## Практическая работа 0. ООП

**Конструкторы класса**

Каждый раз, когда создается экземпляр класса или структура, вызывается конструктор. Класс или структура может иметь несколько конструкторов, принимающих различные аргументы. Конструкторы позволяют программисту задавать значения по умолчанию и писать код, который является гибким и удобным для чтения.

Например:

class CoOrds

{

public int x, y;

// конструктор

public CoOrds()

{

x = 0;

y = 0;

}

// конструктор с двумя аргументами

public CoOrds(int x, int y)

{

this.x = x;

this.y = y;

}

}

static void Main()

{

CoOrds p1 = new CoOrds(); // определение экземпляра класса

CoOrds p2 = new CoOrds(5, 3);

// вывод результатов

Console.WriteLine("CoOrds #1 at {0}", p1);

Console.WriteLine("CoOrds #2 at {0}", p2);

Console.ReadKey();

}

**ЗАДАНИЕ**

**#0A.**

Создайте класс Angle, включающий следующие три поля:

* типа int для числа градусов,
* типа float для числа минут
* типа char для указания направления (N, S, E или W).

Создайте конструктор класса по умолчанию (без параметров) со значениями 0, 0, ‘S’.

Создайте конструктор класса, который принимает три аргумента.

Создайте метод, позволяющий ввести координату точки, направление, в котором она измеряется.

Создайте метод, выводящий на экран значение этой координаты.

В функции main() определите экземпляр (объект) класса вызовом конструктора с параметрами и выведите значение экземпляра на экран.

В функции main() определите экземпляр (объект) класса вызовом конструктора без параметров, вызовите метод, запрашивающий значение координаты; вызовите метод для отображения введенного значения на экране.

**#0B.**

Постройте UML-диаграмму классов и создайте проект, в котором содержатся (рис.1):

1. Абстрактный класс с именем HotDrink, у которого имеются

* методы типа string Drink() (выпить), AddMilk() (добавить молоко), AddSugar() (добавить сахар);
* защищенные поля sugar, milk (значения по умолчанию);
* свойства для этих полей Milk и Sugar.

2. Интерфейс с именем ICup, у которого имеются

* методы Refill() (наполнить повторно), Wash() (вымыть),
* свойства Type (тип) и Capacity (Объем).

3. Класс с именем CupOfCoffee, который является наследником класса HotDrink, поддерживает интерфейс ICup и обладает дополнительным свойством BeanType (тип зерен).

4. Класс с именем CupOfTea, который является производным от класса HotDrink, поддерживает интерфейс ICup и обладает дополнительным свойством LeafType (тип чая).

5. Класс с функцией Main(), а также с функцией ProcessCup, которая получает в качестве параметра объект одного из классов CupOfCoffee или CupOfTea, по выбору пользователя (чай или кофе). Далее функция должна вызывать все методы и свойства абстрактного класса и интерфейса для любого объекта, переданного ей в качестве параметра (т.е. в качестве формального параметра используется абстрактный тип).

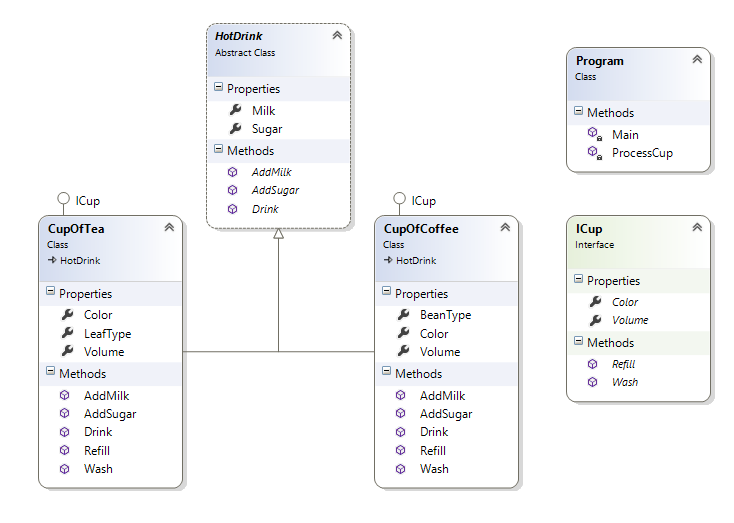


Рис.1. Диаграмма классов проекта

Пример вывода на консоль:

* красным обозначен ввод значений с клавиатуры;
* синим обозначен вывод работы переопределенных абстрактных методов и свойств;
* зеленым обозначен вывод работы реализованных методов и свойств интерфейса.

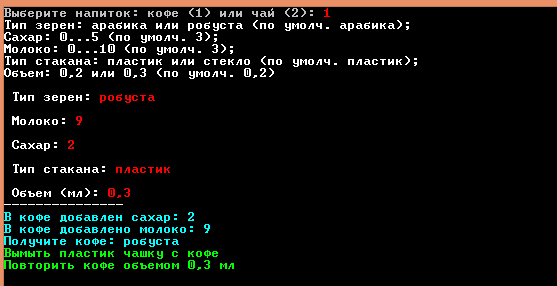


Рис.2. Пример вывода на консоль

**#0C.** Модифицируйте приведенную ниже структуру, включив в нее функцию, которая возвращает суммарную стоимость заказа:

struct orders

{

public string itemname; //наименование

public int unitCount; //число единиц

public double unitCost; //стоимость одной единицы

}

В Main создайте объект структуры, вызовите метод и выведите значение на экран.

**#0D.** Создайте приложение, которое использует структурированную обработку исключительных ситуаций для перевода переменной типа byte в переменную типа orientation безопасным путем. Эту часть требуется использовать в своем приложении:

enum orientation : byte

{

north = 1,

south = 2,

east = 3,

west = 4

}

В блоке try функции Main использовать следующий код:

mydirection = checked((orientation)myByte);