**ФГБОУ ВО**

**Национальный исследовательский университет**

**«МЭИ»**

**Лабораторная работа №8**

по курсу

«Технология программирования»

**ООП В C#**

**Выполнил:**

Балашов Савва

А-08-19

**Москва, 2020**

Содержание

[**Задание 8.1 3**](#_Toc58772425)

[**Задание 8.2 4**](#_Toc58772426)

[**1. Постановка задачи 4**](#_Toc58772427)

[**2. Разработка программы 4**](#_Toc58772428)

[**2.1. Разработка функций и методов классов 4**](#_Toc58772429)

[**2.2. Разработка пользовательского интерфейса 7**](#_Toc58772430)

[**3. Реализация и тестирование программы 7**](#_Toc58772431)

[**3.1. Описание разработанной программы 7**](#_Toc58772435)

[**3.2. Тестирование программы 23**](#_Toc58772436)

[**Приложение. Код программы 31**](#_Toc58772437)

# Задание 8.1

Часть I.

1. Построить иерархию классов в соответствии с вариантом задания.

2. Разработать методы и свойства для каждого из определяемых классов.

3. Реализовать программу на C# в соответствии с вариантом задания.

Вывести результаты

Часть II.

4. Изменить иерархию классов и реализовать ее на С#.

5. Показать на примере одного из методов, присутствующих в каждом классе, свойство

полиморфизма.

6. Расширить иерархию классов с использованием виртуального класса в качестве основы иерархии.

Показать пример использования полиморфизма методов.

Вывести результаты

Часть III.

7. Реализовать для иерархии (п.6) механизм интерфейсов, при этом один из классов должен

реализовывать как минимум два интерфейса. Использовать для проверки всех методов данного

класса многоадресный делегат.

Вывести результаты

Часть IV.

8. Реализовать обработку ошибок для п.7, при этом переопределив с помощью наследования одно из

событий:

- StackOverflowException

- ArrayTypeMismatchException

- DivideByZeroException

- IndexOutOfRangeException

- InvalidCastException

- OutOfMemoryException

- OverflowException

Вывести результаты

# **Задание 8.2**

С использованием методов объектно-ориентированного программирования разработать программу моделирования выбранной предметной области.

# Постановка задачи

Разработать программу в соответствии с заданиями 8.1, 8.2

**Входные данные:** температура и плотность звезды, либо её состояние

**Выходные данные:** состояние звезды, её изображение, синтезируемые элементы

**Функции программы:** принимать на вход температуру и плотность звезды, обрабатывать их и выдавать тип звезды, её изображение и элементы, которые она синтезирует

*Вид приложения – оконное приложение на языке C#*

*Среда разработки – JetBrains Rider*

# Разработка программы

## Разработка функций и методов классов

8.1 Создать абстрактный класс звезды Star, который будут наследовать остальные классы типов звезд. У абстрактного класс создать виртуальные и абстрактные методы, которые будут переопределяться в классах наследниках. У некоторых классов наследников сделать собственные методы, которые будут только у них. Добавить интерфейсы для наследования методов в нескольких классах и делегаты, для объединения методов в один. Диаграмма классов приведена на рисунке 2.1 Диаграмма. В таблицах Табл 1. Конструкторы и Табл.2 Поля и методы описаны конструкторы, поля и методы. В Табл.3 Интерфейсы представлены интерфейсы.

Строить модель буду на основе физических наблюдений, которые находятся в открытом доступе:

Этап – Индекс этапа = температура/плотность

|  |  |
| --- | --- |
| (горят при T, K)  гравитационное сжатие - 0  дейтерий - 1 = 10e6  водород - 2 = 10e7  гелий - 3 = 1.5\*10e8  углерод неон кислород - 4 = 8\*10e8, 1.2\*10e9, 1.5\*10e9  до кремния (вкл) - 5 = (2,7 to 3,5)⋅10e9  железо - 6 only density  взрыв сверхновой - 7 only density | (плотность для горения, kg/m3)  гравитационное сжатие - 0  дейтерий - 1 = 10e7  водород - 2 = 10e7  гелий - 3 = 5\*10e7  углерод неон кислород - 4 = 10e8, 4\*10e9, 10e10  до кремния (вкл) - 5 = 10e5 - 10e6  железо - 6 = 10e10 neutron/cm3  взрыв сверхновой - 7 only density |

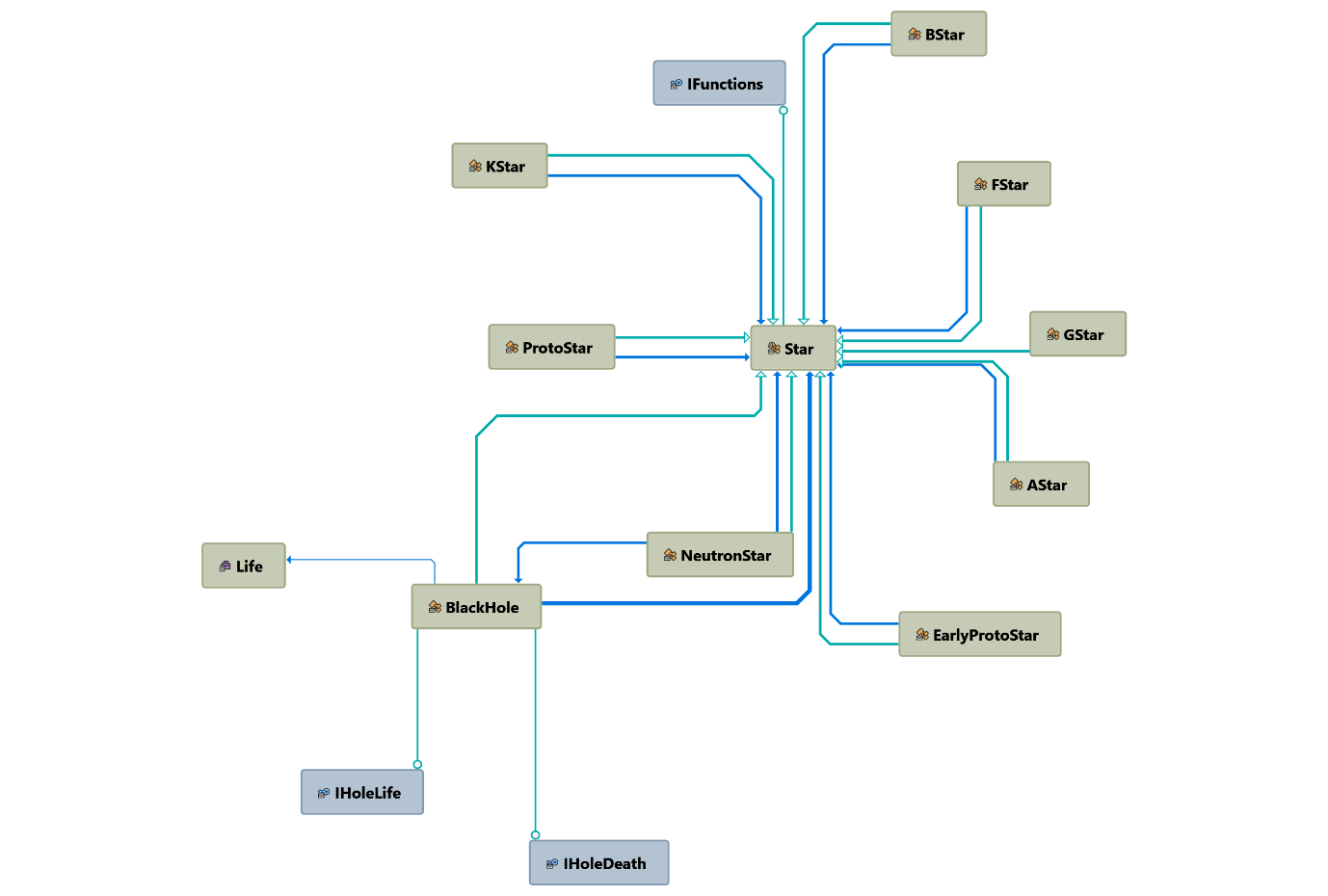


Рис.2.1 Диаграмма

Табл.1 Конструкторы

|  |  |  |
| --- | --- | --- |
| Название | Конструкторы | Описание |
| Star | protected Star(double start\_temperature, double start\_density); | Конструктор абстрактного класса звезды, принимает на вход параметры и обрабатывает их |
| EarlyProtoStar | public EarlyProtoStar(double temperature, double density) : base(temperature, density) | Конструктор класса наследника. Наследует принятые базовым классом параметры. |
| ProtoStar | public ProtoStar(double temperature, double density) : base(temperature, density) | Конструктор класса наследника. Наследует принятые базовым классом параметры. |
| AStar | public AStar(double temperature, double density) : base(temperature, density); | Конструктор класса наследника. Наследует принятые базовым классом параметры. |
| BStar | public BStar(double temperature, double density) : base(temperature, density); | Конструктор класса наследника. Наследует принятые базовым классом параметры. |
| FStar | public FStar(double temperature, double density) : base(temperature, density); | Конструктор класса наследника. Наследует принятые базовым классом параметры. |
| GStar | public GStar(double temperature, double density) : base(temperature, density); | Конструктор класса наследника. Наследует принятые базовым классом параметры. |
| KStar | public KStar(double temperature, double density) : base(temperature, density); | Конструктор класса наследника. Наследует принятые базовым классом параметры. |
| NeutronStar | public NeutronStar(double temperature, double density) : base(temperature, density); | Конструктор класса наследника. Наследует принятые базовым классом параметры. |
| BlackHole | public BlackHole(double temperature, double density) : base(temperature, density); | Конструктор класса наследника. Наследует принятые базовым классом параметры. |

Табл.2 Поля и методы

Продолжение Табл.1 Конструкторы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Поля | Методы | Входные данные | Выходные данные |
| Star | protected string state;  protected double temperature; protected double density;  protected string[] elements; | public abstract void Synthesize();  Наследованы от интерфейсов | Нет | нет |
| EarlyProtoStar | Наследуют поля и методы класса Star | | | |
| ProtoStar |
| AStar |
| BStar |
| FStar |
| GStar |
| KStar |
| NeutronStar | Наследованы от класса Star | public BlackHole TransformIntoBlackHole(); | Нет | Нет |
| BlackHole | Наследованы от класса Star | private void ShowMessage();  delegate void Life();  Наследованы от интерфейсов | Нет | Нет |

Табл.3 Интерфейсы

|  |  |
| --- | --- |
| Название | Методы |
| IFunctions | public string GetStats();  public string GetElements(); |
| IHoleLife | int delay();  int temperatureV();  int densityV(); |
| IHoleDeath | public void Consume();  public string GetTemp();  public string GetDens(); |

## Разработка пользовательского интерфейса

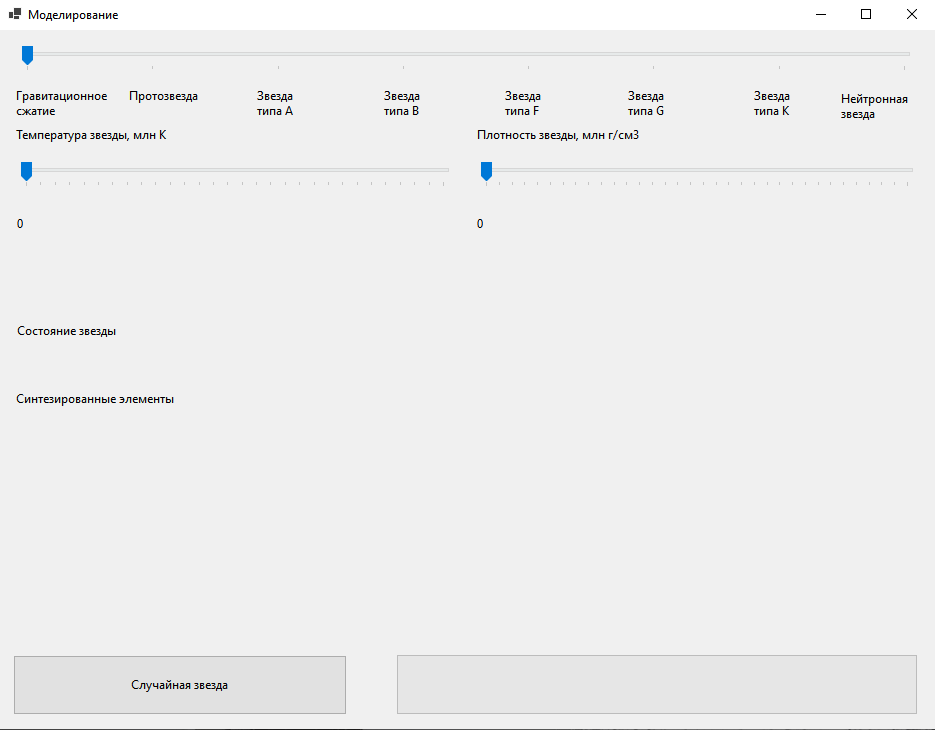
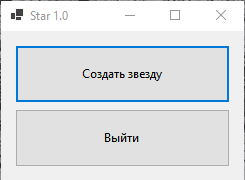
8.2 Создать оконную форму, в которой будет осуществляться ввод входных данных и вывод выходных данных с изображением и состоянием звезды. Для ввода будет использован слайдер TrackBar. Вначале будет показываться окно с вариантами: создать звезду или выйти из программы, при нажатии на первое, будет открываться новое окно с симуляцией.

Рис. 2.2 Интерфейс 8.2

Рис. 2.3 Интерфейс 8.2

# Реализация и тестирование программы



## Описание разработанной программы

Класс Star имеет один конструктор – параметризованный. В классе реализован один абстрактный метод Synthesize и наследовано от интерфейса IFunctions с переопределением два метода: GetStats и GetElements. GetStats возвращает тип звезды, а GetElements – синтезируемые ею элементы.

abstract class Star : IFunctions  
{  
  
 protected Star(double start\_temperature, double start\_density)  
 {  
 temperature = start\_temperature;  
 density = start\_density;  
 }  
  
 public string GetStats()  
 {  
 return state;  
 }  
  
 public string GetElements()  
 {  
 string elements\_un = null;  
 if (elements != null)  
 {  
 foreach (string element in elements)  
 {   
 elements\_un += element + " ";  
 }   
 elements\_un.Remove(elements\_un.Length - 2);   
 return elements\_un;  
  
 } else  
 {  
 return "Не синтезирует элементы";  
 }  
   
 }  
  
 public abstract void Synthesize();  
  
 protected string state;  
 protected double temperature;  
 protected double density;  
 protected string[] elements;  
}

Классы EarlyProtoStar, ProtoStar, AStar, BStar, FStar, GStar, KStar наследуют методы класса Star и переопределяют метод Synthesize, а также устанавливают тип звезды, меняя значение переменной state.

class EarlyProtoStar : Star  
{  
 public EarlyProtoStar(double temperature, double density) : base(temperature, density)  
 {  
 Synthesize();  
 state = "Гравитационное сжатие";  
 }  
  
 public override void Synthesize()  
 {  
 elements = null;  
 }  
}

Класс NeutronStar помимо наследуемых методов от Star имеет собственный метод public BlackHole TransformIntoBlackHole(), который позволяет пользователю создавать из нейтронной звезды черную дыру.

class NeutronStar : Star  
{  
 public NeutronStar(double temperature, double density) : base(temperature, density)  
 {  
 Synthesize();  
 state = "Нейтронная звезда";  
 }  
  
 public BlackHole TransformIntoBlackHole()  
 {  
 BlackHole blackHole = new BlackHole(temperature, density);  
 return blackHole;  
 }  
   
 public override void Synthesize()  
 {  
 elements = null;  
 }  
}

Класс BlackHole наследует методы класса Star, методы интерфейсов IHoleLife и IHoleDeath, обладает многоадресным делегатом Life, который объединяет методы интерфейсов. Также обладает методом ShowMessage для вывода сообщения о состоянии черной дыры после каждого изменения. Метод Consume интерфейса IHoleDeath с течением времени меняет температуру, плотность и синтезируемые элементы звезды, выводя сообщения после каждой перемены.

class BlackHole : Star, IHoleLife, IHoleDeath  
{  
 delegate void Life();  
 public BlackHole(double start\_temperature, double start\_density) : base(start\_temperature, start\_density)  
 {  
 try  
 {  
 state = "Черная дыра";  
 Life life = Synthesize;  
 life += Consume;  
 life();  
 }  
 catch (Exception e)  
 {  
 Console.WriteLine(e);  
 throw;  
 }  
 }  
  
 public int delay()  
 {  
 Random randomDelay = new Random();  
 int tmp\_delay = randomDelay.Next(1000, 3000);  
 return tmp\_delay;  
 }  
  
 public int temperatureV()  
 {  
 Random randomTemp = new Random();  
 return randomTemp.Next(1, 5);  
 }  
  
 public int densityV()  
 {  
 Random randomDens = new Random();  
 return randomDens.Next(10, 50);  
 }  
  
 private void ShowMessage()  
 {  
 MessageBox.Show("Состояние звезды: " + state + "\nЭлементы: " + GetElements() + "\nТемпература: " + GetTemp() + "\nПлотность " + GetDens());  
 }  
 public void Consume()  
 {  
 ShowMessage();  
 Thread.Sleep(delay());  
 elements = new[] {"Dark matter"};  
 temperature = temperatureV();  
 density = densityV();  
 ShowMessage();  
 Thread.Sleep(delay());  
 elements = null;  
 temperature = 0;  
 density = 0;  
 state = "Звезда прекратила свое существование";  
 ShowMessage();  
 }  
  
 public string GetTemp()  
 {  
 return temperature.ToString();  
 }  
  
 public string GetDens()  
 {  
 return density.ToString();  
 }  
  
 public override void Synthesize()  
 {  
 elements = null;  
 }  
  
}

Функция button1\_Click в Form1 открывает окно settingsForm при нажатии.

*// Create*private void button1\_Click(object sender, EventArgs e)  
{  
 SettingsForm settingsForm = new SettingsForm();  
 settingsForm.Show();  
}

Функция button2\_Click в SettingsForm закрывает программу при нажатии.  
*// Exit*private void button2\_Click(object sender, EventArgs e)  
{  
 Application.Exit();  
}

Функция TemperatureCheck ставит в соответствие температуре и плотности состояние звезды и возвращает индекс состояния.

public void TemperatureCheck(double temperature, double density, ref int state)  
{  
 if (temperature == 0 || density == 0)  
 {  
 state = 8;  
 }  
 else  
 {  
 if (temperature < 1 & density < 10)  
 {  
 state = 0;  
 }  
 else if (temperature < 10 & density <= 10)  
 {  
 state = 1;  
 }  
 else if (temperature < 1.5 \* 100 & density < 5 \* 10)  
 {  
 state = 2;  
 }  
 else if (temperature < 1.5 \* 1000 & density < 10000)  
 {  
 state = 3;  
 }  
 else if (temperature < 2.7 \* 1000 & density >= 1)  
 {  
 state = 4;  
 }  
 else if (temperature < 3.5 \* 1000 & density >= 1)  
 {  
 state = 5;  
 }  
 else if (temperature < 100000 & density < 10000)  
 {  
 state = 6;  
 }  
 else  
 {  
 state = 7;  
 }  
 }  
}

Функция Simulate передает в TemperatureCheck значения и принимает индекс состояния, в зависимости от индекса выдает информацию о звезде и изображение с помощью ShowMyImage.

private void Simulate()  
{  
 button2.Visible = false;  
 Double.TryParse(label11.Text, out double result\_temp);  
 Double.TryParse(label12.Text, out double result\_dens);  
 int star\_state = 0;  
 TemperatureCheck(result\_temp, result\_dens, ref star\_state);  
 if (star\_state == 8)  
 {  
 label13.Text = "Звезды не существует";  
 }  
 else  
 {  
 progressBar1.Value = (star\_state + 1) \* 10;  
 switch (star\_state)  
 {  
 case 0:  
 {  
   
 EarlyProtoStar earlyProtoStar = new EarlyProtoStar(result\_temp, result\_dens);  
 label13.Text = earlyProtoStar.GetStats() +"\nТемпература: " + result\_temp + "\nПлотность: " + result\_dens;  
 label14.Text = "Синтез: " + earlyProtoStar.GetElements();  
 ShowMyImage(  
 Properties.Resources.earlyprotostar1,  
 356, 242);  
 break;  
 }  
 case 1:  
 {  
 ProtoStar protoStar = new ProtoStar(result\_temp, result\_dens);  
 label13.Text = protoStar.GetStats() +"\nТемпература: " + result\_temp + "\nПлотность: " + result\_dens;  
 label14.Text = "Синтез: " + protoStar.GetElements();  
 ShowMyImage(  
 Properties.Resources.Protostar,  
 356, 242);  
 break;  
 }  
 case 2:  
 {  
 AStar aStar = new AStar(result\_temp, result\_dens);  
 label13.Text = aStar.GetStats() +"\nТемпература: " + result\_temp + "\nПлотность: " + result\_dens;  
 label14.Text = "Синтез: " + aStar.GetElements();  
 ShowMyImage(  
 Properties.Resources.AStar,  
 356, 242);  
 break;  
 }  
 case 3:  
 {  
 BStar bStar = new BStar(result\_temp, result\_dens);  
 label13.Text = bStar.GetStats() +"\nТемпература: " + result\_temp + "\nПлотность: " + result\_dens;  
 label14.Text = "Синтез: " + bStar.GetElements();  
 ShowMyImage(  
 Properties.Resources.BStar,  
 356, 242);  
 break;  
 }  
 case 4:  
 {  
 FStar fStar = new FStar(result\_temp, result\_dens);  
 label13.Text = fStar.GetStats() +"\nТемпература: " + result\_temp + "\nПлотность: " + result\_dens;  
 label14.Text = "Синтез: " + fStar.GetElements();  
 ShowMyImage(  
 Properties.Resources.FStar,  
 356, 242);  
 break;  
 }  
 case 5:  
 {  
 GStar gStar = new GStar(result\_temp, result\_dens);  
 label13.Text = gStar.GetStats() +"\nТемпература: " + result\_temp + "\nПлотность: " + result\_dens;  
 label14.Text = "Синтез: " + gStar.GetElements();  
 ShowMyImage(  
 Properties.Resources.GStar,  
 356, 242);  
 break;  
 }  
 case 6:  
 {  
 KStar kStar = new KStar(result\_temp, result\_dens);  
 label13.Text = kStar.GetStats() +"\nТемпература: " + result\_temp + "\nПлотность: " + result\_dens;  
 label14.Text = "Синтез: " + kStar.GetElements();  
 ShowMyImage(  
 Properties.Resources.KStar,  
 356, 242);  
 break;  
 }  
 case 7:  
 {  
 NeutronStar neutronStar = new NeutronStar(result\_temp, result\_dens);  
 label13.Text = neutronStar.GetStats() +"\nТемпература: " + result\_temp + "\nПлотность: " + result\_dens;  
 label14.Text = "Синтез: " + neutronStar.GetElements();  
 ShowMyImage(  
 Properties.Resources.NeutronStar,  
 356, 242);  
 button2.Visible = true;  
 break;  
 }  
 }  
 }  
}

Функция ShowMyImage находит указанное изображение и выводит его через PictureBox

private Bitmap MyImage ;  
public void ShowMyImage(Bitmap fileToDisplay, int xSize, int ySize)  
{  
 if (MyImage != null)  
 {  
 MyImage.Dispose();  
 }  
 pictureBox1.SizeMode = PictureBoxSizeMode.**StretchImage** ;  
 MyImage = fileToDisplay;  
 pictureBox1.ClientSize = new Size(xSize, ySize);  
 pictureBox1.Image = (Image) MyImage ;  
}

Функция button1\_Click генерирует случайные значения температуры и плотности в заданном диапазоне с помощью Generate\_Star и запускает Simulate для их обработки

private void button1\_Click(object sender, EventArgs e)  
{  
 double temp\_tmp = 0;  
 double dens\_tmp = 0;  
 Generate\_Star(ref temp\_tmp, ref dens\_tmp);  
 label11.Text = temp\_tmp.ToString();  
 label12.Text = dens\_tmp.ToString();  
 Simulate();  
}

Функция button2\_Click появляется только при наличии созданной нейтронной звезды и создает черную дыру при нажатии.

private void button2\_Click(object sender, EventArgs e)  
{  
 Double.TryParse(label11.Text, out double temp);  
 Double.TryParse(label12.Text, out double dens);  
 NeutronStar neutronStar = new NeutronStar(temp, dens);  
 BlackHole blackHole = neutronStar.TransformIntoBlackHole();  
 ShowMyImage(  
 Properties.Resources.BlackStar,  
 356, 242);  
 label11.Text = blackHole.GetTemp();  
 label12.Text = blackHole.GetDens();  
 label13.Text = blackHole.GetStats() +"\nТемпература: " + blackHole.GetTemp() + "\nПлотность: " + blackHole.GetDens();  
 label14.Text = "Синтез: " + blackHole.GetElements();  
 button2.Visible = false;  
}

Функция trackBar1\_Scroll отвечает за установку значений температуры и плотности в зависимости от положение указателя.

private void trackBar1\_Scroll(object sender, EventArgs e)  
{  
 switch (trackBar1.Value)  
 {  
 case 1:  
 {  
 label11.Text = "0.1";  
 label12.Text = "1";  
 Simulate();  
 break;  
 }  
 case 2:  
 {  
 label11.Text = "1";  
 label12.Text = "10";  
 Simulate();  
 break;  
 }  
 case 3:  
 {  
 label11.Text = "10";  
 label12.Text = "10";  
 Simulate();  
 break;  
 }  
 case 4:  
 {  
 label11.Text = "150";  
 label12.Text = "50";  
 Simulate();  
 break;  
 }  
 case 5:  
 {  
 label11.Text = "1500";  
 label12.Text = "10000";  
 Simulate();  
 break;  
 }  
 case 6:  
 {  
 label11.Text = "2700";  
 label12.Text = "1000";  
 Simulate();  
 break;  
 }  
 case 7:  
 {  
 label11.Text = "3500";  
 label12.Text = "1";  
 Simulate();  
 break;  
 }  
 case 8:  
 {  
 label11.Text = "100000";  
 label12.Text = "10000";  
 Simulate();  
 break;  
 }  
 }  
}

Функции trackBar2\_Scroll и trackBar3\_Scroll отвечают за установку температуры и плотности соответственно.

private void trackBar2\_Scroll(object sender, EventArgs e)  
{  
 switch (trackBar2.Value)  
 {  
 case 0:  
 {  
 label11.Text = "0.1";  
 Simulate();  
 break;  
 }  
 case 1:  
 {  
 label11.Text = "0.5";  
 Simulate();  
 break;  
 }  
 case 2:  
 {  
 label11.Text = "1";  
 Simulate();  
 break;  
 }  
 case 3:  
 {  
 label11.Text = "5";  
 Simulate();  
 break;  
 }  
 case 4:  
 {  
 label11.Text = "10";   
 Simulate();  
 break;  
 }  
 case 5:  
 {  
 label11.Text = "50";  
 Simulate();  
 break;  
 }  
 case 6:  
 {  
 label11.Text = "100";  
 Simulate();  
 break;  
 }  
 case 7:  
 {  
 label11.Text = "150";  
 Simulate();  
 break;  
 }  
 case 8:  
 {  
 label11.Text = "200";  
 Simulate();  
 break;  
 }  
 case 9:  
 {  
 label11.Text = "250";  
 Simulate();  
 break;  
 }  
 case 10:  
 {  
 label11.Text = "300";  
 Simulate();  
 break;  
 }  
 case 11:  
 {  
 label11.Text = "350";  
 Simulate();  
 break;  
 }  
 case 12:  
 {  
 label11.Text = "400";  
 Simulate();  
 break;  
 }  
 case 13:  
 {  
 label11.Text = "450";  
 Simulate();  
 break;  
 }  
 case 14:  
 {  
 label11.Text = "500";  
 Simulate();  
 break;  
 }  
 case 15:  
 {  
 label11.Text = "1000";  
 Simulate();  
 break;  
 }  
 case 16:  
 {  
 label11.Text = "1500";  
 Simulate();  
 break;  
 }  
 case 17:  
 {  
 label11.Text = "2000";  
 Simulate();  
 break;  
 }  
 case 18:  
 {  
 label11.Text = "2500";  
 Simulate();  
 break;  
 }  
 case 19:  
 {  
 label11.Text = "3000";  
 Simulate();  
 break;  
 }  
 case 20:  
 {  
 label11.Text = "3500";  
 Simulate();  
 break;  
 }  
 case 21:  
 {  
 label11.Text = "4000";  
 Simulate();  
 break;  
 }  
 case 22:  
 {  
 label11.Text = "4500";  
 Simulate();  
 break;  
 }  
 case 23:  
 {  
 label11.Text = "5000";  
 Simulate();  
 break;  
 }  
 case 24:  
 {  
 label11.Text = "10000";  
 Simulate();  
 break;  
 }  
 case 25:  
 {  
 label11.Text = "20000";  
 Simulate();  
 break;  
 }  
 case 26:  
 {  
 label11.Text = "30000";  
 Simulate();  
 break;  
 }  
 case 27:  
 {  
 label11.Text = "40000";  
 Simulate();  
 break;  
 }  
 case 28:  
 {  
 label11.Text = "50000";  
 Simulate();  
 break;  
 }  
 case 29:  
 {  
 label11.Text = "100000";  
 Simulate();  
 break;  
 }  
 }  
}  
private void trackBar3\_Scroll(object sender, EventArgs e)  
{  
 switch (trackBar3.Value)  
 {  
 case 0:  
 {  
 label12.Text = "0.1";  
 Simulate();  
 break;  
 }  
 case 1:  
 {  
 label12.Text = "0.5";  
 Simulate();  
 break;  
 }  
 case 2:  
 {  
 label12.Text = "1";  
 Simulate();  
 break;  
 }  
 case 3:  
 {  
 label12.Text = "5";  
 Simulate();  
 break;  
 }  
 case 4:  
 {  
 label12.Text = "10";  
 Simulate();  
 break;  
 }  
 case 5:  
 {  
 label12.Text = "50";  
 Simulate();  
 break;  
 }  
 case 6:  
 {  
 label12.Text = "100";  
 Simulate();  
 break;  
 }  
 case 7:  
 {  
 label12.Text = "150";  
 Simulate();  
 break;  
 }  
 case 8:  
 {  
 label12.Text = "200";  
 Simulate();  
 break;  
 }  
 case 9:  
 {  
 label12.Text = "250";  
 Simulate();  
 break;  
 }  
 case 10:  
 {  
 label12.Text = "300";  
 Simulate();  
 break;  
 }  
 case 11:  
 {  
 label12.Text = "350";  
 Simulate();  
 break;  
 }  
 case 12:  
 {  
 label12.Text = "400";  
 Simulate();  
 break;  
 }  
 case 13:  
 {  
 label12.Text = "450";  
 Simulate();  
 break;  
 }  
 case 14:  
 {  
 label12.Text = "500";  
 Simulate();  
 break;  
 }  
 case 15:  
 {  
 label12.Text = "1000";  
 Simulate();  
 break;  
 }  
 case 16:  
 {  
 label12.Text = "1500";  
 Simulate();  
 break;  
 }  
 case 17:  
 {  
 label12.Text = "2000";  
 Simulate();  
 break;  
 }  
 case 18:  
 {  
 label12.Text = "2500";  
 Simulate();  
 break;  
 }  
 case 19:  
 {  
 label12.Text = "3000";  
 Simulate();  
 break;  
 }  
 case 20:  
 {  
 label12.Text = "3500";  
 Simulate();  
 break;  
 }  
 case 21:  
 {  
 label12.Text = "4000";  
 Simulate();  
 break;  
 }  
 case 22:  
 {  
 label12.Text = "4500";  
 Simulate();  
 break;  
 }  
 case 23:  
 {  
 label12.Text = "5000";  
 Simulate();  
 break;  
 }  
 case 24:  
 {  
 label12.Text = "5500";  
 Simulate();  
 break;  
 }  
 case 25:  
 {  
 label12.Text = "6000";  
 Simulate();  
 break;  
 }  
 case 26:  
 {  
 label12.Text = "6500";  
 Simulate();  
 break;  
 }  
 case 27:  
 {  
 label12.Text = "7000";  
 Simulate();  
 break;  
 }  
 case 28:  
 {  
 label12.Text = "7500";  
 Simulate();  
 break;  
 }  
 case 29:  
 {  
 label12.Text = "8000";  
 Simulate();  
 break;  
 }  
 case 30:  
 {  
 label12.Text = "8500";  
 Simulate();  
 break;  
 }  
 case 31:  
 {  
 label12.Text = "9000";  
 Simulate();  
 break;  
 }  
 case 32:  
 {  
 label12.Text = "9500";  
 Simulate();  
 break;  
 }  
 case 33:  
 {  
 label12.Text = "10000";  
 Simulate();  
 break;  
 }  
 }  
}

Функция Generate\_Star возвращает случайные значения температуры и плотности, ограниченные заданным диапазоном.

private void Generate\_Star(ref double temp\_gen, ref double dens\_gen)  
 {  
 Random random\_value = new Random();  
 temp\_gen = random\_value.NextDouble() \* 100000;  
 dens\_gen = random\_value.NextDouble() \* 10000;  
   
 }

## Тестирование программы

Объект испытаний – программа smol\_PROJECT

Цель испытаний – проверить работу программы

Средства испытаний – локальный отладчик Windows

Метод испытаний – функциональное тестирование

Тесты представлены в Табл. 4 Тесты

Табл. 4 Тесты

|  |  |  |  |
| --- | --- | --- | --- |
| № теста | Входные данные | Выходные данные | Смысл теста |
| 1 | Температура 0.1  Плотность 0.1 |  | Минимальные значения |
| 2 | Температура 1  Плотность 0.1 |  | Проход по основным значениям температуры при постоянной плотности |
| 3 | Температура 10  Плотность 0.1 | Продолжение Табл. 4 Тесты |
| 4 | Температура 150  Плотность 0.1 |  |
| 5 | Температура 1500  Плотность 0.1 |  |
| 6 | Температура 100000  Плотность 0.1 | Продолжение Табл. 4 Тесты |
| 7 | Температура 0.1  Плотность 10 |  | Проход по основным значениям плотности при постоянной температуре |
| 8 | Температура 0.1  Плотность 50 |  |
| 9 | Температура 0.1  Плотность 10000 | Продолжение Табл. 4 Тесты |
| 10 | Температура 3000  Плотность 2000 |  | Средние значения плотности и температуры |
| 11 | Режим – протозвезда |  | Проход по всем режимам |
| 12 | Режим – звезда типа А | Продолжение Табл. 4 Тесты |
| 13 | Режим – звезда типа В |  |
| 14 | Режим – звезда типа F |  |
| 15 | Режим – звезда типа G | Продолжение Табл. 4 Тесты |
| 16 | Режим – звезда типа K |  |
| 17 | Режим – нейтронная звезда |  |
| 18 | Случайная звезда | Продолжение Табл. 4 Тесты | Случайные значения температуры и плотности |
| 19 | Случайная звезда |  |
| 20  Создать черную дыру | |  | Создание черной дыры |
|  |
|  |
|  |

Продолжение Табл. 4 Тесты

# Приложение. Код программы

Продолжение Табл. 4 Тесты

Form1.cs

using System.Windows.Forms;  
  
namespace smol\_PROJECT  
{  
 public partial class Form1 : Form  
 {  
 public Form1()  
 {  
 InitializeComponent();  
 }  
 *// Create* private void button1\_Click(object sender, EventArgs e)  
 {  
 SettingsForm settingsForm = new SettingsForm();  
 settingsForm.Show();  
 }  
 *// Exit* private void button2\_Click(object sender, EventArgs e)  
 {  
 Application.Exit();  
 }  
 }  
  
}

SettingForm.cs

using System;  
using System.Diagnostics;  
using System.Drawing;  
using System.Threading;  
using System.Windows.Forms;  
  
namespace smol\_PROJECT  
{  
 public partial class SettingsForm : Form  
 {  
 public void TemperatureCheck(double temperature, double density, ref int state)  
 {  
 if (temperature == 0 || density == 0)  
 {  
 state = 8;  
 }  
 else  
 {  
 if (temperature < 1 & density < 10)  
 {  
 state = 0;  
 }  
 else if (temperature < 10 & density <= 10)  
 {  
 state = 1;  
 }  
 else if (temperature < 1.5 \* 100 & density < 5 \* 10)  
 {  
 state = 2;  
 }  
 else if (temperature < 1.5 \* 1000 & density < 10000)  
 {  
 state = 3;  
 }  
 else if (temperature < 2.7 \* 1000 & density >= 1)  
 {  
 state = 4;  
 }  
 else if (temperature < 3.5 \* 1000 & density >= 1)  
 {  
 state = 5;  
 }  
 else if (temperature < 100000 & density < 10000)  
 {  
 state = 6;  
 }  
 else  
 {  
 state = 7;  
 }  
 }  
 }  
  
 public SettingsForm()  
 {  
 InitializeComponent();  
 label11.Text = "0";  
 label12.Text = "0";  
 }  
  
   
 private Bitmap MyImage ;  
 public void ShowMyImage(Bitmap fileToDisplay, int xSize, int ySize)  
 {  
 if (MyImage != null)  
 {  
 MyImage.Dispose();  
 }  
 pictureBox1.SizeMode = PictureBoxSizeMode.**StretchImage** ;  
 MyImage = fileToDisplay;  
 pictureBox1.ClientSize = new Size(xSize, ySize);  
 pictureBox1.Image = (Image) MyImage ;  
 }  
   
 private void Simulate()  
 {  
 button2.Visible = false;  
 Double.TryParse(label11.Text, out double result\_temp);  
 Double.TryParse(label12.Text, out double result\_dens);  
 int star\_state = 0;  
 TemperatureCheck(result\_temp, result\_dens, ref star\_state);  
 if (star\_state == 8)  
 {  
 label13.Text = "Звезды не существует";  
 }  
 else  
 {  
 progressBar1.Value = (star\_state + 1) \* 10;  
 switch (star\_state)  
 {  
 case 0:  
 {  
   
 EarlyProtoStar earlyProtoStar = new EarlyProtoStar(result\_temp, result\_dens);  
 label13.Text = earlyProtoStar.GetStats() +"\nТемпература: " + result\_temp + "\nПлотность: " + result\_dens;  
 label14.Text = "Синтез: " + earlyProtoStar.GetElements();  
 ShowMyImage(  
 Properties.Resources.earlyprotostar1,  
 356, 242);  
 break;  
 }  
 case 1:  
 {  
 ProtoStar protoStar = new ProtoStar(result\_temp, result\_dens);  
 label13.Text = protoStar.GetStats() +"\nТемпература: " + result\_temp + "\nПлотность: " + result\_dens;  
 label14.Text = "Синтез: " + protoStar.GetElements();  
 ShowMyImage(  
 Properties.Resources.Protostar,  
 356, 242);  
 break;  
 }  
 case 2:  
 {  
 AStar aStar = new AStar(result\_temp, result\_dens);  
 label13.Text = aStar.GetStats() +"\nТемпература: " + result\_temp + "\nПлотность: " + result\_dens;  
 label14.Text = "Синтез: " + aStar.GetElements();  
 ShowMyImage(  
 Properties.Resources.AStar,  
 356, 242);  
 break;  
 }  
 case 3:  
 {  
 BStar bStar = new BStar(result\_temp, result\_dens);  
 label13.Text = bStar.GetStats() +"\nТемпература: " + result\_temp + "\nПлотность: " + result\_dens;  
 label14.Text = "Синтез: " + bStar.GetElements();  
 ShowMyImage(  
 Properties.Resources.BStar,  
 356, 242);  
 break;  
 }  
 case 4:  
 {  
 FStar fStar = new FStar(result\_temp, result\_dens);  
 label13.Text = fStar.GetStats() +"\nТемпература: " + result\_temp + "\nПлотность: " + result\_dens;  
 label14.Text = "Синтез: " + fStar.GetElements();  
 ShowMyImage(  
 Properties.Resources.FStar,  
 356, 242);  
 break;  
 }  
 case 5:  
 {  
 GStar gStar = new GStar(result\_temp, result\_dens);  
 label13.Text = gStar.GetStats() +"\nТемпература: " + result\_temp + "\nПлотность: " + result\_dens;  
 label14.Text = "Синтез: " + gStar.GetElements();  
 ShowMyImage(  
 Properties.Resources.GStar,  
 356, 242);  
 break;  
 }  
 case 6:  
 {  
 KStar kStar = new KStar(result\_temp, result\_dens);  
 label13.Text = kStar.GetStats() +"\nТемпература: " + result\_temp + "\nПлотность: " + result\_dens;  
 label14.Text = "Синтез: " + kStar.GetElements();  
 ShowMyImage(  
 Properties.Resources.KStar,  
 356, 242);  
 break;  
 }  
 case 7:  
 {  
 NeutronStar neutronStar = new NeutronStar(result\_temp, result\_dens);  
 label13.Text = neutronStar.GetStats() +"\nТемпература: " + result\_temp + "\nПлотность: " + result\_dens;  
 label14.Text = "Синтез: " + neutronStar.GetElements();  
 ShowMyImage(  
 Properties.Resources.NeutronStar,  
 356, 242);  
 button2.Visible = true;  
 break;  
 }  
 }  
 }  
 }  
   
 private void button1\_Click(object sender, EventArgs e)  
 {  
 double temp\_tmp = 0;  
 double dens\_tmp = 0;  
 Generate\_Star(ref temp\_tmp, ref dens\_tmp);  
 label11.Text = temp\_tmp.ToString();  
 label12.Text = dens\_tmp.ToString();  
 Simulate();  
 }  
 private void button2\_Click(object sender, EventArgs e)  
 {  
 Double.TryParse(label11.Text, out double temp);  
 Double.TryParse(label12.Text, out double dens);  
 NeutronStar neutronStar = new NeutronStar(temp, dens);  
 BlackHole blackHole = neutronStar.TransformIntoBlackHole();  
 ShowMyImage(  
 Properties.Resources.BlackStar,  
 356, 242);  
 label11.Text = blackHole.GetTemp();  
 label12.Text = blackHole.GetDens();  
 label13.Text = blackHole.GetStats() +"\nТемпература: " + blackHole.GetTemp() + "\nПлотность: " + blackHole.GetDens();  
 label14.Text = "Синтез: " + blackHole.GetElements();  
 button2.Visible = false;  
 }  
   
 private void trackBar1\_Scroll(object sender, EventArgs e)  
 {  
 switch (trackBar1.Value)  
 {  
 case 1:  
 {  
 label11.Text = "0.1";  
 label12.Text = "1";  
 Simulate();  
 break;  
 }  
 case 2:  
 {  
 label11.Text = "1";  
 label12.Text = "10";  
 Simulate();  
 break;  
 }  
 case 3:  
 {  
 label11.Text = "10";  
 label12.Text = "10";  
 Simulate();  
 break;  
 }  
 case 4:  
 {  
 label11.Text = "150";  
 label12.Text = "50";  
 Simulate();  
 break;  
 }  
 case 5:  
 {  
 label11.Text = "1500";  
 label12.Text = "10000";  
 Simulate();  
 break;  
 }  
 case 6:  
 {  
 label11.Text = "2700";  
 label12.Text = "1000";  
 Simulate();  
 break;  
 }  
 case 7:  
 {  
 label11.Text = "3500";  
 label12.Text = "1";  
 Simulate();  
 break;  
 }  
 case 8:  
 {  
 label11.Text = "100000";  
 label12.Text = "10000";  
 Simulate();  
 break;  
 }  
 }  
 }  
 private void trackBar2\_Scroll(object sender, EventArgs e)  
 {  
 switch (trackBar2.Value)  
 {  
 case 0:  
 {  
 label11.Text = "0.1";  
 Simulate();  
 break;  
 }  
 case 1:  
 {  
 label11.Text = "0.5";  
 Simulate();  
 break;  
 }  
 case 2:  
 {  
 label11.Text = "1";  
 Simulate();  
 break;  
 }  
 case 3:  
 {  
 label11.Text = "5";  
 Simulate();  
 break;  
 }  
 case 4:  
 {  
 label11.Text = "10";   
 Simulate();  
 break;  
 }  
 case 5:  
 {  
 label11.Text = "50";  
 Simulate();  
 break;  
 }  
 case 6:  
 {  
 label11.Text = "100";  
 Simulate();  
 break;  
 }  
 case 7:  
 {  
 label11.Text = "150";  
 Simulate();  
 break;  
 }  
 case 8:  
 {  
 label11.Text = "200";  
 Simulate();  
 break;  
 }  
 case 9:  
 {  
 label11.Text = "250";  
 Simulate();  
 break;  
 }  
 case 10:  
 {  
 label11.Text = "300";  
 Simulate();  
 break;  
 }  
 case 11:  
 {  
 label11.Text = "350";  
 Simulate();  
 break;  
 }  
 case 12:  
 {  
 label11.Text = "400";  
 Simulate();  
 break;  
 }  
 case 13:  
 {  
 label11.Text = "450";  
 Simulate();  
 break;  
 }  
 case 14:  
 {  
 label11.Text = "500";  
 Simulate();  
 break;  
 }  
 case 15:  
 {  
 label11.Text = "1000";  
 Simulate();  
 break;  
 }  
 case 16:  
 {  
 label11.Text = "1500";  
 Simulate();  
 break;  
 }  
 case 17:  
 {  
 label11.Text = "2000";  
 Simulate();  
 break;  
 }  
 case 18:  
 {  
 label11.Text = "2500";  
 Simulate();  
 break;  
 }  
 case 19:  
 {  
 label11.Text = "3000";  
 Simulate();  
 break;  
 }  
 case 20:  
 {  
 label11.Text = "3500";  
 Simulate();  
 break;  
 }  
 case 21:  
 {  
 label11.Text = "4000";  
 Simulate();  
 break;  
 }  
 case 22:  
 {  
 label11.Text = "4500";  
 Simulate();  
 break;  
 }  
 case 23:  
 {  
 label11.Text = "5000";  
 Simulate();  
 break;  
 }  
 case 24:  
 {  
 label11.Text = "10000";  
 Simulate();  
 break;  
 }  
 case 25:  
 {  
 label11.Text = "20000";  
 Simulate();  
 break;  
 }  
 case 26:  
 {  
 label11.Text = "30000";  
 Simulate();  
 break;  
 }  
 case 27:  
 {  
 label11.Text = "40000";  
 Simulate();  
 break;  
 }  
 case 28:  
 {  
 label11.Text = "50000";  
 Simulate();  
 break;  
 }  
 case 29:  
 {  
 label11.Text = "100000";  
 Simulate();  
 break;  
 }  
 }  
 }  
 private void trackBar3\_Scroll(object sender, EventArgs e)  
 {  
 switch (trackBar3.Value)  
 {  
 case 0:  
 {  
 label12.Text = "0.1";  
 Simulate();  
 break;  
 }  
 case 1:  
 {  
 label12.Text = "0.5";  
 Simulate();  
 break;  
 }  
 case 2:  
 {  
 label12.Text = "1";  
 Simulate();  
 break;  
 }  
 case 3:  
 {  
 label12.Text = "5";  
 Simulate();  
 break;  
 }  
 case 4:  
 {  
 label12.Text = "10";  
 Simulate();  
 break;  
 }  
 case 5:  
 {  
 label12.Text = "50";  
 Simulate();  
 break;  
 }  
 case 6:  
 {  
 label12.Text = "100";  
 Simulate();  
 break;  
 }  
 case 7:  
 {  
 label12.Text = "150";  
 Simulate();  
 break;  
 }  
 case 8:  
 {  
 label12.Text = "200";  
 Simulate();  
 break;  
 }  
 case 9:  
 {  
 label12.Text = "250";  
 Simulate();  
 break;  
 }  
 case 10:  
 {  
 label12.Text = "300";  
 Simulate();  
 break;  
 }  
 case 11:  
 {  
 label12.Text = "350";  
 Simulate();  
 break;  
 }  
 case 12:  
 {  
 label12.Text = "400";  
 Simulate();  
 break;  
 }  
 case 13:  
 {  
 label12.Text = "450";  
 Simulate();  
 break;  
 }  
 case 14:  
 {  
 label12.Text = "500";  
 Simulate();  
 break;  
 }  
 case 15:  
 {  
 label12.Text = "1000";  
 Simulate();  
 break;  
 }  
 case 16:  
 {  
 label12.Text = "1500";  
 Simulate();  
 break;  
 }  
 case 17:  
 {  
 label12.Text = "2000";  
 Simulate();  
 break;  
 }  
 case 18:  
 {  
 label12.Text = "2500";  
 Simulate();  
 break;  
 }  
 case 19:  
 {  
 label12.Text = "3000";  
 Simulate();  
 break;  
 }  
 case 20:  
 {  
 label12.Text = "3500";  
 Simulate();  
 break;  
 }  
 case 21:  
 {  
 label12.Text = "4000";  
 Simulate();  
 break;  
 }  
 case 22:  
 {  
 label12.Text = "4500";  
 Simulate();  
 break;  
 }  
 case 23:  
 {  
 label12.Text = "5000";  
 Simulate();  
 break;  
 }  
 case 24:  
 {  
 label12.Text = "5500";  
 Simulate();  
 break;  
 }  
 case 25:  
 {  
 label12.Text = "6000";  
 Simulate();  
 break;  
 }  
 case 26:  
 {  
 label12.Text = "6500";  
 Simulate();  
 break;  
 }  
 case 27:  
 {  
 label12.Text = "7000";  
 Simulate();  
 break;  
 }  
 case 28:  
 {  
 label12.Text = "7500";  
 Simulate();  
 break;  
 }  
 case 29:  
 {  
 label12.Text = "8000";  
 Simulate();  
 break;  
 }  
 case 30:  
 {  
 label12.Text = "8500";  
 Simulate();  
 break;  
 }  
 case 31:  
 {  
 label12.Text = "9000";  
 Simulate();  
 break;  
 }  
 case 32:  
 {  
 label12.Text = "9500";  
 Simulate();  
 break;  
 }  
 case 33:  
 {  
 label12.Text = "10000";  
 Simulate();  
 break;  
 }  
 }  
 }  
  
 private void Generate\_Star(ref double temp\_gen, ref double dens\_gen)  
 {  
 Random random\_value = new Random();  
 temp\_gen = random\_value.NextDouble() \* 100000;  
 dens\_gen = random\_value.NextDouble() \* 10000;  
   
 }  
 }  
}

Star.cs

using System;  
using System.Diagnostics;  
using System.Numerics;  
using System.Threading;  
using System.Windows.Forms;  
  
namespace smol\_PROJECT  
{  
  
 interface IFunctions  
 {  
 public string GetStats();  
 public string GetElements();  
 }  
  
 abstract class Star : IFunctions  
 {  
  
 protected Star(double start\_temperature, double start\_density)  
 {  
 temperature = start\_temperature;  
 density = start\_density;  
 }  
  
 public string GetStats()  
 {  
 return state;  
 }  
  
 public string GetElements()  
 {  
 string elements\_un = null;  
 if (elements != null)  
 {  
 foreach (string element in elements)  
 {   
 elements\_un += element + " ";  
 }   
 elements\_un.Remove(elements\_un.Length - 2);   
 return elements\_un;  
  
 } else  
 {  
 return "Не синтезирует элементы";  
 }  
   
 }  
  
 public abstract void Synthesize();  
  
 protected string state;  
 protected double temperature;  
 protected double density;  
 protected string[] elements;  
 }  
  
 class EarlyProtoStar : Star  
 {  
 public EarlyProtoStar(double temperature, double density) : base(temperature, density)  
 {  
 Synthesize();  
 state = "Гравитационное сжатие";  
 }  
  
 public override void Synthesize()  
 {  
 elements = null;  
 }  
 }  
  
 class ProtoStar : Star  
 {  
 public ProtoStar(double temperature, double density) : base(temperature, density)  
 {  
 Synthesize();  
 state = "Протозвезда";  
 }  
   
 public override void Synthesize()  
 {  
 elements = new[] {"Deuterium"};  
 }  
 }  
  
 class AStar : Star  
 {  
 public AStar(double temperature, double density) : base(temperature, density)  
 {  
 Synthesize();  
 state = "Звезда типа А";  
 }  
  
 public override void Synthesize()  
 {  
 elements = new[] {"Hydrogenium"};  
 }  
 }  
  
 class BStar : Star  
 {  
 public BStar(double temperature, double density) : base(temperature, density)  
 {  
 Synthesize();  
 state = "Звезда типа В";  
 }  
  
 public override void Synthesize()  
 {  
 elements = new[] {"Helium"};  
 }  
 }  
  
 class FStar : Star  
 {  
 public FStar(double temperature, double density) : base(temperature, density)  
 {  
 Synthesize();  
 state = "Звезда типа F";  
 }  
  
  
 public override void Synthesize()  
 {  
 elements = new[] {"Calcium", "Oxygenium", "Neon"};  
 }  
 }  
  
 class GStar : Star  
 {  
 public GStar(double temperature, double density) : base(temperature, density)  
 {  
 Synthesize();  
 state = "Звезда типа G";  
 }  
  
 public override void Synthesize()  
 {  
 elements = new[] {"Magnesium", "Silicium", "Sulfur", "Argon", "Calcium"};  
 }  
 }  
  
 class KStar : Star  
 {  
 public KStar(double temperature, double density) : base(temperature, density)  
 {  
 Synthesize();  
 state = "Звезда типа K";  
 }  
  
 public override void Synthesize()  
 {  
 elements = new[] {"Ferrum"};  
 }  
 }  
  
 class NeutronStar : Star  
 {  
 public NeutronStar(double temperature, double density) : base(temperature, density)  
 {  
 Synthesize();  
 state = "Нейтронная звезда";  
 }  
  
 public BlackHole TransformIntoBlackHole()  
 {  
 BlackHole blackHole = new BlackHole(temperature, density);  
 return blackHole;  
 }  
   
 public override void Synthesize()  
 {  
 elements = null;  
 }  
 }  
  
 interface IHoleLife  
 {  
 int delay();  
 int temperatureV();  
 int densityV();  
 }  
  
 interface IHoleDeath  
 {  
 public void Consume();  
 public string GetTemp();  
 public string GetDens();  
 }  
   
 class BlackHole : Star, IHoleLife, IHoleDeath  
 {  
 delegate void Life();  
 public BlackHole(double start\_temperature, double start\_density) : base(start\_temperature, start\_density)  
 {  
 try  
 {  
 state = "Черная дыра";  
 Life life = Synthesize;  
 life += Consume;  
 life();  
 }  
 catch (Exception e)  
 {  
 Console.WriteLine(e);  
 throw;  
 }  
 }  
  
 public int delay()  
 {  
 Random randomDelay = new Random();  
 int tmp\_delay = randomDelay.Next(1000, 3000);  
 return tmp\_delay;  
 }  
  
 public int temperatureV()  
 {  
 Random randomTemp = new Random();  
 return randomTemp.Next(1, 5);  
 }  
  
 public int densityV()  
 {  
 Random randomDens = new Random();  
 return randomDens.Next(10, 50);  
 }  
  
 private void ShowMessage()  
 {  
 MessageBox.Show("Состояние звезды: " + state + "\nЭлементы: " + GetElements() + "\nТемпература: " + GetTemp() + "\nПлотность " + GetDens());  
 }  
 public void Consume()  
 {  
 ShowMessage();  
 Thread.Sleep(delay());  
 elements = new[] {"Dark matter"};  
 temperature = temperatureV();  
 density = densityV();  
 ShowMessage();  
 Thread.Sleep(delay());  
 elements = null;  
 temperature = 0;  
 density = 0;  
 state = "Звезда прекратила свое существование";  
 ShowMessage();  
 }  
  
 public string GetTemp()  
 {  
 return temperature.ToString();  
 }  
  
 public string GetDens()  
 {  
 return density.ToString();  
 }  
  
 public override void Synthesize()  
 {  
 elements = null;  
 }  
  
 }  
}