

МИНОБРНАУКИ РОССИИ  
федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

---

Институт (факультет)    Институт информационных технологий

Кафедра                      Кафедра математического и программного  
обеспечения ЭВМ

---

**Задание на лабораторную работу №1**

**Дисциплина:** С#-программирование

<b>Темы:</b>	<u>Классы, ассоциация (композиция, агрегация); поля, свойства и методы; статические поля, свойства и методы; операторы и перегрузка операторов; обработка строк и работа с массивами</u>
--------------	--

**Среда разработки:** Microsoft Visual Studio 2022

**Язык программирования:** С# 9.0

**Тип проекта:** Библиотека классов

**ЗАДАНИЕ**

Разработать библиотеку классов для работы со штрихкодами.

Библиотека позволяет:

- Формировать штрихкод на основе текстовой информации;
- Выводить информацию по штрихкоду;
- Кодировать текстовую информацию по двум алгоритмам: числовому и текстовому;
- По желанию, формировать гибридный алгоритм кодирования.

**ТРЕБОВАНИЯ К РАЗРАБОТКЕ**

1. Запрещается использовать обработку исключительных ситуаций и генерировать исключения.
2. Каждый класс должен быть оформлен в отдельном файле.
3. **Придерживайтесь принципа DRY (Don't repeat yourself).**
4. Обязательно наличие комментариев и xml-комментариев.

Создать в **решении**<sup>1</sup> новую библиотеку классов для штрихкода и консольное приложение терминала для отладки всего функционала по штрихкоду и другого функционала в последующих лабораторных работах.

## **ЧАСТЬ 1**

Штрихкод описать в виде класса, без виртуальных методов:

- Класс содержит в себе информацию по исходному тексту для кодирования, коду («штрихкоду») в текстовом виде (см. часть 2, как формируется текстовый код) и типу вывода кода (см. приложение 1);
  - Информацию по тексту для кодирования можно изменять, следовательно необходимо обеспечить обновление и самого кода;
  - Информацию по коду изменять вне класса запрещено;
  - Способ вывода кода, задается один для всех объектов класса и может быть изменен в процессе работы программы;
- При создании объекта класса достаточно передать информацию о тексте для кодирования, код же должен сформироваться автоматически;
- Перегрузить функцию «**ToString()**» для получения информации по тексту и коду, учитывая способ вывода самого кода;
- Класс не должен содержать дополнительных открытых методов и полей («Класс-Модель<sup>2</sup>»).
- Логику формирования штрихкода (в части 2 и 3) вынести в «Класс-Сервис»<sup>3</sup>.

---

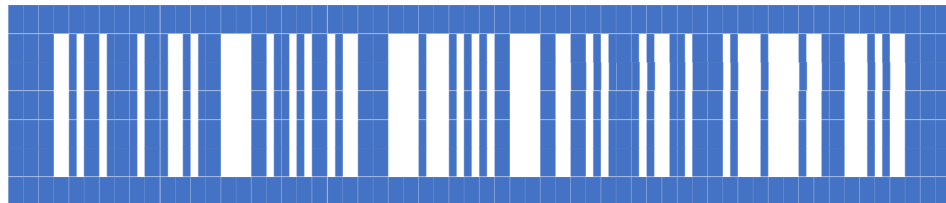
<sup>1</sup> Решение – это сборник нескольких проектов, его название должно соответствовать общей теме, не стоит использовать названия привязанные к одному проекту

<sup>2</sup> Задача класса в первую очередь хранить данные, бизнес-логика класса закрыта и в целом нужна только для формирования связей между данными в классе

<sup>3</sup> Задача класса в первую очередь обрабатывать данные, все необходимые данные поступают в качестве параметров функций или методов

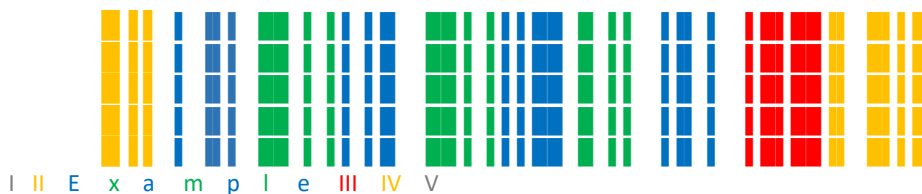
## ЧАСТЬ 2

Для начала рассмотрим структуру штрихкода подробнее, для примера возьмем готовый сформированный текстовый код из строки «Example»:



Example

Структура штрихкода:



Если обратить внимание, то каждый текстовый символ кодируется в набор из трех штрихов различной ширины (размер будем указывать в юнитах):

- - 1 юнит,
- - 2 юнита,
- - 3 юнита,
- - 4 юнита.

Аналогично и с пропусками. Конечная ширина кода каждого кодируемого символа будет составлять 11 юнитов: 3 штриха и три пробела.

Данный набор юнитов будем называть паттерном. Упорядоченный список паттернов представлен как массив **Patterns** в прил. 1.

Помимо закодированной текстовой информации штрихкод включает:

- I. Начало из 6 пустых юнитов.
- II. Паттерн стартового режима кодирования: текстовый режим или числовой режим.
- III. Контрольный паттерн.
- IV. Паттерн остановки.
- V. Конец из 6 пустых юнитов.

## Выбор начального паттерна кодирования II

Паттерны старта кодирования I + II представлены в прил. 1 как:

- **StartText**  
Режим кодирования посимвольно текстовой информации. Упорядоченный список доступных символов для кодирования представлен как массив **TextSymbols** в прил. 1.
- **StartNumbers**  
Режим кодирования пар чисел. Упорядоченный список доступных пар чисел для кодирования представлен как массив **NumberSymbols** в прил. 1.

Стоит обратить внимание, что можно переключать режим кодирования в процессе кодирования текстовой информации, для этого используются паттерны: **SwitchToText** и **SwitchToNumbers**.

Например, строку «123456Example» можно закодировать как:

1. **[StartText]** 1 2 3 4 5 6 E x a m p l e ... (Итого 14+ паттернов)
2. **[StartNumbers]** 12 34 56 **[SwitchToText]** E x a m p l e ... (итого 12+ паттернов)

Во втором случае штрих код будет короче на 2 паттерна или 22 юнита.

### Кодирование текста: Example

Позиция символа в массиве **TextSymbols** соответствует позиции паттерна в массиве **Patterns** см. прил. 1.

Аналогично позиция пары символов в массиве **NumberSymbols** соответствуют позиции паттерна в массиве **Patterns** см. прил. 1.

### Контрольный паттерн III:

На примере нашего штрих кода:



составим следующую таблицу:

Паттерн	II	E	x	a	m	p	l	e	III
Цена	1	1	2	3	4	5	6	7	2159 mod 103
Номер	104	37	88	65	77	80	76	69	99

## С#-программирование

где:

- Цена формируется порядковым номером паттерна, за исключением того, что стартовый паттерн имеет всегда цену 1.
- Номер соответствует номеру паттерна из массива **Patterns**.
- Номер контрольного паттерна рассчитывается, как остаток деления на 103 суммы произведений цены на номер каждого паттерна:  
 $1 \times 104 + \dots + 7 \times 69 = 2159 \bmod 103 = 99$
- Зная номер каждого паттерна, можно сформировать кодовую строку из значений массива **Patterns** по их позициям.

### Паттерн остановки **IV** + конец штрихкода **V**

В справочнике приведена константа: **Stop**

## ЧАСТЬ 3

Текст преобразовывается в строковый код («штрихкод») по следующему алгоритму (доп. см. приложение 1 с описанием используемых структур):

1. Формирование строковой промежуточной записи (строка из «0» и «1»), используя алгоритм из части 2.
2. Кодирование промежуточной записи в штрихи.
3. Формирование полного многострочного штрихкода.





### Этап 1

- Определяемся с текстовой информацией и выбираем алгоритм кодирования:
  - Если текстовая строка имеет четное количество числовых символов и не начинается с 0 выбираем алгоритм кодирования чисел.
  - Если текстовая информация содержит в себе последовательности чисел более чем 2 пары подряд, используем алгоритм переключения кодирования из числового в строковый и наоборот (алгоритм *использовать на свое усмотрение*).
  - Используем только символьный режим кодирования.
- Формируем на основе полученного списка паттернов строку из последовательности «0» и «1».

## С#-программирование



### Этап 2

- Преобразовываем строку из «0» и «1» в строку штрихов (см. **Bars**). Символ можно печатать, зажимая Alt и печатая на цифровой клавиатуре номера кода, отпуская Alt появится нужный символ.

Пара	Символ	Код
«00»	«  »	219
«01»	«  »	221
«10»	«  »	222
«11»	«  »	32

- Фиксируем длину полученной строки.

### Этап 3

- Используя длину строки, формируем полный штрихкод:
  - Сплошная строка из символов 
  - **Height** строк из этапа 2
  - Сплошная строка из символов 
- В конце каждой строки должен быть перенос строки.

### Этап вывода готовой информации:

- В зависимости от способа вывода могут быть варианты:
  - Только текст
  - Только код из Этапа 2
  - Все вместе, при этом обязательно нужно сделать так, чтобы текст был снизу штрихкода ровно на его середине.

## ЧАСТЬ 4

Создать в решении новый проект консольного приложения, подключить в зависимости библиотеку, проверить алгоритм кодирования:

- числовой информации;
- текстовой информации;
- гибридной информации.

## Примеры:

10



123456



Hello world!



10203 text 2



100000 10 20



102030405060 text !



## Используемые структуры данных:

```

/// <summary>
///     Формат сборки штрих-кода, должен быть в отдельном файле
/// </summary>
public enum BarcodeType
{
    /// <summary>
    ///     Текстовая информация
    /// </summary>
    Text,
    /// <summary>
    ///     Штрих-код
    /// </summary>
    Barcode,
    /// <summary>
    ///     Полная информация
    /// </summary>
    Full
}

/// <summary>
///     Допишите функцию <see cref="GetCode"/> класса <see cref="BarcodeHelper"/>
///     чтобы ее использовать при генерации строки штрихкода в вашем классе
/// <para>
///     <see cref="Sample"/> - название функции умышленно используется одно и то же,
///     не забудьте переименовать в правильные названия, включая названия параметров,
///     оценив код.
/// </para>
/// </summary>
public static class BarcodeHelper
{
    #region Help

    /// <summary>
    /// <para>
    ///
    /// 
    ///
    /// Example
    /// </para>
    /// </summary>
    public static string GetCode(string text)
    {
        // TODO из text собрать код, как в описании выше
        return text;
    }

    private static void Sample(string a, StringBuilder b, IList<int> c, ref bool
d, ref int e)
    {
        if ((d && !Sample(a, e, 2)) ||
            (!d && Sample(a, e, 4)))
        {
            d = !d;
            Sample(b, c, d ? SwitchToNumbers : SwitchToText);
        }
        Sample(ref e, c, d, a, b);
    }
}

```



## C#-программирование

```
private static void Sample(ref int a, IList<int> b, bool c, string d,
StringBuilder e)
{
    if (c)
    {
        Sample(e, b, Array.IndexOf(NumberSymbols, d.Substring(a, 2)));
        a += 2;
    }
    else
    {
        Sample(e, b, Array.IndexOf(TextSymbols, d.Substring(a, 1)));
        a++;
    }
}

private static void Sample(StringBuilder a, IList<int> b, int c)
{
    a.Append(Patterns[c]);
    b.Add(c);
}

private static bool Sample(string a, int b, int c)
{
    var chars = a.Skip(b).Take(c);
    return chars.Count() == c && chars.All(x => char.IsDigit(x));
}

private static int Sample(IList<int> a)
{
    var sum = a[0];

    for (var i = 1; i < a.Count; i++)
    {
        sum += i * a[i];
    }

    sum %= 103;

    return sum;
}

private static char Sample(string a) => Bars[Convert.ToInt32(a, 2)];

private static IEnumerable<string> Sample(this string a, int b)
{
    return Enumerable.Range(0, a.Length / b)
        .Select(i => a.Substring(i * b, b));
}

/// <summary>
///  Высота штрихкода (в строках)
/// </summary>
private const int Height = 4;

/// <summary>
///  Для получения рамки штрихкода по краям
/// </summary>
private const string Frame = "0000";

/// <summary>
///  Допустимые варианты штрихов
/// </summary>
public static readonly char[] Bars = { '■', '▬', '▯', ' ' };

/// <summary>
```

## С#-программирование

```
/// Начальный номер паттерна для текстовой строки
/// </summary>
private const int StartText = 104;

/// <summary>
/// Начальный номер паттерна для числовой строки
/// </summary>
private const int StartNumbers = 105;

/// <summary>
/// Переключить в числовой режим кодирования
/// </summary>
private const int SwitchToNumbers = 99;

/// <summary>
/// Переключить в текстовый режим кодирования
/// </summary>
private const int SwitchToText = 100;

/// <summary>
/// Номер паттерна завершения
/// </summary>
private const int Stop = 108;

/// <summary>
/// Доступные паттерны
/// </summary>
private static readonly string[] Patterns = {
    "11011001100", "11001101100", "11001100110", "10010011000", "10010001100",
    "10001001100", "10011001000", "10011000100", "10001100100", "11001001000",
    "11001000100", "11000100100", "10110011100", "10011011100", "10011001110",
    "10111001100", "10011101100", "10011100110", "11001110010", "11001011100",
    "11001001110", "11011100100", "11001110100", "11101101110", "11101001100",
    "11100101100", "11100100110", "11101100100", "11100110100", "11100110010",
    "11011011000", "11011000110", "11000110110", "10100011000", "10001011000",
    "10001000110", "10110001000", "10001101000", "10001100010", "11010001000",
    "11000101000", "11000100010", "10110111000", "10110001110", "10001101110",
    "10111011000", "10111000110", "10001110110", "11101110110", "11010001110",
    "11000101110", "11011101000", "11011100010", "11011100010", "11101011000",
    "11101000110", "11101101000", "11101100010", "10100110000", "10100001100",
    "10010110000", "10010000110", "10000101100", "10000100110", "10110010000",
    "10110000100", "10011010000", "10011000010", "10000110100", "10000110010",
    "11000010010", "11001010000", "11110111010", "11000010100", "10001111010",
    "10100111100", "10010111100", "10010011110", "10111100100", "10011110100",
    "10011110010", "11110100100", "11110010100", "11110010010", "11011011110",
    "11011110110", "11110110110", "10101111000", "10100011110", "10001011110",
    "10111101000", "10111100010", "11110101000", "11110100010", "10111011110",
    // 100+
    "10111101110", "11101011110", "11110101110", "11010000100", "11010010000",
    "11010011100", "11000111010", "11010111000", "1100011101011"};

/// <summary>
/// Разрешенные символы
/// </summary>
private static readonly string[] TextSymbols = {
    " ", "!", "\", "#", "$", "%", "&", "'", "(", ")",
    "*", "+", ",", "-", ".", "/", "0", "1", "2", "3",
    "4", "5", "6", "7", "8", "9", ":", ";", "<", "=",
    ">", "?", "@", "A", "B", "C", "D", "E", "F", "G",
    "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q",
    "R", "S", "T", "U", "V", "W", "X", "Y", "Z", "[",
    "\\", "]", "^", "_", "`", "a", "b", "c", "d", "e",
    "f", "g", "h", "i", "j", "k", "l", "m", "n", "o",
    "p", "q", "r", "s", "t", "u", "v", "w", "x", "y",
    "z", "{", "|", "}", "~"
};
```

## С#-программирование

```
/// <summary>
///     Разрешенные пары числовых строк
/// </summary>
private static readonly string[] NumberSymbols = {
    "00", "01", "02", "03", "04", "05", "06", "07", "08", "09",
    "10", "11", "12", "13", "14", "15", "16", "17", "18", "19",
    "20", "21", "22", "23", "24", "25", "26", "27", "28", "29",
    "30", "31", "32", "33", "34", "35", "36", "37", "38", "39",
    "40", "41", "42", "43", "44", "45", "46", "47", "48", "49",
    "50", "51", "52", "53", "54", "55", "56", "57", "58", "59",
    "60", "61", "62", "63", "64", "65", "66", "67", "68", "69",
    "70", "71", "72", "73", "74", "75", "76", "77", "78", "79",
    "80", "81", "82", "83", "84", "85", "86", "87", "88", "89",
    "90", "91", "92", "93", "94", "95", "96", "97", "98", "99"
};

#endregion
}
```