $"Log: {machinegun.State.Shoot()}";
        private void ButtonStopShoot_Click(object sender, EventArgs e)
            => LabelLog.Text = $"Log: {machinegun.State.StopShoot()}";

        private void InitializeTimer()
        {
            Timer timer = new Timer();
            timer.Interval = (int)((double)(1000 * machinegun.RateOfFire));
            timer.Tick += new EventHandler(Timer_Tick);
            timer.Enabled = true;
```

```
            timer.Start();
        }
        private void UpdateEvent() => LabelState.Text = $"State:
{machinegun.State.Name}";
        private void Timer_Tick(object Sender, EventArgs e)
        {
            machinegun.Shooting();
            LabelAmmo.Text = $"Ammo:
{machinegun.CountOfRoundsInStore}/{machinegun.MaxCountOfRoundsInStore}";
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace lab9
{
    public interface IState
    {
        public string Name { get; }
        public string Fix();
        public string Shoot();
        public string StopShoot();
    }
    public class StateOverheating : IState
    {
        public string Name => "Перегрев";
        private Machinegun machinegun;
        public StateOverheating(Machinegun machinegun) => this.machinegun = machinegun;
        public string Fix()
        {
            if (machinegun.isPullTrigger)
            {
                machinegun.ReleaseTrigger();
            }
            return "Ждите пока остынет";
        }
        public string Shoot() => machinegun.isPullTrigger ?
            "Вы и так стреляете, но лучше остановитесь" : "Ждите пока остынет";
        public string StopShoot()
        {
            if(machinegun.isPullTrigger)
            {
                machinegun.ReleaseTrigger();
                return "Стрельба прекращена, подождите пока оружие остынет";
            }
            return "Оружие и так не стреляет";
        }
    }
    public class StateOutOfAammoInStore : IState
    {
        public string Name => "Патроны в магазине закончились";
        private Machinegun machinegun;
        public StateOutOfAammoInStore(Machinegun machinegun) => this.machinegun =
machinegun;

        public string Fix()
        {
            if (machinegun.isPullTrigger)
            {
                machinegun.ReleaseTrigger();
            }
            machinegun.Recharge();
```

```
            machinegun.State = new StateReady(machinegun);
            machinegun.CallEvent();
            return "Вы можете возобновить стрельбу";
        }
        public string Shoot() => "В данный момент вы не можете стрелять";
        public string StopShoot()
        {
            if (machinegun.isPullTrigger)
            {
                machinegun.ReleaseTrigger();
                return "Вы отпустили курок";
            }
            return "В данный момент вы и так не стреляете";
        }

    }
    public class StateShooting : IState
    {
        public string Name => "Стрельба";
        private Machinegun machinegun;
        public StateShooting(Machinegun machinegun) => this.machinegun = machinegun;

        public string Fix() => "В данный момент всё работает";
        public string Shoot() => "Вы и так стреляете";
        public string StopShoot()
        {
            machinegun.ReleaseTrigger();
            return "Стрельба прекращена";
        }
    }
    public class StateReady : IState
    {
        public string Name => "Готово к использованию";
        private Machinegun machinegun;
        public StateReady(Machinegun machinegun) => this.machinegun = machinegun;
        public string Fix() => "В данный момент всё работает";
        public string Shoot()
        {
            machinegun.PullTrigger();
            return "Стрельба Начата";
        }
        public string StopShoot() => "Вы и так не стреляете";
    }
    public class StateJammed : IState
    {
        public string Name => "Готово к использованию";
        private Machinegun machinegun;
        public StateJammed(Machinegun machinegun) => this.machinegun = machinegun;
        public string Fix()
        {
            machinegun.ReleaseTrigger();
            machinegun.State = new StateReady(machinegun);
            machinegun.CallEvent();
            return "Вы можете возобновить стрельбу";
        }
        public string Shoot() => "Вы не можете стрелять";
        public string StopShoot() => Fix();
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace lab9
```

```
{
    public class Machinegun
    {
        private double _RateOfFire = 0.1;
        private double _Misfires = 0.1;
        private int _MaxCountOfRoundsInStore = 10;
        private int _CountOfRoundsInStore;
        private Random random;
        private bool _isPullTrigger = false;
        private double overheating = 0;
        private double _CriticalOverheating = 2;

        public IState State;

        /// <summary>
        /// Скорость стрельбы в секунду
        /// </summary>
        public double RateOfFire
        {
            get => _RateOfFire;
            set => _RateOfFire = value > 0 ? value : _RateOfFire;
        }
        /// <summary>
        /// Вероятность от 0 до 1, что случиться осечка и оружие заклинит
        /// </summary>
        public double Misfires
        {
            get => _Misfires = 0;
            set => _Misfires = 0 <= value && value <= 1 ? value : _Misfires;
        }
        /// <summary>
        /// Максимальное кол-во патрон в магазине
        /// </summary>
        public int MaxCountOfRoundsInStore
        {
            get => _MaxCountOfRoundsInStore;
            set => _MaxCountOfRoundsInStore = value > 0 ? value :
_MaxCountOfRoundsInStore;
        }
        /// <summary>
        /// Кол-во патрон в магазине
        /// </summary>
        public int CountOfRoundsInStore => _CountOfRoundsInStore;
        /// <summary>
        /// Критическая отметка перегрева
        /// </summary>
        public double CriticalOverheating
        {
            get => _CriticalOverheating;
            set => _CriticalOverheating = value > 0 ? value : _CriticalOverheating;
        }
        /// <summary>
        /// Стреляет ли оружие в данный момент
        /// </summary>
        public bool isPullTrigger => _isPullTrigger;


        public delegate void eventFunction();
        public event eventFunction? UpdateState;

        public Machinegun() : this(0) { }
        public Machinegun(int seed)
        {
            State = new StateReady(this);
            random = new Random(seed);
            _CountOfRoundsInStore = MaxCountOfRoundsInStore;
```

```
        }

        public void CallEvent() => UpdateState?.Invoke();

        public void Shooting()
        {
            if (CountOfRoundsInStore > 0 && State.GetType() == typeof(StateShooting))
            {
                if (random.NextDouble() <= Misfires && Misfires != 0)
                {
                    State = new StateJammed(this);
                    CallEvent();
                }
                else
                {
                    _CountOfRoundsInStore--;
                    if (CountOfRoundsInStore == 0)
                    {
                        State = new StateOutOfAammoInStore(this);
                        CallEvent();
                    }
                    else
                    {
                        overheating += random.NextDouble();
                        if (overheating > CriticalOverheating)
                        {
                            State = new StateOverheating(this);
                            CallEvent();
                        }
                    }
                }
            }
            else
            {
                if(overheating > 0)
                {
                    overheating -= random.NextDouble();
                    if(overheating < 0)
                    {
                        overheating = 0;
                        State = new StateReady(this);
                        CallEvent();
                    }
                }
            }
        }

        public void PullTrigger()
        {
            if(State.GetType() == typeof(StateReady) && CountOfRoundsInStore > 0)
            {
                _isPullTrigger = true;
                State = new StateShooting(this);
                CallEvent();
            }
        }
        public void ReleaseTrigger()
        {
            _isPullTrigger = false;
            if (State.GetType() == typeof(StateShooting))
            {
                State = new StateReady(this);
                CallEvent();
            }
        }
        public bool Recharge()
```

```
        {
            if(!isPullTrigger)
            {
                _CountOfRoundsInStore = MaxCountOfRoundsInStore;
                return true;
            }
            return false;
        }
    }
}
```

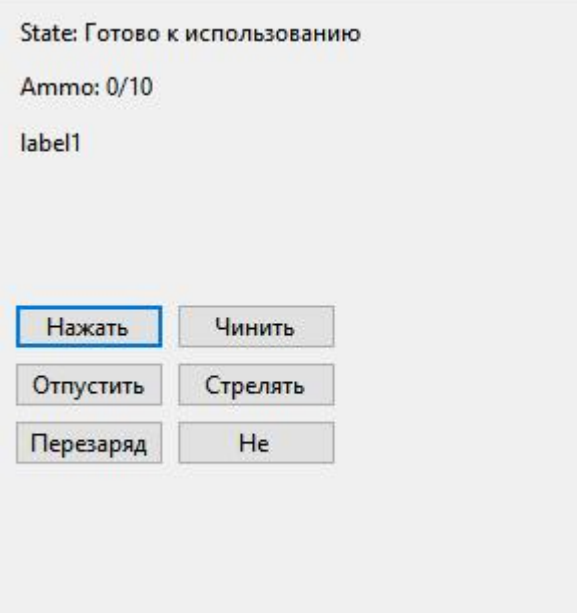


Рис 1 - пример работы программы