Лабораторная работа № 2

Конструкторы. Полиморфизм и наследование.

Цель лабораторной работы: познакомиться с созданием конструкторов для классов. Изучить механизмы наследования и полиморфизма. Познакомиться с управляющими операторами языка C#

Теоретическая часть

Конструктор – специальный метод объекта, решающий задачу начальной инициализации полей объекта и объявленный следующим образом:

- для этого метода всегда используется модификатор доступа **public**,
- нет типа возвращаемого значения (нет даже **void**),
- имя метода совпадает с именем класса.

Пример объявления конструктора в классе MyClass:

```
public class MyClass
{
    int a;

//Конструктор без параметров.
Инициализирует поле a нулем
    public MyClass(){
        a=0;
    }

    //Конструктор с параметром типа int.
public MyClass(char a){this.a=a;}
//Конструктор с параметром типа char.
    public MyClass(int a){
        this.a=a;
    }
}
```

Реализация для одного класса нескольких конструкторов является примером полиморфизма. **Полиморфизм** – механизм, позволяющий использовать одно имя для реализации схожих, но технически разных задач. Целью полиморфизма, применительно к объектно-ориентированному программированию, является использование одного имени для задания

общих для класса действий. В более общем смысле, в основе полиморфизма лежит идея «использовать один интерфейс для множества методов». Для компилятора полиморфные функции должны различаться принимаемыми параметрами. Это различие может быть по их количеству или по их типам.

Наследование – это процесс, посредством которого один объект может наследовать основные свойства другого объекта и добавлять к ним черты, характерные только для него. Наследование является важным, поскольку оно позволяет поддерживать концепцию иерархии классов - **hierarchical classification**. Применение иерархии классов делает управляемыми большие потоки информации. Без использования иерархии классов, для каждого объекта пришлось бы задать все характеристики, которые бы исчерпывающе его определяли. Однако, при использовании наследования можно описать объект путем определения того общего класса (или классов), к которому он относится, но со специальными чертами, делающие объект уникальным.

Синтаксис наследования следующий: при описании класса-потомка его класс-предок указывается через двоеточие.

Пример определения класса-предка Dad и класса-потомка Son:

```
public class Dad {}
public class Son: Dad {}
```

При инициализации полей объектов класса-наследника необходимо также инициализировать и поля базового класса. Инициализация полей обычно осуществляется с использованием конструктора. Передача управления конструктору базового класса при создании объекта – представителя производного класса осуществляется посредством конструкции

```
...(...):base(...){...},
```

которая располагается в объявлении конструктора класса-наследника между заголовком конструктора и телом. После ключевого слова **base** в скобках располагается список значений параметров конструктора базового класса. Очевидно, что выбор соответствующего конструктора определяется типом значений в списке (возможно, пустом) параметров.

Пример:

```
public class Dad
{
    int a;
    public Dad(int s);
}
public class Son: Dad
{
    public Son(int k):base(k) {}
}
```

Если же у базового класса не объявлено ни одного конструктора (оставлен конструктор по умолчанию) или объявлен конструктор без параметров, тогда конструкцию **base** можно не использовать: при ее отсутствии управление передается конструктору без параметров.

Однако, при вызове конструктора можно передавать управление не только конструктору базового класса, но и другому конструктору данного класса. Это удобно в тех случаях, когда необходимо создать множество объектов, различающихся между собой каким-либо образом, но и имеющим некую общую часть. Тогда для реализации общей части можно написать какой-то общий конструктор, а уже в других конструкторах, выполняющих более детальную настройку объекта, вызывать общий. Передача управления собственному конструктору аналогична описанной выше, только вместо ключевого слова **base** используется ключевое слово **this**.

Пример:

```
public class Dad
{
    int a;
    public Dad(int s);
}
public class Son: Dad
{
    public Son(int k):base(k) {} public
Son():this(10) {}
}
```

В рамках данной лабораторной работы согласно варианту предложено познакомиться с одним из управляющих операторов языка C#. Управляющие операторы применяются в рамках методов. Они определяют последовательность выполнения операторов в программе и являются основным средством реализации алгоритмов.

В данной лабораторной работе необходимо познакомиться со следующими категориями управляющих операторов:

1. **Условные операторы**. Вводятся ключевыми словами if, if ... else ..., switch.
2. **Циклы**. Вводятся ключевыми словами while, do ... while, for, foreach.

Условный оператор if имеет следующие правила использования. После ключевого слова if располагается взятое в круглые скобки условное выражение (булево выражение), следом за которым располагается оператор (блок операторов) произвольной сложности. Далее в операторе if ... else ... после ключевого слова else размещается еще один оператор.

Невозможно построить оператор if ... else ... на основе одиночного оператора объявления:

```
if (true) int X = 12;
if (true) int X = 12; else int Z = 1;
```

Такие конструкции ошибочные, так как одиночный оператор в C# – это не блок, и ставить в зависимость от условия (пусть даже всегда истинного) создание объекта нельзя.

Но в блоках операций определена своя область видимости, и создаваемые в них объекты, никому не мешая, существуют по своим собственным правилам.

```
if (true) {int X = 12;}
if (true) {int X = 12;} else {int Z = 0;}
```

Это правило действует во всех случаях, где какой-то оператор выполняется в зависимости от условия.

Оператор switch имеет вид:

```
switch(<проверяемая_переменная>)
{
    case <константа_1_значение_переменной>:
<оператор>; break; ….
    case <константа_n_значение_переменной>:
<оператор>; break;
    default: < оператор>break;
}
```

Пример:

```
int val;
switch (val)
{
    case 0: Console.WriteLine(0);break;
    case 1: Console.WriteLine(1);break;
    default: Console.WriteLine(«Число
неизвестно»);break;
}
```

Так как **case**-блоки строятся на основе одиночного оператора, то объявлять переменные в них нельзя. Ключевое слово **break** управляет выходом из **switch**. В следующем примере, если значение переменной **val** будет равно **0**, то на экран выведется два сообщения: **0** и **1**.

```
int val; switch (val)
{
    case 0: Console.WriteLine(0); // нет break;
case 1: Console.WriteLine(1);break;
    default: Console.WriteLine(«Число
неизвестно»);break;
}
```

Цикл **while** – это цикл с предусловием. Имеет следующий синтаксис:

```
while (УсловиеПродолжения) Оператор
```

Пример:

```
int i=0;
while(i<=10) i++;
```

Работает по следующему правилу: сначала проверяется условие продолжения оператора и в случае, если значение условного выражения равно **true**, соответствующий оператор (блок операторов) выполняется.

Цикл **do ... while** – цикл с постусловием. Синтаксис:

```
do Оператор while (УсловиеПродолжения)
```

Пример:

```
int i=0; do i++;
while(i<=10) ;
```

Разница с ранее рассмотренным оператором цикла состоит в том, что здесь сначала выполняется оператор (блок операторов), а затем проверяется условие продолжения оператора.

Цикл **for** – пошаговый цикл. Имеет синтаксис:

```
for([Выражение_Инициализации];[Условие_Продолжения];[Выражение_Шага])
```

Выражение_Инициализации, Условие_Продолжения, Выражение_Шага в могут быть пустыми, но наличие пары символов ';' в заголовке цикла обязательно.

Пример:

```
for(int i=0;i<10;i++)  Console.WriteLine(i);
```

Переменные, объявленные в операторе инициализации данного цикла, не могут быть использованы непосредственно после оператора до конца блока, содержащего этот оператор.

Практическая часть

В рамках консольного приложения разработать класс B-наследник класса A(из лабораторной работы №1) с полем d и свойством c2. Свойство c2 – результат вычисления выражения над полями a, b, d.

В теле свойства использовать управляющий оператор (см. вариант в таблице 2). У класса A создать конструктор, инициализирующий его поля. Для класса B определить 2 конструктора: один – наследуется от конструктора класса A, второй – собственный.

В теле программы создать объекты классов A и B, продемонстрировав работу всех конструкторов. Вывести значения свойства на экран.

Таблица 1. Варианты заданий

Вариант	Управляющий оператор
1	if
2	switch
3	for
4	while
5	do while
6	if
7	switch
8	for
9	while
10	do while
11	if