**ПРАКТИЧЕСКАЯ РАБОТА № 23**

**Структура. Перегрузка методов и классов С#**

****

**Цель работы**: научиться разрабатывать СТРУКТУРЫ, их основныекомпоненты: поля, методы, конструкторы, создавать объекты класса, обращаться к открытым полям и методам СЬРУКТУРЫ.

Для защиты практической работы необходимо изучить теоретическую часть, выполнить практическое задание, ответить на вопросы.

|  |  |  |
| --- | --- | --- |
| №  п/п | Ф.И.О. | НОМЕР ВАРИАНТА/ЗАДАНИЯ |
|  | Барбарас Александр | 1 |
|  | Бортник Иван Артемович | 2 |
|  | Бура Артур Дмитриевич | 3 |
|  | Вернер Андрей Уриелович | 4 |
|  | Дабежа Максим Сергеевич | 5 |
|  | Давискиба Максим Александрович | 6 |
|  | Деде Иван Дмитриевич | 7 |
|  | Дидур Даниил Юрьевич | 8 |
|  | Доцин Даниил Владимирович | 9 |
|  | Журба Иван Сергеевич | Набор примера из лекционного материала |
|  | Занделов Данил Павлович | 10 |
|  | Иордатий Кирилл Валерьевич | 11 |
|  | Кирчу Игорь | 12 |
|  | Кырлан Александра Олеговна | 13 |
|  | Лозинский Даниил Максимович | 14 |
|  | Макаренко Данил Александрович | 15 |
|  | Мартынов Даниил Анатольевич | 1 |
|  | Марущак Анатолий Сергеевич | Набор примера из лекционного материала |
|  | Морошан Никита Павлович | 2 |
|  | Мотынга Иван Александрович | 3 |
|  | Павлин Кирилл Юрьевич | 4 |
|  | Розенцвит Михаил Александрович | 5 |
|  | Сагайдак Глеб Андреевич | 6 |
|  | Сакара Анна Ивановна | 7 |
|  | Сулак Екатерина Александровна | 8 |
|  | Ушаков Геннадий Геннадьевич | 9 |
|  | Чайковский Андриан Андреевич | 10 |
|  | Чепалыга Амалия Максимовна | 11 |
|  | Шевченко Александр Александрович | 12 |
|  | Шеленков Кирилл Вячеславович | 13 |

**Теоретические сведения**

**Структура.**

**Определение структуры**

Наряду с классами структуры представляют еще один способ создания собственных типов данных в C#. Более того многие примитивные типы, например, int, double и т.д., по сути являются структурами.

**Определение структуры**

Для определения структуры применяется ключевое слово **struct**:

|  |  |
| --- | --- |
| 1  2  3  4 | struct имя\_структуры  {      // элементы структуры  } |

После слова **struct** идет название структуры и далее в фигурных скобках размещаются элементы структуры - поля, методы и т.д.

Например, определим структуру, которая будет называться Person и которая будет представлять человека:

|  |  |
| --- | --- |
| 1  2  3 | struct Person  {  } |

Начиная с версии C# 12, если структура имеет пустое определение (не содержат полей, свойств, методов), то фигурные скобки после названия типа можно не использовать:

|  |  |
| --- | --- |
| 1 | struct Person; |

Как и классы, структуры могут хранить состояние в виде полей (переменных) и определять поведение в виде методов. Например, добавим в структуру Person пару полей и метод:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | struct Person  {      public string name;      public int age;        public void Print()      {          Console.WriteLine($"Имя: {name}  Возраст: {age}");      }  } |

В данном случае определены две переменные - name и age для хранения соответственно имени и возраста человека и метод Print для вывода информации о человеке на консоль.

И как и в случае с классами, для обращения к функциональности структуры - полям, методам и другим компонентам структуры применяется точечная нотация - после объекта структуры ставится точка, а затем указывается компонент структуры:

|  |  |
| --- | --- |
| 1  2 | объект.поле\_структуры  объект.метод\_структуры(параметры\_метода) |

**Создание объекта структуры**

**Инициализация с помощью конструктора**

Для использования структуры ее необходмо инициализировать. Для инициализации создания объектов структуры, как и в случае с классами, применяется вызов конструктура с оператором **new**. Даже если в коде стуктуры не определено ни одного конструктора, тем не менее имеет как минимум один конструктор - конструктор по умолчанию, который генерируется компилятором. Этот конструктор не принимает параметров и создает объект структуры со значениями по умолчанию.

|  |  |
| --- | --- |
| 1 | new название\_структуры(); |

Например, создадим объект структуры Person с помощью конструктора по умолчанию:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | Person tom = new Person();  // вызов конструктора  // или так  // Person tom = new();    tom.name = "Tom";   // изменяем значение по умолчанию в поле name    tom.Print();    // Имя: Tom  Возраст: 0    struct Person  {      public string name;      public int age;        public void Print()      {          Console.WriteLine($"Имя: {name}  Возраст: {age}");      }  } |

В данном случае создается объект tom. Для его создания вызывается конструктор по умолчанию, который устанавливает значения по умолчанию для его полей. Для числовых данных это значение 0, поэтому поле age будет иметь значение 0. Для строк это значение null, которое указывает на отсутствие значения. Но далее, если поля доступны (а в данном случае поскольку они имеют модификатор public они доступны), мы можем изменить их значения. Так, здесь полю name присваивается строка "Tom". Соответственно при выполнении метода Print() мы получим следующий консольный вывод:

Имя: Tom Возраст: 0

**Непосредственная иницилизация полей**

Если все поля структуры доступны (как в случае с полями структуры Person, который имеют модификатор **public**), то структуру можно инициализировать без вызова конструктора. В этом случае необходимо присвоить значения всем полям структуры перед получением значений полей и обращением к методам структуры. Например:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | Person tom;         // не вызываем конструктор  // инициализация полей  tom.name = "Sam";  tom.age = 37;    tom.Print();    // Имя: Sam  Возраст: 37    struct Person  {      public string name;      public int age;        public void Print()      {          Console.WriteLine($"Имя: {name}  Возраст: {age}");      }  } |

**Инициализация полей по умолчанию**

Стоит отметить, что начиная с версии C# 10, мы можем напрямую инициализировать поля структуры при их определении (до C# 10 это делать было нельзя):

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | Person tom = new Person();  tom.Print();    // Имя:Tom  Возраст: 1    struct Person  {      // инициализация полей значениями по умолчанию - доступна с C#10      public string name = "Tom";      public int age = 1;      public Person() { }      public void Print() =>Console.WriteLine($"Имя: {name}  Возраст: {age}");  } |

Однако даже в этом случае, несмотря на значения по умолчанию, необходимо явно определить и вызывать конструктор, если мы хотим использовать эти значения.

**Конструкторы структуры**

Как и класс, структура может определять конструкторы. Например, добавим в структуру Person конструктор:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | Person tom = new();  Person bob = new("Bob");  Person sam = new("Sam", 25);    tom.Print();    // !!!! Имя:   Возраст: 0  bob.Print();    // Имя: Bob  Возраст: 1  sam.Print();    // Имя: Sam  Возраст: 25    struct Person  {      public string name;      public int age;        public Person(string name = "Tom", int age = 1)      {          this.name = name;          this.age = age;      }      public void Print() => Console.WriteLine($"Имя: {name}  Возраст: {age}");  } |

В данном случае в структуре Person определен конструктор с двумя параметрами, для которых предоставлены значения по умолчания. Однако обратите внимание на создание первого объекта структуры:

|  |  |
| --- | --- |
| 1  2 | Person tom = new(); // по прежнему используется конструктор без параметров по умолчанию  tom.Print();    // !!!! Имя:   Возраст: 0 |

Здесь по-прежнему применяется конструктор по умолчанию, тогда как при инициализации остальных двух переменных структуры применяется явно определенный конструктор.

Однако начиная с версии C# 10 мы можем определить свой конструктор без параметров:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | Person tom = new();    tom.Print();    // Имя: Tom  Возраст: 37    struct Person  {      public string name;      public int age;        public Person()      {          name = "Tom";          age = 37;      }      public void Print() => Console.WriteLine($"Имя: {name}  Возраст: {age}");  } |

Стоит отметить, что до версии C# 11 при определении конструктора структуру в нем необходимо было инициализировать все поля структуры, начиная с версии C# 11 это делать необязательно.

В случае если нам необходимо вызывать конструкторы с различным количеством параметров, то мы можем, как и в случае с классами, вызывать их по цепочке:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | Person tom = new();  Person bob = new("Bob");  Person sam = new("Sam", 25);    tom.Print();    // Имя: Tom  Возраст: 1  bob.Print();    // Имя: Bob  Возраст: 1  sam.Print();    // Имя: Sam  Возраст: 25    struct Person  {      public string name;      public int age;        public Person() : this("Tom")      { }      public Person(string name) : this(name, 1)      { }      public Person(string name, int age)      {          this.name = name;          this.age = age;      }      public void Print() => Console.WriteLine($"Имя: {name}  Возраст: {age}");  } |

Консольный вывод программы:

Имя: Tom Возраст: 1

Имя: Bob Возраст: 1

Имя: Sam Возраст: 25

Начиная с версии **C# 12** для структур, как и для классов, можно определять первичные конструкторы. Первичные конструкторы позволяют добавлять параметры к определению класса/структуры и использовать эти параметры внутри класса/структуры:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | var tom = new Person("Tom", 38);  tom.Print();    public struct Person(string name, int age)  {      public Person(string name) : this(name, 18) { }      public void Print() => Console.WriteLine($"name: {name}, age: {age}");  } |

Здесь для структуры Person определен первичный конструктор с двумя параметрами - name и age. За кадром для каждого параметра первичного конструктора в классе создается приватное поле, которое хранит значение параметра. Благодаря этому они могут использоваться в теле класса.

Кроме первичных конструкторов класс может определять дополнительные конструкторы, как примере выше. Но эти дополнительные конструкторы должны вызывать первичный конструктор:

|  |  |
| --- | --- |
| 1 | public Person(string name) : this(name, 18) { } |

**Инициализатор структуры**

Также, как и для класса, можно использовать инициализатор для создания структуры:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | Person tom = new Person { name = "Tom", age = 22 };    tom.Print();    // Имя: Tom  Возраст: 22    struct Person  {      public string name;      public int age;      public void Print() => Console.WriteLine($"Имя: {name}  Возраст: {age}");  } |

При использовании инициализатора сначала вызывается конструктор без параметров: если мы явным образом не определили конструктор без параметров, то вызывается конструктор по умолчанию. А затем его полям присваиваются соответствующие значения.

**Копирование структуры с помощью with**

Если нам необходимо скопировать в один объект структуры значения из другого с небольшими изменениями, то мы можем использовать оператор **with**:

|  |  |
| --- | --- |
| 1  2  3 | Person tom = new Person { name = "Tom", age = 22 };  Person bob = tom with { name = "Bob" };  bob.Print();    // Имя: Bob  Возраст: 22 |

В данном случае объект bob получает все значения объекта tom, а затем после оператора **with** в фигурных скобках указывается поля со значениями, которые мы хотим изменить.

**Перегрузка методов и классов С#**

Наряду с методами мы можем также перегружать операторы. Например, пусть у нас есть следующий класс Counter:

class Counter

{

public int Value { get; set; }

}

Данный класс представляет некоторый счетчик, значение которого хранится в свойстве Value.

И допустим, у нас есть два объекта класса Counter - два счетчика, которые мы хотим сравнивать или складывать на основании их свойства Value, используя стандартные операции сравнения и сложения:

Counter c1 = new Counter { Value = 23 };

Counter c2 = new Counter { Value = 45 };

bool result = c1 > c2;

Counter c3 = c1 + c2;

Но на данный момент ни операция сравнения, ни операция сложения для объектов Counter не доступны. Эти операции могут использоваться для ряда примитивных типов. Например, по умолчанию мы можем складывать числовые значения, но как складывать объекты комплексных типов - классов и структур компилятор не знает. И для этого нам надо выполнить перегрузку нужных нам операторов.

Перегрузка операторов заключается в определении в классе, для объектов которого мы хотим определить оператор, специального метода:

public static возвращаемый\_тип operator оператор(параметры)

{ }

Этот метод должен иметь модификаторы public static, так как перегружаемый оператор будет использоваться для всех объектов данного класса. Далее идет название возвращаемого типа. Возвращаемый тип представляет тот тип, объекты которого мы хотим получить. К примеру, в результате сложения двух объектов Counter мы ожидаем получить новый объект Counter. А в результате сравнения двух мы хотим получить объект типа bool, который указывает истинно ли условное выражение или ложно. Но в зависимости от задачи возвращаемые типы могут быть любыми.

Затем вместо названия метода идет ключевое слово operator и собственно сам оператор. И далее в скобках перечисляются параметры. Бинарные операторы принимают два параметра, унарные - один параметр. И в любом случае один из параметров должен представлять тот тип - класс или структуру, в котором определяется оператор.

Например, перегрузим ряд операторов для класса Counter:

class Counter

{

public int Value { get; set; }

public static Counter operator +(Counter c1, Counter c2)

{

return new Counter { Value = c1.Value + c2.Value };

}

public static bool operator >(Counter c1, Counter c2)

{

return c1.Value > c2.Value;

}

public static bool operator <(Counter c1, Counter c2)

{

return c1.Value < c2.Value;

}

}

Поскольку все перегруженные операторы - бинарные - то есть проводятся над двумя объектами, то для каждой перегрузки предусмотрено по два параметра.

Так как в случае с операцией сложения мы хотим сложить два объекта класса Counter, то оператор принимает два объекта этого класса. И так как мы хотим в результате сложения получить новый объект Counter, то данный класс также используется в качестве возвращаемого типа. Все действия этого оператора сводятся к созданию, нового объекта, свойство Value которого объединяет значения свойства Value обоих параметров:

public static Counter operator +(Counter c1, Counter c2)

{

return new Counter { Value = c1.Value + c2.Value };

}

Также переопределены две операции сравнения. Если мы переопределяем одну из этих операций сравнения, то мы также должны переопределить вторую из этих операций. Сами операторы сравнения сравнивают значения свойств Value и в зависимости от результата сравнения возвращают либо true, либо false.

Теперь используем перегруженные операторы в программе:

static void Main(string[] args)

{

Counter c1 = new Counter { Value = 23 };

Counter c2 = new Counter { Value = 45 };

bool result = c1 > c2;

Console.WriteLine(result); // false

Counter c3 = c1 + c2;

Console.WriteLine(c3.Value); // 23 + 45 = 68

Console.ReadKey();

}

Стоит отметить, что так как по сути определение оператора представляет собой метод, то этот метод мы также можем перегрузить, то есть создать для него еще одну версию. Например, добавим в класс Counter еще один оператор:

public static int operator +(Counter c1, int val)

{

return c1.Value + val;

}

Данный метод складывает значение свойства Value и некоторое число, возвращая их сумму. И также мы можем применить этот оператор:

Counter c1 = new Counter { Value = 23 };

int d = c1 + 27; // 50

Console.WriteLine(d);

Следует учитывать, что при перегрузке не должны изменяться те объекты, которые передаются в оператор через параметры. Например, мы можем определить для класса Counter оператор инкремента:

public static Counter operator ++(Counter c1)

{

c1.Value += 10;

return c1;

}

Поскольку оператор унарный, он принимает только один параметр - объект того класса, в котором данный оператор определен. Но это неправильное определение инкремента, так как оператор не должен менять значения своих параметров.

И более корректная перегрузка оператора инкремента будет выглядеть так:

public static Counter operator ++(Counter c1)

{

return new Counter { Value = c1.Value + 10 };

}

То есть возвращается новый объект, который содержит в свойстве Value инкрементированное значение.

При этом нам не надо определять отдельно операторы для префиксного и для постфиксного инкремента (а также декремента), так как одна реализация будет работать в обоих случаях.

Например, используем операцию префиксного инкремента:

Counter counter = new Counter() { Value = 10 };

Console.WriteLine($"{counter.Value}"); // 10

Console.WriteLine($"{(++counter).Value}"); // 20

Console.WriteLine($"{counter.Value}"); // 20

Теперь используем постфиксный инкремент:

Counter counter = new Counter() { Value = 10 };

Console.WriteLine($"{counter.Value}"); // 10

Console.WriteLine($"{(counter++).Value}"); // 10

Console.WriteLine($"{counter.Value}"); // 20

Консольный вывод:

10

10

20

Также стоит отметить, что мы можем переопределить операторы true и false. Например, определим их в классе Counter:

class Counter

{

public int Value { get; set; }

public static bool operator true(Counter c1)

{

return c1.Value != 0;

}

public static bool operator false(Counter c1)

{

return c1.Value == 0;

}

// остальное содержимое класса

}

Эти операторы перегружаются, когда мы хотим использовать объект типа в качестве условия. Например:

Counter counter = new Counter() { Value = 0 };

if (counter)

Console.WriteLine(true);

else

Console.WriteLine(false);

При перегрузке операторов надо учитывать, что не все операторы можно перегрузить. В частности, мы можем перегрузить следующие операторы:

унарные операторы +, -, !, ~, ++, --

бинарные операторы +, -, \*, /, %

операции сравнения ==, !=, <, >, <=, >=

логические операторы &&, ||

Как уже отмечалось ранее, в теле класса могут быть объявлены:

константы;

поля;

конструкторы и деструкторы;

методы;

события;

делегаты;

вложенные классы (структуры, интерфейсы, перечисления). Методы, называемые **свойствами (***Properties***)**, представляют

специальную синтаксическую конструкцию, предназначенную для обеспечения эффективной работы клиентов класса с его полями. При работе со свойствами объекта (полями) нужно решить, какую стратегию доступа использовать, чтобы обеспечить требуемый режим к полю класса со стороны клиентов класса. Наиболее часто используют следующие стратегии:

чтение, запись (*Read, Write*);

чтение, запись при первом обращении (*Read, Write-once*);

только чтение (*Read-only*);

только запись (*Write-only*);

ни чтения, ни записи (*Not Read, Not Write*).

Чаще всего, поля класса объявляют закрытыми, а обращение к ним организуют через свойства.

Рассмотрим класс *Pers*, у которого пять полей: *fam, status, zrpl, vozras*,

*zdor*, характеризующих соответственно фамилию, статус, зарплату, возраст издоровье человека. Для каждого из этих полей может быть использована своя стратегия доступа. Например:

1. возраст доступен для чтения и записи;
2. фамилию можно задать только один раз;
3. статус можно только читать;
4. зарплата недоступна для чтения;
5. показатель здоровья закрыт для доступа со стороны клиентов класса, и только специальные методы класса могут сообщать некоторую информацию о здоровье персоны.

При обращении к свойству со стороны клиента класса, набор

операторов, входящий в тело аксессора *set* будет выполняться при вводе данных, а набор операторов, входящий в тело аксессора *get* будет выполняться при чтении данных, при этом вводимые данные имеют имя *value*.

Примеры возможной реализации указанных стратегий доступа к полям класса, приведены в примере:

*public class Pers {*

*//поля (все закрыты)*

*string fam="", status="", zdor ="";*

*int vozrast =0, zrpl =0;*

*//методы - свойства*

*//стратегия: Read,Write-once (Чтение, запись при первом обращении)*

*public string Fam {*

*set {if (fam == "") fam = value;}*

*get {return(fam);}*

*}*

*//стратегия: Read-only(Только чтение)*

*public string Status {*

*get {return(status);}*

*}*

*//стратегия: Read,Write (Чтение, запись)*

*public int Vozrast {*

*set {*

*vozrast = value;*

*if(vozrast < 7) status ="ребенок";*

*else if(vozrast <17) status ="школьник";*

*else if (vozrast < 22) status = "студент";*

*else status = "служащий";*

*}**get {return(vozrast );}*

*}*

*//стратегия: Write-only (Только запись)*

*public int Zrpl {*

*set { zrpl = value;}*

*}*

*}*

*public void TestPers(){*

*Pers pers1 = new Pers();*

*pers1.Fam = "Петров";*

*pers1.Vozrast = 21;*

*pers1.Zrpl = 1000;*

*Console.WriteLine ("Фам={0}, возраст={1}, статус={2}", pers1.Fam, pers1.Vozrast, pers1.Status);*

*pers1.Fam = "Иванов"; pers1.Vozrast += 1;*

*Console.WriteLine ("Фам={0}, возраст={1}, статус={2}",pers1.Fam, pers1.Vozrast, pers1.Status);*

*}*

Свойства являются частным случаем метода класса, обладающим особым синтаксисом. Еще одним частным случаем является **индексатор**. Метод-индексатор является обобщением метода-свойства. Он обеспечивает доступ к закрытому полю, представляющему массив. Объекты класса индексируются по этому полю.

Синтаксически объявление индексатора - такое же, как и в случае свойств, но аксессоры *get* и *set* приобретают аргументы по числу размерности массива, задающего индексы элемента, значение которого читается или обновляется. Класс может иметь не более **одного индексатора,** имя индексатора известно заранее, это - *this*. В том случае, если класс использует несколько массивов, то индексация объектов может быть выполнена только по одному из них.

Добавим в класс *Pers* свойство *children*, задающее детей персоны, сделаем это свойство закрытым, а доступ к нему обеспечит **индексатор**:

*…..*

*const int Child\_Max = 10;* *//максимальное число детей*

*Pers[] children = new Pers[Child\_Max];*

*int count\_children=0;* *//число детей*

*public Pers this[int i] {* *//индексатор*

*get {*

*if (i>=0 && i< count\_children) return(children[i]); else return(children[0]);*

*}**set {*

*if (i==count\_children && i< Child\_Max) { children[i] = value; count\_children++; }*

*}}*

*public void TestPersChildren(){*

*Pers pers1 = new Pers(),*

*pers2 = new Pers();*

*pers1.Fam = "Петров";*

*pers1.Vozrast = 42;*

*pers1.Zrpl = 10000;*

*pers1[pers1.Count\_children] = pers2;*

*pers2.Fam ="Петров";*

*pers2.Vozrast = 21;*

*pers2.Zrpl = 1000;*

*Pers pers3= new Pers("Петрова");*

*pers1[pers1.Count\_children] = pers3;*

*pers3.Fam ="Петрова"; pers3. Vozrast = 5;*

*Console.WriteLine ("Фам={0}, возраст={1}, статус={2}",pers1.Fam, pers1.Vozrast , pers1.Status);*

*Console.WriteLine ("Сын={0}, возраст={1}, статус={2}",pers1[0].Fam, pers1[0].Vozrast , pers1[0].Status);*

*Console.WriteLine ("Дочь={0}, возраст={1}, статус={2}",pers1[1].Fam, pers1[1].Vozrast , pers1[1].Status);*

*}*

Когда конструктор класса создает новый объект, то в памяти создается структура данных с полями, определяемыми классом. Но не все поля отражаются в структуре объекта. У класса могут быть поля, связанные не с объектами, а с самим классом. Эти поля объявляются как **статические** с модификатором *static*. **Статические поля** доступны всем методам класса. Независимо от того, какой объект класса обратился к статическому полю, организуется обращение к одному и тому же значению, определяемому статическим полем. Например, у класса *Pers* может быть **статическое поле** *message*, в котором каждый объект данного класса может оставитьсообщение для других объектов класса.

Аналогично статическим полям, у класса могут быть и **статические** **методы**, объявленные с модификатором*static*. Такие методы не используютинформацию о **свойствах конкретных объектов класса** - они обрабатывают общую для класса информацию, хранящуюся в его статических полях. Например, в классе *Pers* может быть **статический метод**, обрабатывающий данные из **статического поля** *message*.

Вызов **статических полей** и **методов класса** его клиентами имеет свои особенности.

Так при обращении к обычному открытому полю/методу, необходимо создать объект класса, а затем, указав имя объекта, через точку указать имя поля/метода.

Пример такого обращения приведен ниже.

*Student st=new Snudent();*

*st.fio=”Сидоров Иван Петрович”;*

*st.status();*

Для вызова открытого статического поля/метода необходимо указать имя класса и через точку имя поля/метода. Таким образом, обращение к открытым статическим полям/методам не требует создания объект данного класса.

Пример такого обращения, с использованием стандартной библиотеки классов приведен ниже.

*Console.WriteLine(“Привет”);*

*Double r=Math.Pi;*

Здесь мы не создавали объект класса *Console*, а сразу обратились к его статическому методу *WriteLine*, во второй строке примера мы не создавали объект класса *Math*, а сразу обратились к его статическому полю *Pi*.

**ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОГО ВЫПОЛНЕНИЯ**

Разработать согласно заданию СТРУКТУРЫ. Предусмотреть конструктор по умолчанию и с параметрами. Создать методы, работающие с полями структуры. Предусмотреть наличие методов-свойств, статических полей и методов. Продемонстрировать работу объектов структуры. Организовать перегрузку методов и операторов.

Описания членов-данных пользовательских классов:

1. СТУДЕНТ: ФИО, курс, пол, оценки. Решить задачу определения студентов, которые будут получать стипендию, вычислить размер стипендии. Учесть, что стипендия начисляется в условных единицах (РУМЗП) и в момент расчета учитывается текущее значение РУМЗП. Величина стипендии

зависит от полученных оценок, но ряд студентов получают ее вне зависимости от оценок, например староста и студенты- сироты. В программе рассмотреть случай группы из 10 студентов.

2 СЛУЖАЩИЙ: имя, возраст, рабочий стаж, должности. Решить задачу начисления заработной платы (оклад, пропорционально количеству отработанных в этом месяце дней, премия1, премия2, вычеты, итого на руки). В программе рассмотреть случай, для двух отделов, в которых работают по 5 человек. Процент премии 2 (от суммы начисленной з/пл) одинаков по всей организации.

3 КАДРЫ: ФИО, номер цеха, разряд, специальность, должность. В зависимости от стажа работы на предприятии, каждому сотруднику выплачивается доплата (<5 лет – 0 руб., 5-10 лет *x*1 рублей, 10-15 лет – *x*2 рублей, больше 15 лет *x*3 рублей). Эти значения в разные месяцы могут меняться. Если объявляется рабочему выговор, то доплата не производится. В программе рассмотреть случай 10 рабочих имеющих различный стаж работы. Сформировать ведомости на выдачу доплаты за 3 месяца.

* 1. ИЗДЕЛИЕ: группа 1, группа 2, название, шифр, вес, комплектация, стоимость. Стоимость изделия задается в условных единицах и в момент продажи переводится в рубли по текущему курсу. Группа 1 может быть: малоценка, если стоимость меньше 10 у.е., средней цены, если стоимость от 10 до 1000 у.е., повышенной стоимости (от 1000 до 10000 у.е.), значительной стоимости (>10000 у.е.). Группа 2 может быть: легкое (до 500 грамм), средней тяжести (от 500 грамм до 5 кг), значительной тяжести (от 5 кг до 500 кг), очень тяжелые (> 500 кг). При продаже изделия, в сертификате указыва-ется название страны - получателя. Программа должна обеспечить формиро-вание в виде текста сертификата изделия с указанием всех его параметров и страны назначения. Рассмотреть случай, когда продают по 5-6 изделий в 3 страны.
  2. ПЕЧАТНОЕ ИЗДАНИЕ: название, ФИО автора, стоимость, оглавление.

Стоимость книги со временем меняется.

Для художественной литературы: первые 5 лет она равна стоимости покупки книги, следующие 5 лет уменьшается на 20% в год, пока не станет равной нулю, если возраст книги превышает 100 лет, то ее стоимость в k раз превышает первоначальную стоимость.

Для прочих книг: первые 5 лет она равна стоимости покупки книги, следующие 5 лет уменьшается на 20% в год, пока не станет равной нулю.

Решить задачу по учету 10 книг, выдаваемых различным читателям, возможна в случае порчи замена книги на книгу с не меньшей стоимостью.

Программа должна: показать не выданные книги; Показать все книги данного автора; Дать информацию о замене книг.

6. ЭКЗАМЕН: ФИО студента, дата, оценка, перечень вопросов.

Создать класс «Билет», в котором указать 2 вопроса и задачу, а также зафиксировать ФИО студента и результаты его ответов. Организовать прове-дение экзамена:

- выбор студентам одного из 10 билетов, за каждый вопрос студент по-лучает некую оценку;

- если оценка за один пункт билета равна «неуд», то задается дополни-тельный вопрос (преподаватель имеет 3 любимых вопроса);

* если оценки за два пункта билета «неуд», студент получает общую оценку «неуд» и ему назначается дата пересдачи.

Программа должна выдать на экран результаты экзамена (5 студентов) в кратком виде (только оценки) и в полном виде (оценки за каждый вопрос билета + дополнительны = общая оценка).

7. АДРЕС: город, улица, номер дома, номер квартиры, № телефона, список жильцов.

Написать программу, которая использует объекты класса «адрес». Программа формирует адреса 10 человек и по запросу пользователя выдает на экран:

* по заданной фамилии, имени, отчеству - адрес;
* по заданному адресу (город, улица, дом) – ФИО;
* по номеру телефона – адрес и ФИО;

Учесть то, что названия улиц могут быть изменены.

1. ТОВАР: название, артикул, стоимость, даты (изготовление, срок реализации).

Написать программу, использующую объекты класса «товар». Учесть, что стоимость товара задается в условных единицах и только в момент продажи умножается на соответствующее значение. Кроме того, покупателю может быть предоставлена скидка (0-50%). Программа должна предоставлять пользователю информацию о 10 товарах, пользователь оформляет покупку выбранного товара. Если сумма покупки превышает некоторое значение, автоматически формируется некая скидка. Продавец по своему желанию может указать свое значение скидки.

Программа должна формировать список закупленных товаров с указанием покупателя, количества и вида товара, суммы и предоставленной скидки.

1. ЦЕХ: название, начальник, кол-во рабочих, перечень номенклатуры выпускаемых изделий.

В зависимости от разряда рабочего меняется его оплата (1-й разряд – 100% стоимости выполненной работы, 2-й 110%, 3-й 120, 4-й 130%, 5-й 150%). Эти значения в разные месяцы могут меняться. Если объявляется рабочему выговор, то доплата не производится. Рабочий, не имеющий выговора, может получить премию, которая задается для всех одинаково в рублях. В программе рассмотреть случай 10 рабочих имеющих различный стаж работы. Каждый рабочий может изготавливать 5 различных изделий. За каждое изделие он получает некоторую сумму. Рассчитать зарплату 10 рабочим.

1. АВТОМОБИЛЬ: марка, цвет, мощность двигателя, расход горючего на 100 км, стоимость, виды и даты ремонтов.

Написать программу, позволяющую вести набор данных о состоянии 10 автомобилей. Дается некий список видов ремонтов (не менее 5) с указанием стоимости ремонта и его длительности. Программа должна фиксировать ве-личину пробега каждого автомобиля (каждый день), проведенные ремонты. Рассчитать стоимость эксплуатации каждого автомобиля в течение 10 дней. Учесть, что стоимость горючего для всех автомобилей одинакова.

1. ОБЛАСТЬ: страна, название области, областной центр, форма прав-ления, площадь, население, список районов, основные отрасли экономики. Написать программу, которая поддерживает справочник областей страны и позволяет произвести несколько (не менее 3-х) видов поиска. Учесть, что данные меняются от года в год. Справочник содержит информацию о странах за 10 лет.
2. КОРАБЛЬ: название, год и место постройки, водоизмещение, тип, вид корабля, порт приписки, грузоподъемность, количество мест для пасса-жиров (1, 2, 3 го классов), расход топлива в день при: экономичном ходе, максимально ходе; запас топлива.

Создать класс «корабль».

Написать программу- справочник кораблей, содержащую данные о 10 кораблях. Произвести расчет стоимости перехода на указанное расстояние, для указанной даты, для всех кораблей. Учесть, что стоимость топлива для всех кораблей одинакова.

13.СТУДЕНТ: ФИО, курс, пол, группа, специальность. Решить задачу определения числа студентов, которые: а) обучаются в заданной оператором группе; б) изучают заданную оператором специальность. Сформировать массив объектов класса Студент. Использовать статические поля. Статический метод должен выдавать название вуза.

14.СПИСОК ЖИЛЬЦОВ: ФИО, № квартиры, улица, № дома. Решить задачу нахождения числа жителей: а) заданной квартиры; б) заданного дома; в) города. Использовать статические поля. Статический метод должен выдавать название страны.

15.АВТОМОБИЛЬ: марка, цвет, мощность двигателя, расход горючего на 100 км, год изготовления, стоимость.

Написать программу, позволяющую вести набор данных о 10 автомоби-лях. Рассчитать стоимость автомобиля. Учесть, что эта стоимость состоит из стоимости, заданной владельцем автомобиля и налога, который зависит от даты изготовления автомобиля. Например, если возраст автомобиля >10 лет, то это 20% его стоимости, если <=10 лет, то это 10% его стоимости. Статиче-ский метод должен выдавать название страны, в которой производится рас-таможивание автомобиля.

КОНТРОЛЬНЫЕ ВОПРОСЫ:

1. Методы-свойства класса. Назначение и описание.
2. Стратегии, реализуемые свойствами.
3. Роль аксессоров *set* и *get*.
4. Назначение, описание и вызов статических методов.
5. Описание и вызов статических переменных.
6. Индексаторы. Назначение и описание.
7. В тексте программы присутствуют следующие строки:

*int k;*

*k=classA.M1(100);*

*Itrw h=new Itrw();*

*h.tt=k;*

*if (h.tt!=k)*

*Console.WrineLine(“Error!”);*

Прокомментируйте их.