Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИУК «Информатика и управление»
КАФЕДРА ИУК2 «Информационные системы и сети»

Практическая работа 3.1
«Примеры стандартных решений в C#»

ДИСЦИПЛИНА: «Объектно-ориентированное программирование»

Выполнил: студент гр. ИУК4-21Б (Суриков Н.С)
(подпись) (Ф.И.О.)

Проверил: (Дерюгина Е. О.)
(подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга, 2024

Цель: сформировать навыки решения задач с применением ООП.

Листинг:

```
public class Program
{
    // 1. Переписать числа в обратном порядке
    static void ReverseNumbersInFile(string inputFileName, string
    outputFileName)
    {
        Stack<string> numbersStack = new Stack<string>();
        using (StreamReader sr = new StreamReader(inputFileName))
        {
            while (!sr.EndOfStream)
            {
                string line = sr.ReadLine();
                numbersStack.Push(line);
            }
        }
        using (StreamWriter sw = new StreamWriter(outputFileName))
        {
            while (numbersStack.Count > 0)
            {
                sw.WriteLine(numbersStack.Pop());
            }
        }
    }
    static void PrintReversedVowels(string fileName)
    {
        Stack<char> vowelsStack = new Stack<char>();
        using (StreamReader sr = new StreamReader(fileName))
        {
            while (!sr.EndOfStream)
            {
                char c = (char)sr.Read();
                if (IsVowel(c))
                {
                    vowelsStack.Push(c);
                }
            }
        }
        while (vowelsStack.Count > 0)
        {
            Console.Write(vowelsStack.Pop());
        }
        Console.WriteLine();
    }
    // 3. Напечатать литеры каждой строки в обратном порядке
    static void PrintCharactersInReverseOrder(string fileName)
    {
        using (StreamReader sr = new StreamReader(fileName))
        {
            while (!sr.EndOfStream)
            {
```

```
                string line = sr.ReadLine();
                char[] chars = line.ToCharArray();
                Array.Reverse(chars);
                Console.WriteLine(chars);
            }
        }
    }
    // Вспомогательная функция для определения гласных букв
    static bool IsVowel(char c)
    {
        return "aeiouAEIOU".IndexOf(c) != -1;
    }
    // 4. Проверка, является ли строка s2 обратной s1
    static bool IsReverseString(string s1, string s2)
    {
        if (s1.Length != s2.Length)
            return false;
        Stack<char> stack = new Stack<char>();
        foreach (char c in s1)
        {
            stack.Push(c);
        }
        foreach (char c in s2)
        {
            if (stack.Count == 0 || stack.Pop() != c)
                return false;
        }
        return true;
    }
    // 5. Подсчет выражения в префиксной форме
    static int EvaluatePrefixExpression(string expression)
    {
        Stack<int> stack = new Stack<int>();
        for (int i = expression.Length - 1; i >= 0; i--)
        {
            char c = expression[i];
            if (char.IsDigit(c))
            {
                stack.Push(c - '0');
            }
            else if (c == '+' || c == '-' || c == '*' || c == '/')
            {
                int operand1 = stack.Pop();
                int operand2 = stack.Pop();
                switch (c)
                {
                    case '+':
                        stack.Push(operand1 + operand2);
                        break;
                    case '-':
                        stack.Push(operand1 - operand2);
                        break;
                    case '*':
                        stack.Push(operand1 * operand2);
                        break;
```

```
                    case '/':
                        stack.Push(operand1 / operand2);
                        break;
                }
            }
        }
        return stack.Pop();
    }
    // 6. Преобразование выражения из инфиксной формы в префиксную
    static string ConvertInfixToPrefix(string infixExpression)
    {
        Stack<string> operatorStack = new Stack<string>();
        Stack<string> operandStack = new Stack<string>();
        string[] tokens = infixExpression.Split(' ');
        for (int i = tokens.Length - 1; i >= 0; i--)
        {
            string token = tokens[i];
            if (IsOperator(token))
            {
                while (operatorStack.Count > 0 &&
                Precedence(operatorStack.Peek()) >= Precedence(token))
                {
                    string operand1 = operandStack.Pop();
                    string operand2 = operandStack.Pop();
                    operandStack.Push(token + " " + operand1 + " " +
                    operand2);
                    operatorStack.Pop();
                }
                operatorStack.Push(token);
            }
            else
            {
                operandStack.Push(token);
            }
        }
        while (operatorStack.Count > 0)
        {
            string operand1 = operandStack.Pop();
            string operand2 = operandStack.Pop();
            operandStack.Push(operatorStack.Pop() + " " + operand1 + " "
+
            operand2);
        }
        return operandStack.Pop();
    }
    // Проверка является ли символ оператором
    static bool IsOperator(string token)
    {
        return token == "+" || token == "-" || token == "*" || token ==
"/";
    }
    // Определение приоритета оператора
    static int Precedence(string op)
    {
        switch (op)
```

```
159         {
160             case "+":
161             case "-":
162                 return 1;
163             case "*:":
164             case "/":
165                 return 2;
166             default:
167                 return 0;
168         }
169     }
170     // 7. Преобразование выражения из постфиксной формы в инфиксную
171     static string ConvertPostfixToInfix(string postfixExpression)
172     {
173         Stack<string> stack = new Stack<string>();
174         string[] tokens = postfixExpression.Split(' ');
175         foreach (string token in tokens)
176         {
177             if (IsOperator(token))
178             {
179                 string operand2 = stack.Pop();
180                 string operand1 = stack.Pop();
181                 stack.Push("(" + operand1 + " " + token + " " + operand2
+
182                 ")");
183             }
184             else
185             {
186                 stack.Push(token);
187             }
188         }
189         return stack.Pop();
190     }
191     // 8. Вычисление значения формулы из файла
192     static int EvaluateFormulaFromFile(string fileName)
193     {
194         Stack<char> operators = new Stack<char>();
195         Stack<int> values = new Stack<int>();
196         using (StreamReader sr = new StreamReader(fileName))
197         {
198             string formula = sr.ReadLine();
199             foreach (char c in formula)
200             {
201                 if (c == '(')
202                 {
203                     continue;
204                 }
205                 else if (char.IsDigit(c))
206                 {
207                     values.Push(int.Parse(c.ToString()));
208                 }
209                 else if (c == 'm' || c == 'M')
210                 {
211                     operators.Push(c);
212                 }
```

```
                else if (c == ')' && operators.Count > 0)
                {
                    int val1 = values.Pop();
                    int val2 = values.Pop();
                    int result;
                    if (operators.Pop() == 'M')
                    {
                        result = Math.Max(val1, val2);
                    }
                    else
                    {
                        result = Math.Min(val1, val2);
                    }
                    values.Push(result);
                }
            }
        }
        return values.Pop();
    }
    // 9. Преобразование формулы из файла по заданному правилу
    static int EvaluateFormulaWithRules(string fileName)
    {
        Stack<char> operators = new Stack<char>();
        Stack<int> values = new Stack<int>();
        using (StreamReader sr = new StreamReader(fileName))
        {
            string formula = sr.ReadLine();
            foreach (char c in formula)
            {
                if (c == '(')
                {
                    continue;
                }
                else if (char.IsDigit(c))
                {
                    values.Push(int.Parse(c.ToString()));
                }
                else if (c == 'm' || c == 'p')
                {
                    operators.Push(c);
                }
                else if (c == ')' && operators.Count > 0)
                {
                    int val1 = values.Pop();
                    int val2 = values.Pop();
                    int result;
                    if (operators.Pop() == 'm')
                    {
                        result = (val2 - val1) % 10;
                    }
                    else
                    {
                        result = (val2 + val1) % 10;
                    }
                    values.Push(result);
```

```
                }
            }
        }
        return values.Pop();
    }
    // 10. Преобразование текста с учетом символа "#"
    static string ProcessTextWithBackspace(string text)
    {
        Stack<char> stack = new Stack<char>();
        foreach (char c in text)
        {
            if (c != '#')
            {
                stack.Push(c);
            }
            else if (stack.Count > 0)
            {
                stack.Pop();
            }
        }
        char[] resultChars = stack.ToArray();
        Array.Reverse(resultChars);
        return new string(resultChars);
    }
    // 11. Печать элементов файла: сначала все символы, отличные от
цифр, затем все цифры
    static void PrintFileElements(string fileName)
    {
        Queue<char> nonDigitChars = new Queue<char>();
        Queue<char> digitChars = new Queue<char>();
        using (StreamReader sr = new StreamReader(fileName))
        {
            while (!sr.EndOfStream)
            {
                char c = (char)sr.Read();
                if (char.IsDigit(c))
                {
                    digitChars.Enqueue(c);
                }
                else
                {
                    nonDigitChars.Enqueue(c);
                }
            }
        }
        while (nonDigitChars.Count > 0)
        {
            Console.Write(nonDigitChars.Dequeue());
        }
        while (digitChars.Count > 0)
        {
            Console.Write(digitChars.Dequeue());
        }
        Console.WriteLine();
    }
```

```
322     // 12. Печать элементов файла: сначала числа из интервала [a,b],
затем числа меньше a, затем числа больше b
323     static void PrintFileElementsInRange(string fileName, int a, int b)
324     {
325         Queue<int> numbersInRange = new Queue<int>();
326         Queue<int> numbersLessThanA = new Queue<int>();
327         Queue<int> numbersGreaterThanB = new Queue<int>();
328         using (StreamReader sr = new StreamReader(fileName))
329         {
330             while (!sr.EndOfStream)
331             {
332                 int number = int.Parse(sr.ReadLine());
333                 if (number >= a && number <= b)
334                 {
335                     numbersInRange.Enqueue(number);
336                 }
337                 else if (number < a)
338                 {
339                     numbersLessThanA.Enqueue(number);
340                 }
341                 else
342                 {
343                     numbersGreaterThanB.Enqueue(number);
344                 }
345             }
346         }
347         while (numbersInRange.Count > 0)
348         {
349             Console.WriteLine(numbersInRange.Dequeue());
350         }
351         while (numbersLessThanA.Count > 0)
352         {
353             Console.WriteLine(numbersLessThanA.Dequeue());
354         }
355         while (numbersGreaterThanB.Count > 0)
356         {
357             Console.WriteLine(numbersGreaterThanB.Dequeue());
358         }
359     }
360     // 13. Печать слов из файла: сначала слова, начинающиеся на гласную,
затем на согласную
361     static void PrintWordsByFirstLetter(string fileName)
362     {
363         Queue<string> vowelsWords = new Queue<string>();
364         Queue<string> consonantsWords = new Queue<string>();
365         using (StreamReader sr = new StreamReader(fileName))
366         {
367             while (!sr.EndOfStream)
368             {
369                 string word = sr.ReadLine();
370                 if (!string.IsNullOrEmpty(word))
371                 {
372                     char firstChar = word[0];
373                     if (IsVowel(firstChar))
374                     {
```

```
                        vowelsWords.Enqueue(word);
                    }
                    else
                    {
                        consonantsWords.Enqueue(word);
                    }
                }
            }
        }
        while (vowelsWords.Count > 0)
        {
            Console.WriteLine(vowelsWords.Dequeue());
        }
        while (consonantsWords.Count > 0)
        {
            Console.WriteLine(consonantsWords.Dequeue());
        }
    }
    // 14. Печать чисел из файла: сначала положительные числа, затем
отрицательные числа
    static void PrintNumbersBySign(string fileName)
    {
        Queue<int> positiveNumbers = new Queue<int>();
        Queue<int> negativeNumbers = new Queue<int>();
        using (StreamReader sr = new StreamReader(fileName))
        {
            while (!sr.EndOfStream)
            {
                int number = int.Parse(sr.ReadLine());
                if (number >= 0)
                {
                    positiveNumbers.Enqueue(number);
                }
                else
                {
                    negativeNumbers.Enqueue(number);
                }
            }
        }
        while (positiveNumbers.Count > 0)
        {
            Console.WriteLine(positiveNumbers.Dequeue());
        }
        while (negativeNumbers.Count > 0)
        {
            Console.WriteLine(negativeNumbers.Dequeue());
        }
    }
    // 15. Печать слов из файла: сначала слова с прописной буквы, затем
со строчной
    static void PrintWordsByCase(string fileName)
    {
        Queue<string> capitalizedWords = new Queue<string>();
        Queue<string> lowercasedWords = new Queue<string>();
        using (StreamReader sr = new StreamReader(fileName))
```

```
428         {
429             while (!sr.EndOfStream)
430             {
431                 string word = sr.ReadLine();
432                 if (!string.IsNullOrEmpty(word))
433                 {
434                     if (char.IsUpper(word[0]))
435                     {
436                         capitalizedWords.Enqueue(word);
437                     }
438                     else
439                     {
440                         lowercasedWords.Enqueue(word);
441                     }
442                 }
443             }
444         }
445         while (capitalizedWords.Count > 0)
446         {
447             Console.WriteLine(capitalizedWords.Dequeue());
448         }
449         while (lowercasedWords.Count > 0)
450         {
451             Console.WriteLine(lowercasedWords.Dequeue());
452         }
453     }
454     // 16. Печать данных о сотрудниках: сначала данные о мужчинах, затем
о женщинах
455     static void PrintEmployeesByGender(string fileName)
456     {
457         Queue<string> maleEmployees = new Queue<string>();
458         Queue<string> femaleEmployees = new Queue<string>(); using
(StreamReader sr = new StreamReader(fileName))
459         {
460             while (!sr.EndOfStream)
461             {
462                 string employeeData = sr.ReadLine();
463                 if (!string.IsNullOrEmpty(employeeData))
464                 {
465                     string[] data = employeeData.Split(',');
466                     string gender = data[3].Trim();
467                     if (gender.ToLower() == "male")
468                     {
469                         maleEmployees.Enqueue(employeeData);
470                     }
471                     else if (gender.ToLower() == "female")
472                     {
473                         femaleEmployees.Enqueue(employeeData);
474                     }
475                 }
476             }
477         }
478         while (maleEmployees.Count > 0)
479         {
480             Console.WriteLine(maleEmployees.Dequeue());
```

```
        }
        while (femaleEmployees.Count > 0)
        {
            Console.WriteLine(femaleEmployees.Dequeue());
        }
    }
    // 17. Печать данных о сотрудниках: сначала данные о сотрудниках с зарплатой меньше 10000, затем остальные
    static void PrintEmployeesBySalary(string fileName)
    {
        Queue<string> lowSalaryEmployees = new Queue<string>();
        Queue<string> otherEmployees = new Queue<string>();
        using (StreamReader sr = new StreamReader(fileName))
        {
            while (!sr.EndOfStream)
            {
                string employeeData = sr.ReadLine();
                if (!string.IsNullOrEmpty(employeeData))
                {
                    string[] data = employeeData.Split(',');
                    int salary = int.Parse(data[5].Trim());
                    if (salary < 10000)
                    {
                        lowSalaryEmployees.Enqueue(employeeData);
                    }
                    else
                    {
                        otherEmployees.Enqueue(employeeData);
                    }
                }
            }
        }
        while (lowSalaryEmployees.Count > 0)
        {
            Console.WriteLine(lowSalaryEmployees.Dequeue());
        }
        while (otherEmployees.Count > 0)
        {
            Console.WriteLine(otherEmployees.Dequeue());
        }
    }
    // 18. Печать данных о сотрудниках: сначала данные о сотрудниках младше 30 лет, затем остальные
    static void PrintEmployeesByAge(string fileName)
    {
        Queue<string> youngEmployees = new Queue<string>();
        Queue<string> otherEmployees = new Queue<string>();
        using (StreamReader sr = new StreamReader(fileName))
        {
            while (!sr.EndOfStream)
            {
                string employeeData = sr.ReadLine();
                if (!string.IsNullOrEmpty(employeeData))
                {
                    string[] data = employeeData.Split(',');
```

```
                    int age = int.Parse(data[4].Trim());
                    if (age < 30)
                    {
                        youngEmployees.Enqueue(employeeData);
                    }
                    else
                    {
                        otherEmployees.Enqueue(employeeData);
                    }
                }
            }
        }
        while (youngEmployees.Count > 0)
        {
            Console.WriteLine(youngEmployees.Dequeue());
        }
        while (otherEmployees.Count > 0)
        {
            Console.WriteLine(otherEmployees.Dequeue());
        }
    }
    // 19. Печать данных о студентах: сначала данные о студентах,
успешно сдавших сессию, затем остальные
    static void PrintStudentsBySessionResult(string fileName)
    {
        Queue<string> passedStudents = new Queue<string>();
        Queue<string> otherStudents = new Queue<string>();
        using (StreamReader sr = new StreamReader(fileName))
        {
            while (!sr.EndOfStream)
            {
                string studentData = sr.ReadLine();
                if (!string.IsNullOrEmpty(studentData))
                {
                    string[] data = studentData.Split(',');
                    int mark1 = int.Parse(data[4].Trim());
                    int mark2 = int.Parse(data[5].Trim());
                    int mark3 = int.Parse(data[6].Trim());
                    if (mark1 >= 4 && mark2 >= 4 && mark3 >= 4)
                    {
                        passedStudents.Enqueue(studentData);
                    }
                    else
                    {
                        otherStudents.Enqueue(studentData);
                    }
                }
            }
        }
        while (passedStudents.Count > 0)
        {
            Console.WriteLine(passedStudents.Dequeue());
        }
        while (otherStudents.Count > 0)
        {
```

```
588             Console.WriteLine(otherStudents.Dequeue());
589         }
590     }
591     // 20. Печать данных о студентах: сначала данные о студентах,
успешно обучающихся на оценки 4 и 5, затем остальные
592     static void PrintStudentsByGoodMarks(string fileName)
593     {
594         Queue<string> goodStudents = new Queue<string>();
595         Queue<string> otherStudents = new Queue<string>();
596         using (StreamReader sr = new StreamReader(fileName))
597         {
598             while (!sr.EndOfStream)
599             {
600                 string studentData = sr.ReadLine();
601                 if (!string.IsNullOrEmpty(studentData))
602                 {
603                     string[] data = studentData.Split(',');
604                     int mark1 = int.Parse(data[4].Trim());
605                     int mark2 = int.Parse(data[5].Trim());
606                     int mark3 = int.Parse(data[6].Trim());
607                     if (mark1 == 4 || mark1 == 5 && mark2 == 4 || mark2
== 5
608                     && mark3 == 4 || mark3 == 5)
609                     {
610                         goodStudents.Enqueue(studentData);
611                     }
612                     else
613                     {
614                         otherStudents.Enqueue(studentData);
615                     }
616                 }
617             }
618         } while (goodStudents.Count > 0)
619         {
620             Console.WriteLine(goodStudents.Dequeue());
621         }
622         while (otherStudents.Count > 0)
623         {
624             Console.WriteLine(otherStudents.Dequeue());
625         }
626     }
627     static void Main(string[] args)
628     {
629         // Вызов функций для решения задач с использованием класса Stack
630         Console.WriteLine("Задачи с использованием класса Stack:");
631         Console.WriteLine();
632         // 1. Переписать в другой файл все числа в обратном порядке
633         ReverseNumbersInFile("input.txt", "output.txt");
634         // 2. Распечатать гласные буквы текстового файла в обратном
порядке
635         PrintReversedVowels("text.txt");
636         // 3. Напечатать содержимое текстового файла, выписывая литеры
каждой строки в обратном порядке
637         PrintCharactersInReverseOrder("text.txt");
638         // 4. Проверка строки s2 на обратность по отношению к s1
```

```
639         string s1 = "hello";
640         string s2 = "olleh";
641         bool isReverse = IsReverseString(s1, s2);
642         Console.WriteLine($"Строка s2 обратна строке s1: {isReverse}");
643         // 5. Вычисление выражения в префиксной форме
644         string prefixExpression = "+ 3 * 4 5";
645         int prefixResult = EvaluatePrefixExpression(prefixExpression);
646         Console.WriteLine($"Результат выражения в префиксной форме:
{prefixResult}");
647         // 6. Преобразование выражения из инфиксной формы в префиксную
648         string infixExpression = "3 + 4 * 5";
649         string prefixExpressionConverted =
ConvertInfixToPrefix(infixExpression);
650         Console.WriteLine($"Префиксная форма выражения:
{prefixExpressionConverted}");
651         // 7. Преобразование выражения из постфиксной формы в инфиксную
652         string postfixExpression = "3 4 5 * +";
653         string infixExpressionConverted =
654         ConvertPostfixToInfix(postfixExpression);
655         Console.WriteLine($"Инфиксная форма выражения:
{infixExpressionConverted}");
656         // 8. Вычисление значения формулы из текстового файла
657         int formulaValue = EvaluateFormulaFromFile("formula.txt");
658         Console.WriteLine($"Значение формулы: {formulaValue}");
659         // 9. Вычисление значения формулы с операциями m и p из
текстового файла
660         int formulaValue2 = EvaluateFormulaWithRules("formula2.txt");
661         Console.WriteLine($"Значение формулы с операциями m и p:
{formulaValue2}");
662         // 10. Преобразование текста с учетом символа "#"
663         string processedText = ProcessTextWithBackspace("abc#d##c");
664         Console.WriteLine($"Обработанный текст: {processedText}");
665         Console.WriteLine();
666         Console.WriteLine("Задачи с использованием класса Queue:");
667         Console.WriteLine();
668         // Вызов функций для решения задач с использованием класса Queue
669         // 11. Печать элементов файла: сначала все символы, отличные от
цифр, затем все цифры
670         PrintFileElements("text.txt");
671         // 12. Печать элементов файла: сначала числа из интервала [a,b],
затем числа меньше a, затем числа больше b
672         PrintFileElementsInRange("numbers.txt", 3, 7);
673         // 13. Печать слов из файла: сначала слова, начинающиеся на
гласную, затем на согласную
674         PrintWordsByFirstLetter("words.txt");
675         // 14. Печать чисел из файла: сначала положительные числа, затем
отрицательные числа
676         PrintNumbersBySign("numbers.txt");
677         // 15. Печать слов из файла: сначала слова с прописной буквы,
затем со строчной
678         PrintWordsByCase("words.txt");
679         // 16. Печать данных о сотрудниках: сначала данные о мужчинах,
затем о женщинах
680         PrintEmployeesByGender("employees.txt");
```

```
681         // 17. Печать данных о сотрудниках: сначала данные о сотрудниках
с зарплатой меньше 10000, затем остальные
682         PrintEmployeesBySalary("employees.txt");
683         // 18. Печать данных о сотрудниках: сначала данные о сотрудниках
младше 30 лет, затем остальные
684         PrintEmployeesByAge("employees.txt");
685         // 19. Печать данных о студентах: сначала данные о студентах,
успешно сдавших сессию, затем остальные
686         PrintStudentsBySessionResult("students.txt");
687         // 20. Печать данных о студентах: сначала данные о студентах,
успешно обучающихся на оценки 4 и 5, затем остальные
688         PrintStudentsByGoodMarks("students.txt");
689         Console.ReadLine();
690     }
691 }
```

Вывод: в результате работы мы получили навыки использования решения задач с использованием стандартных коллекций на языке C#.

Основная литература

1. Зыков, С. В. Введение в теорию программирования. Объектно-ориентированный подход : учебное пособие / С. В. Зыков. — 3-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2021. — 187 с. — ISBN 978-5-4497-0926-4. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: http://www.iprbookshop.ru/102007.html.
2. Павловская, Т. А. Программирование на языке высокого уровня C# : учебное пособие / Т. А. Павловская. — 3-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2021. — 245 с. — Текст : электронный — URL: http://www.iprbookshop.ru/102051.html.
3. Биллиг, В. А. Основы объектного программирования на C# (C# 3.0, Visual Studio 2008) : учебник / В. А. Биллиг. — 3-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2021. — 409 с. — Текст : электронный — URL: http://www.iprbookshop.ru/102029.html.
4. Горелов, С. В. Современные технологии программирования: разработка Windows-приложений на языке C#. В 2 томах. Т.I : учебник / С. В. Горелов ; под редакцией П. Б. Лукьянова. — Москва : Прометей, 2019. — 362 с. — Текст : электронный — URL: http://www.iprbookshop.ru/94532.html.
5. Горелов, С. В. Современные технологии программирования: разработка Windows-приложений на языке C#. В 2 томах. Т.II : учебник / С. В. Горелов ; под редакцией П. Б. Лукьянова. — Москва : Прометей, 2019. — 378 с. — Текст : электронный — URL: http://www.iprbookshop.ru/94533.html.