**Типы**

**Кортежи**

var tuple = (5, 10); (int, int) tuple = (5, 10);

**класс**

**структуры**

struct Person

{

public string name;

public int age;

public void Print()

{

Console.WriteLine($"Имя: {name} Возраст: {age}");

}

}

объект.поле\_структуры

объект.метод\_структуры(параметры\_метода)

**конструкторы свойства**

class Person

{

public string name;

public int age;

//конструктор

public Person()

{

Console.WriteLine("Создание объекта Person");

name = "Tom";

age = 37;

}

public void Print()

{

Console.WriteLine($"Имя: {name} Возраст: {age}");

}

//деконструктор

~Person(){

}

}

**индексаторы**

class Person

{

public string Name { get;}

public Person(string name) => Name=name;

}

class Company

{

Person[] personal;

public Company(Person[] people) => personal = people;

// индексатор

public Person this[int index]

{

get => personal[index];

set => personal[index] = value;

}

}

class Company

{

Person[] personal;

public Company(Person[] people) => personal = people;

// индексатор

public Person this[int index]

{

get

{

// если индекс имеется в массиве

if (index >= 0 && index < personal.Length)

return personal[index]; // то возвращаем объект Person по индексу

else

throw new ArgumentOutOfRangeException(); // иначе генерируем исключение

}

set

{

// если индекс есть в массиве

if (index >= 0 && index < personal.Length)

personal[index] = value; // переустанавливаем значение по индексу

}

}

}

**константы,**

**доступность,**

**переопределения**

class Person

{

public string Name { get; set; }

public Person(string name)

{

Name = name;

}

public virtual void Print()

{

Console.WriteLine(Name);

}

}

class Employee : Person

{

public string Company { get; set; }

public Employee(string name, string company)

: base(name)

{

Company = company;

}

public override void Print()

{

Console.WriteLine($"{Name} работает в {Company}");

}

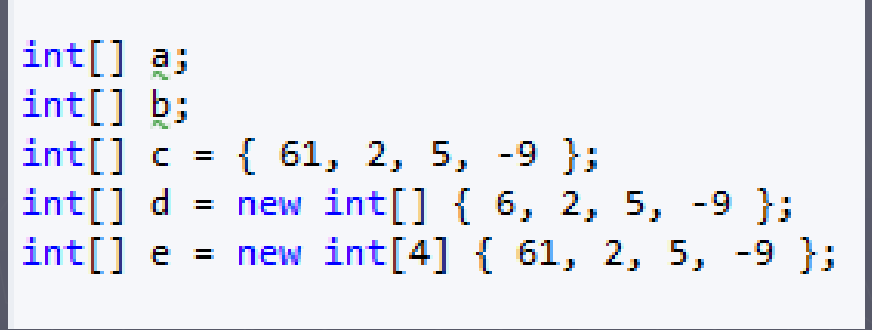
}

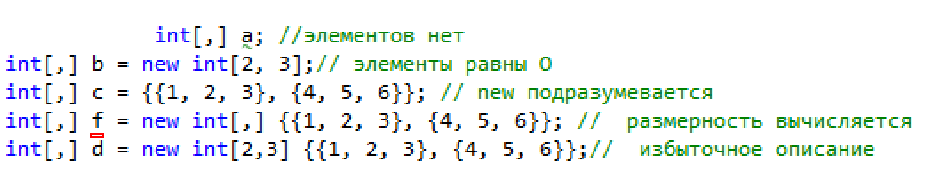
**Преобразования**

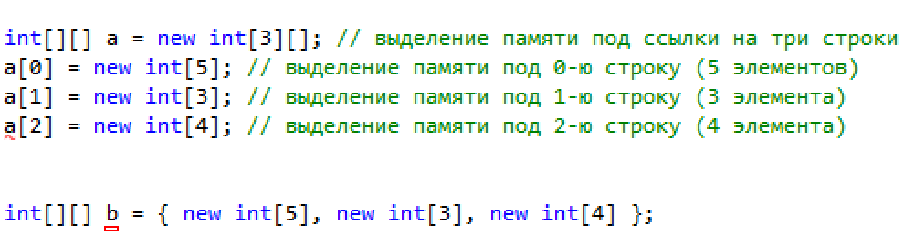
ushort a = 4;

byte b = (byte) a;

**массивы,**







**Наследование**

class Person

{

public string Name { get; set;}

public Person(string name)

{

Name = name;

}

public void Print()

{

Console.WriteLine(Name);

}

}

class Employee : Person

{

public string Company { get; set; }

public Employee(string name, string company)

: base(name)

{

Company = company;

}

}

**Интерфейс**

**станд. интерфейсы,**

**статика**

**виртуальность**

class Person

{

public string Name { get; set; }

public Person(string name)

{

Name = name;

}

public virtual void Print()

{

Console.WriteLine(Name);

}

}

class Employee : Person

{

public string Company { get; set; }

public Employee(string name, string company) : base(name)

{

Company = company;

}

}

**Коллекции**

**Список List<T>**

List<string> people = new List<string> () { "Tom" };

people.Add("Bob"); // добавление элемента

// people = { "Tom", "Bob" };

people.AddRange(new[] { "Sam", "Alice" }); // добавляем массив

// people = { "Tom", "Bob", "Sam", "Alice" };

// также можно было бы добавить другой список

// people.AddRange(new List<string>(){ "Sam", "Alice" });

people.Insert(0, "Eugene"); // вставляем на первое место

// people = { "Eugene", "Tom", "Bob", "Sam", "Alice" };

people.InsertRange(1, new string[] {"Mike", "Kate"}); // вставляем массив с индекса 1

// people = { "Eugene", "Mike", "Kate", "Tom", "Bob", "Sam", "Alice" };

// также можно было бы добавить другой список

// people.InsertRange(1, new List<string>(){ "Mike", "Kate" });

people.RemoveAt(1); // удаляем второй элемент

// people = { "Eugene", "Kate", "Tom", "Bob", "Sam", "Tom", "Alice" };

people.Remove("Tom"); // удаляем элемент "Tom"

// people = { "Eugene", "Kate", "Bob", "Sam", "Tom", "Alice" };

// удаляем из списка все элементы, длина строки которых равна 3

people.RemoveAll(person => person.Length == 3);

// people = { "Eugene", "Kate", "Alice" };

// удаляем из списка 2 элемента начиная с индекса 1

people.RemoveRange(1, 2);

// people = { "Eugene"};

// полностью очищаем список

people.Clear();

// people = { };

**Делегаты**

Message mes; // 2. Создаем переменную делегата

mes = Hello; // 3. Присваиваем этой переменной адрес метода

mes(); // 4. Вызываем метод

void Hello() => Console.WriteLine("Hello METANIT.COM");

delegate void Message(); // 1. Объявляем делегат

**стандартные делегаты**

public delegate void Action()

public delegate void Action<in T>(T obj)

DoOperation(10, 6, Add); // 10 + 6 = 16

DoOperation(10, 6, Multiply); // 10 \* 6 = 60

void DoOperation(int a, int b, Action<int, int> op) => op(a, b);

void Add(int x, int y) => Console.WriteLine($"{x} + {y} = {x + y}");

void Multiply(int x, int y) => Console.WriteLine($"{x} \* {y} = {x \* y}");

delegate bool Predicate<in T>(T obj);

Predicate<int> isPositive = (int x) => x > 0;

Console.WriteLine(isPositive(20));

Console.WriteLine(isPositive(-20));

TResult Func<out TResult>()

TResult Func<in T, out TResult>(T arg)

TResult Func<in T1, in T2, out TResult>(T1 arg1, T2 arg2)

int result1 = DoOperation(6, DoubleNumber); // 12

Console.WriteLine(result1);

int result2 = DoOperation(6, SquareNumber); // 36

Console.WriteLine(result2);

int DoOperation(int n, Func<int, int> operation) => operation(n);

int DoubleNumber(int n) => 2 \* n;

int SquareNumber(int n) => n \* n;

**события**

using System;

namespace ConsoleApplication1

{

delegate void UI ();

class MyEvent

{

// Объявляем событие

public event UI UserEvent;

// Используем метод для запуска события

public void OnUserEvent()

{

UserEvent();

}

}

class UserInfo

{

string uiName, uiFamily;

int uiAge;

public UserInfo(string Name, string Family, int Age)

{

this.Name = Name;

this.Family = Family;

this.Age = Age;

}

public string Name { set { uiName = value; } get { return uiName; } }

public string Family { set { uiFamily = value; } get { return uiFamily; } }

public int Age { set { uiAge = value; } get { return uiAge; } }

// Обработчик события

public void UserInfoHandler()

{

Console.WriteLine("Событие вызвано!\n");

Console.WriteLine("Имя: {0}\nФамилия: {1}\nВозраст: {2}",Name,Family,Age);

}

}

class Program

{

static void Main()

{

MyEvent evt = new MyEvent();

UserInfo user1 = new UserInfo(Name: "Alex", Family: "Erohin", Age: 26);

// Добавляем обработчик события

evt.UserEvent += user1.UserInfoHandler;

// Запустим событие

evt.OnUserEvent();

Console.ReadLine();

}

}

}

**Обобщения**

SendMessage(new Message("Hello World"));

SendMessage(new EmailMessage("Bye World"));

void SendMessage<T>(T message) where T: Message

{

Console.WriteLine($"Отправляется сообщение: {message.Text}");

}

class Message

{

public string Text { get; } // текст сообщения

public Message(string text)

{

Text = text;

}

}

class EmailMessage : Message

{

public EmailMessage(string text) : base(text) { }

}

**Исключения**

try

{

}

Catch или Catch (тип\_исключения) или Catch (тип\_исключения имя\_переменной)или Catch(тип\_исключения имя\_переменной) when (условие)

{

}

finally

{

}

**методы расширения**

string s = "Привет мир";

char c = 'и';

int i = s.CharCount(c);

Console.WriteLine(i);

public static class StringExtension

{

public static int CharCount(this string str, char c)

{

int counter = 0;

for (int i = 0; i < str.Length; i++)

{

if (str[i] == c)

counter++;

}

return counter;

}

}

**перегрузка операций**

class Counter

{

public int Value { get; set; }

public static Counter operator +(Counter counter1, Counter counter2)

{

return new Counter { Value = counter1.Value + counter2.Value };

}

public static bool operator >(Counter counter1, Counter counter2)

{

return counter1.Value > counter2.Value;

}

public static bool operator <(Counter counter1, Counter counter2)

{

return counter1.Value < counter2.Value;

}

}

public static Counter operator ++(Counter counter1)

{

counter1.Value += 10;

return counter1;

}

public static Counter operator --(Counter counter1)

{

counter1.Value -= 10;

return counter1;

}

public static bool operator !(Counter counter1)

{

return counter1.Value == 0;

}

public static bool operator true(Counter counter1)

{

return counter1.Value != 0;

}

public static bool operator false(Counter counter1)

{

return counter1.Value == 0;

}

**анонимные функции**

using System;

using System.Collections.Generic;

namespace ConsoleApplication1

{

delegate int Sum(int number);

class Program

{

static Sum SomeVar()

{

int result = 0;

// Вызов анонимного метода

Sum del = delegate (int number)

{

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

result += i;

return result;

};

return del;

}

static void Main()

{

Sum del1 = SomeVar();

for (int i = 1; i <= 5; i++)

{

Console.WriteLine("Cумма {0} равна: {1}",i,del1(i));

}

Console.ReadLine();

}

}

}

**лямбда выражения**

Operation sum = (x, y) => Console.WriteLine($"{x} + {y} = {x + y}");

sum(1, 2); // 1 + 2 = 3

sum(22, 14); // 22 + 14 = 36

delegate void Operation(int x, int y);

**LINQ**

string[] people = { "Tom", "Bob", "Sam", "Tim", "Tomas", "Bill" };

// создаем новый список для результатов

var selectedPeople = from p in people // передаем каждый элемент из people в переменную p

where p.ToUpper().StartsWith("T") //фильтрация по критерию

orderby p // упорядочиваем по возрастанию

select p; // выбираем объект в создаваемую коллекцию

foreach (string person in selectedPeople)

Console.WriteLine(person);

**Select**

var people = new List<Person>

{

new Person ("Tom", 23),

new Person ("Bob", 27),

new Person ("Sam", 29),

new Person ("Alice", 24)

};

var names = from p in people select p.Name;

foreach (string n in names)

Console.WriteLine(n);

**WHERE**

string[] people = { "Tom", "Alice", "Bob", "Sam", "Tim", "Tomas", "Bill" };

var selectedPeople = from p in people

where p.Length == 3

select p;

**OrderBy**

int[] numbers = { 3, 12, 4, 10};

var orderedNumbers = from i in numbers

orderby i

select i;

**String**

string s3 = "world";

string s4 = string.Concat(s3, "!!!");// соединение

int result = string.Compare(s3, s4); // сравнение

char ch = 'o';

int indexOfChar = s1.IndexOf(ch);// индекс первого вхождения

string text = "И поэтому все так произошло";

string[] words = text.Split(new char[] { ' ' });

**потоковые классы**

**FileStream**

// запись в файл

using (FileStream fstream = new FileStream(path, FileMode.OpenOrCreate))

{

// преобразуем строку в байты

byte[] buffer = Encoding.Default.GetBytes(text);

// запись массива байтов в файл

await fstream.WriteAsync(buffer, 0, buffer.Length);

Console.WriteLine("Текст записан в файл");

}

// чтение из файла

using (FileStream fstream = File.OpenRead(path))

{

// выделяем массив для считывания данных из файла

byte[] buffer = new byte[fstream.Length];

// считываем данные

await fstream.ReadAsync(buffer, 0, buffer.Length);

// декодируем байты в строку

string textFromFile = Encoding.Default.GetString(buffer);

Console.WriteLine($"Текст из файла: {textFromFile}");

}

BinnaryWriter

using (BinaryWriter writer = new BinaryWriter(File.Open(path, FileMode.OpenOrCreate)))

{

// записываем в файл строку

writer.Write("Tom");

// записываем в файл число int

writer.Write(37);

Console.WriteLine("File has been written");

}

BinaryReader

using (BinaryReader reader = new BinaryReader(File.Open("person.dat", FileMode.Open)))

{

// считываем из файла строку

string name = reader.ReadString();

// считываем из файла число

int age = reader.ReadInt32();

Console.WriteLine($"Name: {name} Age: {age}");

}

**классы для работы с файловой системой**

**//** **Пути к файлам**

string path = @"C:\Users\eugene\Documents\content.txt";

// string path = "/Users/eugene/Documents/content.txt"; // для MacOS/Linux

FileInfo fileInfo = new FileInfo(path);

if (fileInfo.Exists)

{

Console.WriteLine($"Имя файла: {fileInfo.Name}");

Console.WriteLine($"Время создания: {fileInfo.CreationTime}");

Console.WriteLine($"Размер: {fileInfo.Length}");

}

// Получение информации о файле

string path = @"C:\Users\eugene\Documents\content.txt";

// string path = "/Users/eugene/Documents/content.txt"; // для MacOS/Linux

FileInfo fileInfo = new FileInfo(path);

if (fileInfo.Exists)

{

Console.WriteLine($"Имя файла: {fileInfo.Name}");

Console.WriteLine($"Время создания: {fileInfo.CreationTime}");

Console.WriteLine($"Размер: {fileInfo.Length}");

}

// Удаление файла

string path = @"C:\app\content.txt";

FileInfo fileInf = new FileInfo(path);

if (fileInf.Exists)

{

fileInf.Delete();

// альтернатива с помощью класса File

// File.Delete(path);

}

// Перемещение файла

string path = @"C:\OldDir\content.txt";

string newPath = @"C:\NewDir\index.txt";

FileInfo fileInf = new FileInfo(path);

if (fileInf.Exists)

{

fileInf.MoveTo(newPath);

// альтернатива с помощью класса File

// File.Move(path, newPath);

}

//Копирование файла

string path = @"C:\OldDir\content.txt";

string newPath = @"C:\NewDir\index2.txt";

FileInfo fileInf = new FileInfo(path);

if (fileInf.Exists)

{

fileInf.CopyTo(newPath, true);

// альтернатива с помощью класса File

// File.Copy(path, newPath, true);

}

**Рефлексия**

Type myType = typeof(PeopleTypes.Person);

Console.WriteLine($"Name: {myType.Name}"); // получаем краткое имя типа

Console.WriteLine($"Full Name: {myType.FullName}"); // получаем полное имя типа

Console.WriteLine($"Namespace: {myType.Namespace}"); // получаем пространство имен типа

Console.WriteLine($"Is struct: {myType.IsValueType}"); // является ли тип структурой

Console.WriteLine($"Is class: {myType.IsClass}"); // является ли тип классом

namespace PeopleTypes

{

class Person

{

public string Name { get; }

public Person(string name) => Name = name;

}

}

**Сериализация**

[Serializable]

class Person

{

public string Name { get; set; }

public int Year { get; set; }

[NonSerialized]

public string accNumber;

public Person(string name, int year, string acc)

{

Name = name;

Year = year;

accNumber = acc;

}

}

**BinaryFormatter**

class Program

{

static void Main(string[] args)

{

// объект для сериализации

Person person = new Person("Tom", 29);

Console.WriteLine("Объект создан");

// создаем объект BinaryFormatter

BinaryFormatter formatter = new BinaryFormatter();

// получаем поток, куда будем записывать сериализованный объект

using (FileStream fs = new FileStream("people.dat", FileMode.OpenOrCreate))

{

formatter.Serialize(fs, person);

Console.WriteLine("Объект сериализован");

}

// десериализация из файла people.dat

using (FileStream fs = new FileStream("people.dat", FileMode.OpenOrCreate))

{

Person newPerson = (Person)formatter.Deserialize(fs);

Console.WriteLine("Объект десериализован");

Console.WriteLine($"Имя: {newPerson.Name} --- Возраст: {newPerson.Age}");

}

Console.ReadLine();

}

}

**JSON**

using System.Text.Json;

// сохранение данных

using (FileStream fs = new FileStream("user.json", FileMode.OpenOrCreate))

{

Person tom = new Person("Tom", 37);

await JsonSerializer.SerializeAsync<Person>(fs, tom);

Console.WriteLine("Data has been saved to file");

}

// чтение данных

using (FileStream fs = new FileStream("user.json", FileMode.OpenOrCreate))

{

Person? person = await JsonSerializer.DeserializeAsync<Person>(fs);

Console.WriteLine($"Name: {person?.Name} Age: {person?.Age}");

}

class Person

{

public string Name { get;}

public int Age { get; set; }

public Person(string name, int age)

{

Name = name;

Age = age;

}

}

**XML Сериализация**

using System.Xml.Serialization;

// объект для сериализации

Person person = new Person("Tom", 37);

// передаем в конструктор тип класса Person

XmlSerializer xmlSerializer = new XmlSerializer(typeof(Person));

// получаем поток, куда будем записывать сериализованный объект

using (FileStream fs = new FileStream("person.xml", FileMode.OpenOrCreate))

{

xmlSerializer.Serialize(fs, person);

Console.WriteLine("Object has been serialized");

}

//[Serializable]

public class Person

{

public string Name { get; set; } = "Undefined";

public int Age { get; set; } = 1;

public Person() { }

public Person(string name, int age)

{

Name = name;

Age = age;

}

}

**XML Диссериализация**  
using System.Xml.Serialization;

XmlSerializer xmlSerializer = new XmlSerializer(typeof(Person));

// десериализуем объект

using (FileStream fs = new FileStream("person.xml", FileMode.OpenOrCreate))

{

Person? person = xmlSerializer.Deserialize(fs) as Person;

Console.WriteLine($"Name: {person?.Name} Age: {person?.Age}");

}

public class Person

{

public string Name { get; set; } = "Undefined";

public int Age { get; set; } = 1;

public Person() { }

public Person(string name, int age)

{

Name = name;

Age = age;

}

}

**Процессы**

using System.Diagnostics;

var process = Process.GetCurrentProcess();

Console.WriteLine($"Id: {process.Id}");

Console.WriteLine($"Name: {process.ProcessName}");

Console.WriteLine($"VirtualMemory: {process.VirtualMemorySize64}");

using System.Diagnostics;

Process proc = Process.GetProcessesByName("devenv")[0]; // Windows

// Process proc = Process.GetProcessesByName("VisualStudio")[0]; // MacOS

ProcessThreadCollection processThreads = proc.Threads;

foreach(ProcessThread thread in processThreads)

{

Console.WriteLine($"ThreadId: {thread.Id}");

}

Запуск нового процесса

Process.Start(@"C:\Program Files\Google\Chrome\Application\chrome");

//более сложный пример

ProcessStartInfo procInfo = new ProcessStartInfo();

// исполняемый файл программы - браузер хром

procInfo.FileName = @"C:\Program Files\Google\Chrome\Application\chrome";

// аргументы запуска - адрес интернет-ресурса

procInfo.Arguments = "https://metanit.com";

Process.Start(procInfo);

**потоки – управление и синхронизация**

int x = 0;

object locker = new(); // объект-заглушка

// запускаем пять потоков

for (int i = 1; i < 6; i++)

{

Thread myThread = new(Print);

myThread.Name = $"Поток {i}";

myThread.Start();

}

void Print()

{

lock (locker)

{

x = 1;

for (int i = 1; i < 6; i++)

{

Console.WriteLine($"{Thread.CurrentThread.Name}: {x}");

x++;

Thread.Sleep(100);

}

}

}

//Monitor

int x = 0;

object locker = new(); // объект-заглушка

// запускаем пять потоков

for (int i = 1; i < 6; i++)

{

Thread myThread = new(Print);

myThread.Name = $"Поток {i}";

myThread.Start();

}

void Print()

{

bool acquiredLock = false;

try

{

Monitor.Enter(locker, ref acquiredLock);

x = 1;

for (int i = 1; i < 6; i++)

{

Console.WriteLine($"{Thread.CurrentThread.Name}: {x}");

x++;

Thread.Sleep(100);

}

}

finally

{

if (acquiredLock) Monitor.Exit(locker);

}

}

//Mutex

int x = 0;

Mutex mutexObj = new();

// запускаем пять потоков

for (int i = 1; i < 6; i++)

{

Thread myThread = new(Print);

myThread.Name = $"Поток {i}";

myThread.Start();

}

void Print()

{

mutexObj.WaitOne(); // приостанавливаем поток до получения мьютекса

x = 1;

for (int i = 1; i < 6; i++)

{

Console.WriteLine($"{Thread.CurrentThread.Name}: {x}");

x++;

Thread.Sleep(100);

}

mutexObj.ReleaseMutex(); // освобождаем мьютекс

}

//Семафоры

// запускаем пять потоков

for (int i = 1; i < 6; i++)

{

Reader reader = new Reader(i);

}

class Reader

{

// создаем семафор

static Semaphore sem = new Semaphore(3, 3);

Thread myThread;

int count = 3;// счетчик чтения

public Reader(int i)

{

myThread = new Thread(Read);

myThread.Name = $"Читатель {i}";

myThread.Start();

}

public void Read()

{

while (count > 0)

{

sem.WaitOne(); // ожидаем, когда освободиться место

Console.WriteLine($"{Thread.CurrentThread.Name} входит в библиотеку");

Console.WriteLine($"{Thread.CurrentThread.Name} читает");

Thread.Sleep(1000);

Console.WriteLine($"{Thread.CurrentThread.Name} покидает библиотеку");

sem.Release(); // освобождаем место

count--;

Thread.Sleep(1000);

}

}

}

**Task**

Task task1 = new Task(() => Console.WriteLine("Task1 is executed"));

task1.Start();

Task task2 = Task.Factory.StartNew(() => Console.WriteLine("Task2 is executed"));

Task task3 = Task.Run(() => Console.WriteLine("Task3 is executed"));

task1.Wait(); // ожидаем завершения задачи task1

task2.Wait(); // ожидаем завершения задачи task2

task3.Wait(); // ожидаем завершения задачи task3

**//Синхронный запуск задачи**

Console.WriteLine("Main Starts");

// создаем задачу

Task task1 = new Task(() =>

{

Console.WriteLine("Task Starts");

Thread.Sleep(1000);

Console.WriteLine("Task Ends");

});

task1.RunSynchronously(); // запускаем задачу синхронно

Console.WriteLine("Main Ends"); // этот вызов ждет завершения задачи task1

**//вложенные задачи**

var outer = Task.Factory.StartNew(() => // внешняя задача

{

Console.WriteLine("Outer task starting...");

var inner = Task.Factory.StartNew(() => // вложенная задача

{

Console.WriteLine("Inner task starting...");

Thread.Sleep(2000);

Console.WriteLine("Inner task finished.");

});

});

outer.Wait(); // ожидаем выполнения внешней задачи

Console.WriteLine("End of Main");

//Возвращение результатов из задачи

int n1 =4, n2 = 5;

Task<int> sumTask = new Task<int>(() => Sum(n1, n2));

sumTask.Start();

int result = sumTask.Result;

Console.WriteLine($"{n1} + {n2} = {result}"); // 4 + 5 = 9

int Sum(int a, int b) => a + b;

**//Задачи продолжения**

Task task1 = new Task(() =>

{

Console.WriteLine($"Id задачи: {Task.CurrentId}");

});

// задача продолжения - task2 выполняется после task1

Task task2 = task1.ContinueWith(PrintTask);

task1.Start();

// ждем окончания второй задачи

task2.Wait();

Console.WriteLine("Конец метода Main");

void PrintTask(Task t)

{

Console.WriteLine($"Id задачи: {Task.CurrentId}");

Console.WriteLine($"Id предыдущей задачи: {t.Id}");

Thread.Sleep(3000);

}

**//Parallel**

Parallel.Invoke(

Print,

() =>

{

Console.WriteLine($"Выполняется задача {Task.CurrentId}");

Thread.Sleep(3000);

},

() => Square(5)

);

void Print()

{

Console.WriteLine($"Выполняется задача {Task.CurrentId}");

Thread.Sleep(3000);

}

// вычисляем квадрат числа

void Square(int n)

{

Console.WriteLine($"Выполняется задача {Task.CurrentId}");

Thread.Sleep(3000);

Console.WriteLine($"Результат {n \* n}");

}

//CancellationToken

CancellationTokenSource cancelTokenSource = new CancellationTokenSource();

CancellationToken token = cancelTokenSource.Token;

// задача вычисляет квадраты чисел

Task task = new Task(() =>

{

for (int i = 1; i < 10; i++)

{

if (token.IsCancellationRequested) // проверяем наличие сигнала отмены задачи

{

Console.WriteLine("Операция прервана");

return; // выходим из метода и тем самым завершаем задачу

}

Console.WriteLine($"Квадрат числа {i} равен {i \* i}");

Thread.Sleep(200);

}

}, token);

task.Start();

Thread.Sleep(1000);

// после задержки по времени отменяем выполнение задачи

cancelTokenSource.Cancel();

// ожидаем завершения задачи

Thread.Sleep(1000);

// проверяем статус задачи

Console.WriteLine($"Task Status: {task.Status}");

cancelTokenSource.Dispose(); // освобождаем ресурсы

**TPL**

await PrintAsync(); // вызов асинхронного метода

Console.WriteLine("Некоторые действия в методе Main");

void Print()

{

Thread.Sleep(3000); // имитация продолжительной работы

Console.WriteLine("Hello METANIT.COM");

}

// определение асинхронного метода

async Task PrintAsync()

{

Console.WriteLine("Начало метода PrintAsync"); // выполняется синхронно

await Task.Run(() => Print()); // выполняется асинхронно

Console.WriteLine("Конец метода PrintAsync");

}

**Паттерны**

**//** **Билдер**

class Client

{

void Main()

{

Builder builder = new ConcreteBuilder();

Director director = new Director(builder);

director.Construct();

Product product = builder.GetResult();

}

}

class Director

{

Builder builder;

public Director(Builder builder)

{

this.builder = builder;

}

public void Construct()

{

builder.BuildPartA();

builder.BuildPartB();

builder.BuildPartC();

}

}

abstract class Builder

{

public abstract void BuildPartA();

public abstract void BuildPartB();

public abstract void BuildPartC();

public abstract Product GetResult();

}

class Product

{

List<object> parts = new List<object>();

public void Add(string part)

{

parts.Add(part);

}

}

class ConcreteBuilder : Builder

{

Product product = new Product();

public override void BuildPartA()

{

product.Add("Part A");

}

public override void BuildPartB()

{

product.Add("Part B");

}

public override void BuildPartC()

{

product.Add("Part C");

}

public override Product GetResult()

{

return product;

}

}

**//Singleton**

**class Singleton**

**{**

**private static Singleton instance;**

**private Singleton()**

**{}**

**public static Singleton getInstance()**

**{**

**if (instance == null)**

**instance = new Singleton();**

**return instance;**

**}**

**}**

//конкретный пример

class Program

{

static void Main(string[] args)

{

Computer comp = new Computer();

comp.Launch("Windows 8.1");

Console.WriteLine(comp.OS.Name);

// у нас не получится изменить ОС, так как объект уже создан

comp.OS = OS.getInstance("Windows 10");

Console.WriteLine(comp.OS.Name);

Console.ReadLine();

}

}

class Computer

{

public OS OS { get; set; }

public void Launch(string osName)

{

OS = OS.getInstance(osName);

}

}

class OS

{

private static OS instance;

public string Name { get; private set; }

protected OS(string name)

{

this.Name=name;

}

public static OS getInstance(string name)

{

if (instance == null)

instance = new OS(name);

return instance;

}

}

**//Decorator**

class Program

{

static void Main(string[] args)

{

Pizza pizza1 = new ItalianPizza();

pizza1 = new TomatoPizza(pizza1); // итальянская пицца с томатами

Console.WriteLine("Название: {0}", pizza1.Name);

Console.WriteLine("Цена: {0}", pizza1.GetCost());

Pizza pizza2 = new ItalianPizza();

pizza2 = new CheesePizza(pizza2);// итальянская пиццы с сыром

Console.WriteLine("Название: {0}", pizza2.Name);

Console.WriteLine("Цена: {0}", pizza2.GetCost());

Pizza pizza3 = new BulgerianPizza();

pizza3 = new TomatoPizza(pizza3);

pizza3 = new CheesePizza(pizza3);// болгарская пиццы с томатами и сыром

Console.WriteLine("Название: {0}", pizza3.Name);

Console.WriteLine("Цена: {0}", pizza3.GetCost());

Console.ReadLine();

}

}

abstract class Pizza

{

public Pizza(string n)

{

this.Name = n;

}

public string Name {get; protected set;}

public abstract int GetCost();

}

class ItalianPizza : Pizza

{

public ItalianPizza() : base("Итальянская пицца")

{ }

public override int GetCost()

{

return 10;

}

}

class BulgerianPizza : Pizza

{

public BulgerianPizza()

: base("Болгарская пицца")

{ }

public override int GetCost()

{

return 8;

}

}

abstract class PizzaDecorator : Pizza

{

protected Pizza pizza;

public PizzaDecorator(string n, Pizza pizza) : base(n)

{

this.pizza = pizza;

}

}

class TomatoPizza : PizzaDecorator

{

public TomatoPizza(Pizza p)

: base(p.Name + ", с томатами", p)

{ }

public override int GetCost()

{

return pizza.GetCost() + 3;

}

}

class CheesePizza : PizzaDecorator

{

public CheesePizza(Pizza p)

: base(p.Name + ", с сыром", p)

{ }

public override int GetCost()

{

return pizza.GetCost() + 5;

}

}

**//Prototype**

class Client

{

void Operation()

{

Prototype prototype = new ConcretePrototype1(1);

Prototype clone = prototype.Clone();

prototype = new ConcretePrototype2(2);

clone = prototype.Clone();

}

}

abstract class Prototype

{

public int Id { get; private set; }

public Prototype(int id)

{

this.Id = id;

}

public abstract Prototype Clone();

}

class ConcretePrototype1 : Prototype

{

public ConcretePrototype1(int id)

: base(id)

{ }

public override Prototype Clone()

{

return new ConcretePrototype1(Id);

}

}

class ConcretePrototype2 : Prototype

{

public ConcretePrototype2(int id)

: base(id)

{ }

public override Prototype Clone()

{

return new ConcretePrototype2(Id);

}

}

**//Memento**

class Memento

{

public string State { get; private set;}

public Memento(string state)

{

this.State = state;

}

}

class Caretaker

{

public Memento Memento { get; set; }

}

class Originator

{

public string State { get; set; }

public void SetMemento(Memento memento)

{

State = memento.State;

}

public Memento CreateMemento()

{

return new Memento(State);

}

}

**//Adapter**

class Client

{

public void Request(Target target)

{

target.Request();

}

}

// класс, к которому надо адаптировать другой класс

class Target

{

public virtual void Request()

{}

}

// Адаптер

class Adapter : Target

{

private Adaptee adaptee = new Adaptee();

public override void Request()

{

adaptee.SpecificRequest();

}

}

// Адаптируемый класс

class Adaptee

{

public void SpecificRequest()

{}

}

**//Заместитель (Прокси)**

class Client

{

void Main()

{

Subject subject = new Proxy();

subject.Request();

}

}

abstract class Subject

{

public abstract void Request();

}

class RealSubject : Subject

{

public override void Request()

{}

}

class Proxy : Subject

{

RealSubject realSubject;

public override void Request()

{

if (realSubject == null)

realSubject = new RealSubject();

realSubject.Request();

}

}