Практическая работа №1. Знакомство с базовыми типами и конструкциями в C#

Задание: в консольном проекте реализуйте вывод информации по типам.

Начальное меню, которое определяет основную функциональность приложения, выглядит следующим образом:

```
Информация по типам:
1 – Общая информация по типам
2 – Выбрать тип из списка
3 – Параметры консоли
0 - Выход из программы
```

Работа с консолью в C# реализуется с помощью статических методов класса `Console`. Основные операции с консолью:

- вывод строки на консоль: `Write` и `WriteLine`;
- ожидание ввода пользователя: `ReadKey` и `ReadLine`;
- очистка консоли: `Clear`.
- изменение цвета фона и текста: `BackgroundColor`, `ForegroundColor`.

В пункте №3 (параметры консоли) предоставьте пользователю возможность изменить цвет шрифта и фона. Для полной «перекраски» консоли после изменения свойств `BackgroundColor` и `ForegroundColor` необходимо вызвать метод `Clear`.

Чтение символа с консоли реализуется с помощью метода `Console.ReadKey`. Метод возвращает объект типа `ConsoleKeyInfo`, свойство `KeyChar` которого возвращает код символа типа `Char`.

Для организации выбора действия в зависимости от нажатой кнопки используйте конструкцию `switch`**.** В качестве параметра могут быть и символы, и строки.

```
while(true)
{
	switch(char.ToLower(Console.ReadKey(true).KeyChar))
	{
		case ‘1’ : ShowAllTypeInfo(); break;
		case ‘2’ : SelectType(); break;
		case ‘3’ : ChangeConsoleView(); break;
		default: break;
	}
}
```

Для каждого пункта меню необходимо создать отдельный метод. Если методы определяются в том же классе, что и метод `Main`, то они должны быть помечены как статические:

```
class Program {
	public static void ChangeConsoleView() { .. }
	public static void ShowAllTypeInfo() { .. }
	public static Type SelectType() { .. }
	public static void Main() { .. }
}
```

Методы одного и того же класса могут быть расположены в разных файлах. В этом случае объявление класса во всех файлах должно включать модификатор `partial` (частично-определенный класс).

В пункте №2 главного меню отображается информация для выбранного базового типа из списка:

```
Информация по типам
Выберите тип:
---------------------------------------
      1 – uint
      2 – int
      3 – long
      4 – float
      5 – double
      6 – char
      7 - string
      8 – Vector
      9 – Matrix
      0 – Выход в главное меню
```

Для работы с типами, которые определены в нашей сборке и подключенных сборках необходимо использовать элементы пространства имен `System.Reflection` (технология «Отражение»). Основной объект для работы с информацией о типах – `Type`.

В зависимости от выбора пользователя инициализируем экземпляр класса `Type`:

```
Type t = typeof(int)
```

Вывод информации о типе должен быть следующим (значения зависят от платформы .NET и могут различаться):

```
Информация по типу: System.Int32
      Значимый тип: +
      Пространство имен: System
      Сборка: mscorlib
      Общее число элементов: 19
      Число методов: 17
      Число свойств: 0
      Число полей: 2
      Список полей: MaxValue, MinValue
      Список свойств: -

Нажмите ‘M’ для вывода дополнительной информации по методам:
Нажмите ‘0’ для выхода в главное меню
```

Объект Type содержит множество булевых свойств, характеризующих тип: `IsValueType`, `IsAbstract`, `IsPointer`, `IsByRef`, `IsClass` и т.д.

Короткое имя сборки (без указания версии, региональных настроек) можно получить следующим образом:

```
string assemblyName = t.Assembly.GetName().Name;
```

Информацию об отдельных элементах типа (свойства, поля, методы) можно получить с помощью методов объекта `Type`: `GetMethods`, `GetFields`, `GetProperties`, `GetMembers`. Каждый из этих методов возвращает массив элементов соответствующего типа.

Например, чтобы узнать общее число полей используем свойство массива:

```
int nFields = t.GetFields().Length;
```

Для вывода имени отдельного поля используем:

```
string sField = t.GetFields()[i].Name;
```

Объединить имена полей можно с помощью метода `String.Join`:

```
// предварительно получаем имена полей
string[] fieldNames = ..
string sFieldNames = String.Join(“, “, fieldNames);
```

При нажатии на клавишу «M» предполагается вывод информации по методам, сгруппированным по именам:

```
Методы типа System.Int32
Название     Число перегрузок          Число параметров
ToString     4                         0..2
Parse        4                         1..3
CompareTo    2                         1
```

Для группировки методов можно воспользоваться *словарем*, который позволяет для одного уникального ключа (название метода) хранить одну агрегирующую характеристику (число методов с определенным названием) или набор характеристик в виде массива или структуры (число методов, минимальное количество параметров, максимальное количество параметров). Например, для подсчёта количества перегрузок:

```
var overloads = new Dictionary<string, int>();
foreach (var m in allMethods)
{
      if(overloads.ContainsKey(m.Name))
      // в словаре уже есть такое имя, обновляем статистику
            overloads[m.Name]++;
      else
      // в словаре нет такого имени, добавляем элемент
            overloads.Add(m.Name, 1);
}
```

В пункте №1 главного меню (*«Общая информация по типам»*) отображается следующий экран:

```
Общая информация по типам
Подключенные сборки: 17
Всего типов по всем подключенным сборкам: 26103
Ссылочные типы (только классы): 20601
Значимые типы: 4377

Информация в соответствии с вариантом №2
..

Нажмите любую клавишу, чтобы вернуться в главное меню
```

Для работы с типами используем следующие методы рефлексии. Получаем ссылку на исполняемую сборку (наш проект):

```
Assembly myAsm = Assembly.GetExecutingAssembly();
```

Получаем множество типов, определенных в нашей сборке:

```
Type[] thisAssemblyTypes = myAsm.GetTypes();
```

Но в нашей сборке типов не так много. Поэтому необходимо сформировать список типов, определенных во всех подключенных сборках:

```
Assembly[] refAssemblies = AppDomain.CurrentDomain.GetAssemblies();

// C# позволяет не дублировать название класса при создании объекта:
List<Type> types = new ();
foreach(Assembly asm in refAssemblies)
      types.AddRange(asm.GetTypes());
```

Далее, циклически обрабатывая список типов, опрашиваем их по ряду свойств:

```
foreach(var t in types) {
      ..
      if(t.IsClass)
            nRefTypes++;
      else if(t.IsValueType)
            nValueTypes++;
      ..
}
```

Задание по варианту

В пункте «*Общая информация по типам*» также необходимо вывести следующую информацию в зависимости от варианта:

V = (F + N) % 10, где F – первая буква фамилии в верхнем регистре на английском языке, N – первая буква имени в верхнем регистре на английском языке.

V	Общая информация по типам
0	Самое длинное название метода Метод с наибольшим числом аргументов
1	Самое длинное название поля Тип с наибольшим числом методов
2	Самое длинное название свойства Тип с наибольшим числом полей
3	Самое длинное название типа Метод с наибольшим числом аргументов целочисленного типа
4	Количество методов, не возвращающих значения Тип с наибольшим числом конструкторов
5	Количество свойств целочисленного типа Интерфейс с наибольшим числом методов
6	Количество методов без аргументов Тип с наибольшим числом конструкторов
7	Количество полей-констант Тип, реализующий наибольшее количество интерфейсов
8	Тип, у которого конструктор имеет наибольшее число аргументов Самое длинное название интерфейса
9	Самое короткое название метода Тип с наибольшим числом свойств булевского типа