

struct Customer

{

public string LastName;

public string FirstName;

public string MiddleName;

public string Address;

public string PhoneNumber;

public string CreditCardNumber;

}

class Program

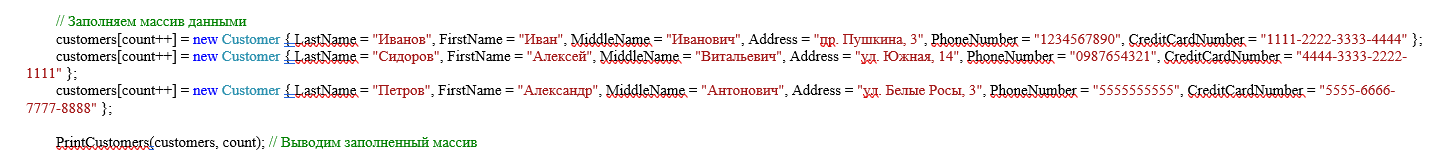
{

static void Main()

{

Customer[] customers = new Customer[100];

int count = 0;



// Удаляем K элементов из начала массива

int K = 1; // Количество удаляемых элементов

count = RemoveCustomers(customers, count, K);

Console.WriteLine($"\nПосле удаления {K} элементов:");

PrintCustomers(customers, count);

// Вводим количество добавляемых элементов с клавиатуры

Console.Write("Введите количество добавляемых элементов: ");

int N;

while (!int.TryParse(Console.ReadLine(), out N) || N < 1)

{

Console.WriteLine("Пожалуйста, введите положительное число.");

}

AddCustomers(customers, ref count, N);

Console.WriteLine($"\nПосле добавления {N} новых элементов:");

PrintCustomers(customers, count);

}

static void PrintCustomers(Customer[] customers, int count)

{

Console.WriteLine("Список покупателей:");

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

{

Console.WriteLine($"Фамилия: {customers[i].LastName}, Имя: {customers[i].FirstName}, Отчество: {customers[i].MiddleName}, " +

$"Адрес: {customers[i].Address}, Телефон: {customers[i].PhoneNumber}, Номер карты: {customers[i].CreditCardNumber}");

}

}

static int RemoveCustomers(Customer[] customers, int count, int K)

{

if (K >= count)

{

return 0; // Если K больше или равно количеству, возвращаем 0

}

for (int i = K; i < count; i++)

{

customers[i - K] = customers[i]; // Сдвигаем элементы влево

}

return count - K; // Возвращаем новую длину массива

}

static void AddCustomers(Customer[] customers, ref int count, int N)

{

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

{

customers[count++] = new Customer

{

LastName = $"Иванов{i + 1}",

FirstName = $"Василий{i + 1}",

MiddleName = $"Викторович{i + 1}",

Address = $"пр. Клецкова, {i + 1}",

PhoneNumber = $"7777777{i}",

CreditCardNumber = $"9999-8888-7777-{i + 1}999"

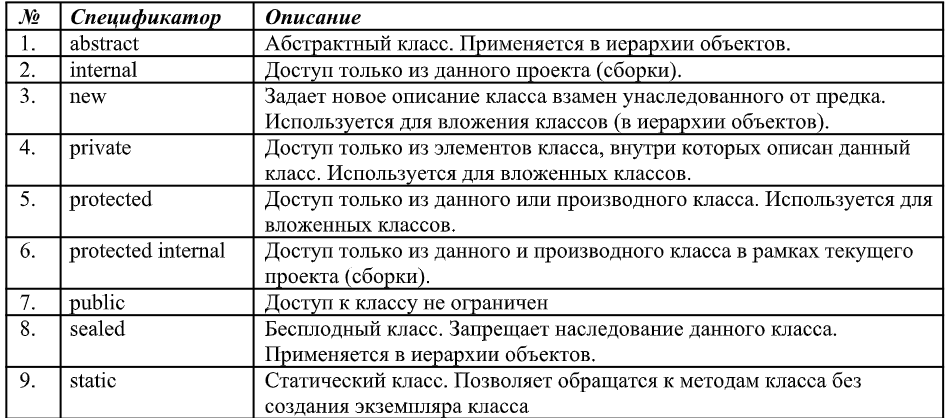
};

}

}

}

Класс – это обобщенное понятие, определяющее характеристики и поведение некоторого множества объектов, называемых экземплярами класса. В программном понимании класс является типом данных, определяемым пользователем, в котором объединены структуры данных и методы их обработки. Описание класса содержит ключевое слово class, за которым следует его имя, а далее в фигурных скобках — тело класса. Кроме того, для класса можно задать его базовый класс (предок) и ряд необязательных атрибутов и спецификаторов, определяющих различные характеристики класса: [ атрибуты ] [ спецификаторы ] class имя\_класса [ : предки ]{тело\_класса}



Индексатор представляет собой разновидность свойства и позволяет организовать доступ к скрытым полям класса по индексу, например, так же, как мы обращаемся к элементу массива.

В С# существует специальный вид метода, называемый деструктором, который вызывается сборщиком мусора непосредственно перед удалением объекта из памяти. В деструкторе описываются действия, гарантирующие корректность последующего удаления объекта. Например, проверяется все ли ресурсы, используемые объектом, освобождены (файлызакрыты, удаленное соединение разорвано и т. п.).

Индексатор представляет собой разновидность свойства и позволяет организовать доступ к скрытым полям класса по индексу, например, так же, как мы обращаемся к элементу массива.

Пример программы с классами:class Car

{

// Свойства класса

public string Make { get; set; }

public string Model { get; set; }

public int Year { get; set;

public Car(string make, string model, int year) // Конструктор

{

Make = make;

Model = model;

Year = year;

}

public void DisplayInfo() // Метод для отображения информации об автомобиле

{

Console.WriteLine($"Автомобиль: {Year} {Make} {Model}");

}

}

class Program { static void Main()

{

Car car1 = new Car("Toyota", "Camry", 2021);

Car car2 = new Car("Honda", "Civic", 2020); car1.DisplayInfo(); car2.DisplayInfo();

}

}

Наследование в C# — это один из основных принципов объектно-ориентированного программирования (ООП), который позволяет создавать новые классы на основе существующих.

class Animal

{ public void Speak()

{

Console.WriteLine("Животное делает звук.");

}

}

class Dog : Animal

{

public void Bark()

{

Console.WriteLine("Собака лает.");

}

}

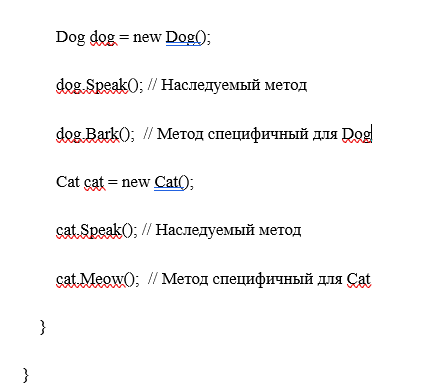
class Cat : Animal

{

public void Meow()

{

Console.WriteLine("Кошка мяукает.");

 }

}

class Program

{

static void Main()

{