**Задание 14.**

В среде *Microsoft* *Visual Studio* на языке *Visual C#* в консольном режиме составить программное обеспечение для решения типовых задач программирования по тематике «Классы и объекты» («*Classes and Objects*»), заданных по варианту.

**Элементы рабочего задания:**

1. Общими для всех вариантов являются следующие положения:

–  все поля и методы создаваемого класса должны быть глобальными / публичными (*public*);

–  по умолчанию необходимо отказаться от пустого конструктора класса (*instance constructor*), если иное не предусмотрено вариантом индивидуального задания;

–  конструктор класса должен содержать перегрузку (или несколько перегрузок (*overloading*)), предоставляющую возможность для копирования экземпляра класса (или его части);

– вести документирование полей и методов в авторских классах (раздел документирования открывается постановкой трёх подряд идущих символов «/» перед сигнатурой метода или перед полем);

– вызов методов должен быть организован через строковое меню;

–  обязательному переопределению (*overriding*) подлежит **виртуальный метод *ToString()* класса *Object[[1]](#footnote-1)***.

2. **Учесть все возможные ограничения**, накладываемые не только на переменные, но и на участвующие в расчёте функциональные зависимости**[[2]](#footnote-2)**.

2. В части вариантов индивидуального задания необходимо сведения об объектах хранить в текстовых файлах, в другой части – в константах, являющихся неотъемлемой частью кода программы.

3. **Подобрать корректные тестовые примеры в необходимом количестве[[3]](#footnote-3).**

4. Код составленного программного обеспечения **сопроводить комментариями** (элементы программистской этики)**[[4]](#footnote-4)**. Комментарии призваны облегчить восприятие кода как самим автором спустя время, прошедшее с момента завершения работы над программным обеспечением, так и лицом, осуществляющим нормоконтроль кода.

5. **Отчёт по решённой задаче составить** по схеме, изложенной в методических указаниях**[[5]](#footnote-5)**. Для удобства работы обучающихся далее представлена памятка основных разделов отчёта:

0. Титульный лист

1. Цель работы

2. Формулировка задачи

3. Блок-схема алгоритма

4. Подбор тестовых примеров

5. Листинг (код) программы

6. Расчёт тестовых примеров на ПК

7. Вывод по работе

6. **Схему алгоритма оформлять** согласно правилам**[[6]](#footnote-6)**. Помнить, что при работе с методами необходимо вводить нумерацию блоков и рассматривать содержимое методов в отдельных схемах алгоритма. Помнить, что оконечные блоки методов помимо надписей «Начало» и «Конец» должны содержать сигнатуры методов: либо краткие, в которых присутствует только наименование метода без перечисления формальных параметров (все параметры перечисляются далее в блоке «Данные» внутри метода); либо полные, в которых перечисляются все формальные параметры и их типы данных (в этом случае блок «Данные» внутри метода не размещается и не заполняется).

**7. Для привлечения внимания к наиболее существенным изменениям элементов рекомендуется не забывать про возможности изменения цветов фона и шрифта консоли.**

8. Материалы, подготавливаемые **к отправке по электронной почте**, оформляются согласно **Приложению 4** Сборника задач**[[7]](#footnote-7)**. Если материалы направляются на проверку через платформу *GitHub*, то **Приложением 4** разрешается не руководствоваться.

9. Настоятельно рекомендовано перед решением задачи **ознакомиться с рекомендациями**, собранными в **Приложении 3** Сборника задач**[[8]](#footnote-8)**.

10. Не забывать **давать значимые имена переменным**, проектам, решениям, файлам, методам, перечислимым переменным и элементам внутри них, а также другим разрабатываемым компонентам**[[9]](#footnote-9)**. При нежелании использовать значимые имена переменных в отчёте необходимо составить в разделе, содержащем формулировку задачи, таблицу соответствия вида:

Таблица 1 – Пример таблицы соответствия переменных

|  |  |  |  |
| --- | --- | --- | --- |
| **№ п/п** | **Наименование переменной в программе** | **Смысловое содержание используемой переменной** | **Тип данных** |
| 1 | *a* | Ускорение | Вещественное |
| 2 | *v* | Скорость | Вещественное |
| 3 | *S* | Пройдённый путь | Вещественное |
| 4 | *t* | Затраченное время | Вещественное |
| 5 | *x* | Абсцисса точки на Декартовой плоскости | Целое |
| 6 | *y* | Ордината точки на Декартовой плоскости | Целое |
| 7 | *z* | Аппликата точки в пространстве | Целое |

**Внимание! При выявлении противоречивых условий и формулировок среди вариантов индивидуального задания не предпринимать самостоятельных решений по устранению противоречий, а обратиться за консультацией к преподавателю, читающему курс «Программирования и основ алгоритмизации».**

**Дополнительно обратите внимание:**

**1. В подавляющем большинстве вариантов обход сопрягаемых (вложенных) классов реализуется посредством перечислимого типа данных (*enum*).**

**2. Во всех вариантах дата/временные параметры намеренно указанно объявлять длинным целым типом данных в формате «ддммгггг». Так, если известно, что дата рождения некоторой персоны – 28 февраля 1978, то это значение даты должно соответствовать целому «28021978». Разбор целочисленной даты выполнять посредством алгоритма деления на степень (степени) 10.**

**Задание состоит только из Индивидуальной части. Общая часть в нём отсутствует.**

**Вариант №1**

# Создайте класс: «Круг».

Возможное имя класса: «*clsCircle*».

**Важно!** Ни один круг не может быть создан с отрицательным радиусом.

**Поля (*fields*):**

– идентификатор круга (тип: *int*);

– абсцисса центральной точки круга (тип: *float*);

– ордината центральной точки круга (тип: *float*);

– радиус круга (тип: *float*).

**Конструктор (*instance constructor*):**

1.  Пустой конструктор;

2.  Конструктор с тремя входными параметрами:

а) целый для идентификатора,

б) строковый для абсциссы и ординаты одновременно (#.##; #.##),

в) вещественный одинарной точности – радиус.

**Методы (*methods*):**

1. Инициализация параметров круга, созданного пустым конструктором (имя: *initCircle*; тип: *-*);

2. Создание концентрических кругов (круг, вызывающий метод, «подтягивается» своим центром к кругу-эталону, указанному в качестве параметра; имя: *forceCenter*; тип: *clsCircle*);

3. Выравнивание центров кругов (оба круга меняют положение своих центров на усреднённое значение между их центрами с округлением в сторону большего целого значения; имя: *alignCenter*; тип: *clsCircle*);

4. Получение площади круга (имя: *findArea*;тип: *double*).

**Хранение элементов в программе (*storage*):**

– круги хранятся в текстовом файле (не менее 5 в начальном приближении для тестирования);

– в программе круги хранятся в статическом массиве (тип: *clsCircle[]*), размещённом в модуле\*-классе данных (имя: *mdlData*);

– новые круги сохраняются во второй текстовый файл, будучи созданными в процессе работы программы, и, одновременно, в массив, увеличивающийся для хранения новых экземпляров;

– в новой сессии работы программы второй файл составляется заново.

**Вывод (*ToString()*):**

«<(𝑥; 𝑦); 𝑟>».

\* – статический класс.

**Вариант №2**

# Создайте класс: «Интернет-сайт».

Возможное имя класса: «*clsSite*».

**Поля (*fields*):**

– уникальный номер сайта в списке (тип: *short*);

– протокол доступа к сайту (тип: *string*);

– доменная зона сайта (тип: *string*);

– хостинг / владелец сайта (тип: *string*);

– верхние уровни домена сайта (тип: *string[]*);

– *IP*-адрес сайта (тип: *long*);

– название заглавной страницы сайта (тип: *string*);

– название текущей просматриваемой страницы сайта (тип: *string*);

– дата посещения сайта (тип: *long*);

– признак защищённости протокола (тип: *bool*).

**Конструктор (*instance constructor*):**

1. Пять входных параметров\*:

– адресная строка сайта,

– строка с заголовком сайта,

– строка с названием текущей страницы сайта,

– строка *IP*-адреса (в формате *IPv4*),

– строка с датой посещения сайта (в формате дд.мм.гггг).

\* – идентификатор сайта не хранится в файле, он назначается по порядковому номеру строки, содержащей, собственно, сайт и его параметры.

**Методы (*methods*):**

1. Формирование адресной строки сайта для вставки в браузер по параметрам сайта: (имя: *createSiteString*; тип: *string*).

2. Срез верхних уровней домена из адресной строки сайта, сформированной для браузера (имя: *getUpperDomainLevels*; тип: *string[]*).

3. Формирование библиографической ссылки\*\* на сайт (имя: *formatBibliography*; тип: *-*).

4. Изменение даты посещения сайта по входному строковому представлению (имя: *changeVisitDateByString*; тип: *long*).

5. Определение защищённости\*\*\* протокола (имя: *detectSecurity*; тип: *bool*).

\*\*\* – последняя буква «*s*» в названии протокола является признаком защищённости протокола, в частности *http*.

**Хранение элементов (*storage*):**

– адреса сайтов и их параметры записаны в текстовый файл; файл дополняется и обновляется только вручную через «Блокнот»; изменения, выполняемые в процессе работы программы, не сказываются на содержимом файла;

– массивы и списки для хранения не предусмотрены.

**Вывод (*ToString()*):**

Выводить информацию о сайте в формате:

«Яндекс [[*https://ya.ru*](https://ya.ru/)] – *IP*: 5.255.255.242»

«Ресторан *MANSARD* [[*http://mansard-moscow.ru*](http://mansard-moscow.ru/)] – *IP*: 185.165.123.36»

\*\* – библиографическая ссылка на сайт: «{название сайта} [Электронный ресурс] : {название текущей страницы сайта}. *URL*: {адресная строка сайта} (дата обращения: {дата в формате дд.мм.гггг})»

Параметрические скобки {} – не печатаемая информация.

Пример: «*Lenta.ru* – Новости России и мира сегодня [Электронный ресурс] : Эффективные кадры. Московский метрополитен увеличивает набор водителей трамвая в два раза. *URL*: *https:/​/​lenta.ru/​articles/​2022/​03/​18/​moskkadry/* (дата обращения: 23.03.2025)».

**Вариант №3**

# Создайте класс: «Покупатель».

Возможное имя класса: «*clsCustomer*».

**Поля (*fields*):**

– идентификатор покупателя (тип: *int*);

– фамилия покупателя (тип: *string*);

– имя покупателя (тип: *string*);

– отчество покупателя (тип: *string*);

– адрес покупателя (тип: *string*);

– пол покупателя (тип: *enum*);

– номер кредитной карточки покупателя (тип: *string*);

– номер дисконтной карты покупателя (тип: *string*);

– категория скидки для покупателя (тип: *byte*).

**Конструктор (*instance constructor*):**

1. Строковый массив входных параметров, разбираемых в конструкторе экземпляра класса.

**Методы (*methods*):**

1. Формирование целостной строки Ф.И.О. по отдельным значениям фамилии, имени и отчества (как под требуемый формат *ToString()*, так и общего вида, имя: *concatSurnameNamePat()*; тип: *string*).

2. Возврат процентного значения скидки в зависимости от её категории (имя: *discountPercent()*; тип: *float*);

3. Определение города проживания клиента по известному адресу (имя: *getCustomerCityName()*; тип: *string*);

4. Изменение категории скидки клиента (имя: *changeDiscountCat()*; тип: *-*).

**Хранение элементов (*storage*):**

– константно в коде программы в модуле\*-классе *mdlBase*, в массиве, содержатся 10 покупателей (*clsCustomer[10]*);

– имеется возможностью дополнения перечня покупателей в сессию работы программы, но они записываются в структурный список (*List*<*clsCustomer*>) модуля-класса *mdlBase*;

– чтение из файла и запись в файл не предусмотрены.

**Вывод (*ToString()*):**

«1. Иван Иванович И. (м.) - 8 6845 763 3 - статус скидки: 15%»

«2. Алексей Алексеевич А. (м.) - 4 9999 265 0 - скидка не предоставляется»

\* – статический класс.

**Вариант №4**

# Создайте класс: «Текст».

Возможное имя класса: «*clsText*».

**Поля (*fields*):**

– идентификатор текста (тип: *int*);

– содержимое, собственно, текст (тип: *string*);

– семейство шрифта текста (тип: *enum*);

– размер шрифта текста (тип: *int*);

– цвет шрифта текста (тип: *enum*);

– регистр шрифта текста (тип: *sbyte*: -1 – нижний, 1 – верхний, 0 – нормальный);

– признак всех заглавных букв в тексте (тип: *bool*).

**Конструктор (*instance constructor*):**

1. Пустой конструктор для инициализации экземпляра ненулевыми значениями по умолчанию.

**Методы (*methods*):**

1. Полная реинициализация экземпляра по указанным входным параметрам (имя: *toReInitAll()*; тип: *-*);

2. Реинициализация экземпляра по семейству и размеру шрифта (имя: *toReInitFont()*; тип: *-*);

3. Реинициализация цвета (имя: *toReInitColor()*; тип: *-*);

4. Изменение размера шрифта по указанному значению шага\* (имя: *toChangeFontSize()*; тип: *int*).

\* – Шаги изменения размеров шрифта выполнить по аналогии с тем, как реализовано в офисных приложениях (*Word*, *Excel*, *PowerPoint*)

**Хранение элементов (*storage*):**

– предусмотреть возможность хранения текстовых сообщений в структурном списке (*List<clsText>*) в модуле\*\*-классе *mdlBook* только во время сессии работы с программой.

– чтение в файл и запись в файл не предусмотрены.

**Вывод (*ToString()*):**

«1.«Новый Текст»[*Times New Roman*][14][Красный][нормальный]»

«2.«ДРУГОЙ ТЕКСТ»[*COMIC SANS MS*][10][СИНИЙ][НИЖНИЙ]»

\*\* – статический класс.

**Вариант №5**

# Создайте класс: «Обувь».

Возможное имя класса: «*clsShoes*».

**Поля (*fields*):**

– артикул товара-обуви (тип: *string*);

– размер обуви (тип: *int*);

– бренд обуви (тип: *string*);

– цвет обуви (тип: *enum*);

– тип / разновидность обуви (тип: *enum*);

– цена на обувь (тип: *double*).

**Конструктор (*instance constructor*):**

1. Входные параметры соответствующих типов в количестве 6 штук.

2. Входные параметры поступают по строковым группам:

а) собственно, строки;

б) перечислимые значения;

в) числовые значения.

**Методы (*methods*):**

1. Фильтр обуви по размеру и цвету (имя: *filtBySizeAndColor()*; тип: *clsShoes[]* или *List<clsShoes>*);

2. Фильтр обуви по совпадению начала артикула (имя: *startsWithItemNum()*; тип: *clsShoes[]* или *List<clsShoes>*);

3. Фильтр обуви по совпадению завершения артикула (имя: *endsWithItemNum()*; тип: *clsShoes[]* или *List<clsShoes>*);

4. Переход к следующему цвету (имя: *changeColorNext()*; тип: *-*);

5. Переход к предыдущему цвету (имя: *changeColorPrev()*; тип: *-*);

6. Переход к следующему типу (имя: *nextType()*; тип: *-*);

7. Переход к предыдущему типу (имя: *prevType()*; тип: *-*);

**Хранение элементов (*storage*):**

– в модуле\*-классе *mdlStore* в массиве (*clsShoes[16]*) исходно хранится не менее 16 наименований обуви;

– новые наименования обуви заносятся в текстовый файл;

– наименования обуви, хранящиеся в файле, можно редактировать и удалять.

**Вывод (*ToString()*):**

«(8 07В3 43БА) *Ecco* 37 р. полуботинки красные (3 500 руб.)»

\* – статический класс.

**Вариант №6**

# Создайте класс: «Полином».

Возможное имя класса: «*clsPolynom*».

**Поля (*fields*):**

– идентификатор полинома (тип: *int*);

– наименование переменной аргумента в полиноме (тип: *string*);

– наименование переменной функции полинома (тип: *string*);

– степень полинома *n* (тип: *int*);

– коэффициенты при аргументах полинома (тип: *double[n]*).

**Конструктор (*instance constructor*):**

1. Пустой конструктор.

2. Конструктор со степенью и массивом коэффициентов;

3. Конструктор только со значением степени полинома. Массив коэффициентов при этом создаётся и заполняется нулевыми значениями.

**Методы (*methods*):**

1. Изменение наименования переменной аргумента (имя: *changeArgName()*; тип: *-*).

2. Изменение наименования переменной функции (имя: *changeFuncName()*; тип: *-*).

3. Вывод вектора коэффициентов полинома (имя: *showCoeffsOnly()*; тип: *-*).

4. Переназначение всех коэффициентов полинома (имя: *changeAllCoeffs()*; тип: *-*).

5. Переназначение коэффициента при указанной степени (имя: *changeCoeffAt()*; тип: *-*).

**Хранение элементов (*storage*):**

– массив из 5 полиномов (*clsPolynom[5]*), хранимый в модуле\*-классе *mdlMathStruct*, предполагающий возможность увеличения и сокращения;

– удаление всех элементов из массива запрещено – хотя бы два полинома должно содержаться в нём на постоянной основе.

**Вывод (*ToString()*):**

*G(x) = x^3 – 2x^2 + 7x + 1*

*R(z) = –z^5 + 9z^3 – z*

\* – статический класс.

**Вариант №7**

# Создайте класс: «Принтер».

Возможное имя класса: «*clsPrinter*».

**Поля (*fields*):**

– идентификатор принтера (тип: *int*);

– производитель принтера (тип: *enum*);

– тип принтера (тип: *enum*);

– модель принтера (тип: *string*);

– дата приобретения принтера (тип: *long*);

– гарантийный срок службы принтера в годах (тип: *byte*);

– модель и производитель картриджа принтера (тип: *string*);

– возможность подключения по беспроводной сети (тип: *bool*).

**Конструктор (*instance constructor*):**

1. Конструктор с параметрами соответствующих типов в нужном количестве (разбор входной информации должен происходить перед конструированием экземпляра класса). Идентификаторы присваиваются принтерам по мере появления экземпляров в сессию работы программного обеспечения.

**Методы (*methods*):**

1. Изменить дату приобретения принтера (имя: *changePurchaseDate()*; тип: *-*).

2. Подсчёт количества дней до истечения гарантийного срока принтера (имя: *daysToExpire()*; тип: *int*).

3. Проверка наличия картриджа (имя: *cartridgeExists()*; тип: *bool*).

4. Проверка установки «родных» (от того же производителя) расходных материалов (имя: *cartridgeWithLicense()*; тип: *bool*).

**Хранение элементов (*storage*):**

– массив строк на 3 позиции (*string[3]*) содержится в модуле\*-классе *mdlTextData* и хранит параметры принтеров в формате кортежей (строковых констант) со значениями, разделёнными символами «$»;

– массив принтеров на 6 позиций (*clsPrinter[6]*) содержится в модуле-классе *mdlItems*;

– принтеры, которые перестают помещаться в массив, переносятся на хранение в файл *prn.txt*.

**Вывод (*ToString()*):**

\*\* 1. Лазерный *Hewlett-Packard LaserJet M111a* (19.10.2017) \*\*

\* – статический класс.

**Вариант №8**

# Создайте класс: «Картридж».

Возможное имя класса: «*clsCartridge*».

**Поля (*fields*):**

– идентификатор картриджа для принтера (тип: *int*);

– производитель картриджа для принтера (тип: *enum*);

– тип картриджа для принтера (тип: *sbyte;* -1 – чёрный; 1 – цветной; 0 – комбинированный);

– модель картриджа для принтера (тип: *string*);

– дата приобретения картриджа для принтера (тип: *long*);

– дата заправки картриджа для принтера (тип: *long*);

– количество доступных для печати страниц (тип: *int*);

– признак возможности заправки картриджа для принтера (тип: *bool*).

**Конструктор (*instance constructor*):**

1. Только пустой конструктор.

**Методы (*methods*):**

1. Заправка картриджа (имя: *fillCartridge()*; тип: *-*).

2. Инициализация параметров картриджа (имя: *initCartridge()*; тип: *-*).

3. Вывод информации о картриджах, содержащихся в списке (имя: *showCartridges()*; тип: *-*).

4. Имитация печати указанного количества страниц (имя: *printPages()*; тип: *-*).

5. Активация признака возможности заправки картриджа (имя: *allowToFill()*; тип: *-*).

6. Показ даты в формате «мм/дд/гггг» (имя: *printDate()*; тип: *string*).

**Хранение элементов (*storage*):**

– в модуле\*-классе *mdlShelf* исходно хранится пустой список картриджей для принтера (*List<clsCartridge>*);

– в общем случае список не ограничен по количеству элементов, но требуется реализовать ограничение на 10 наименований картриджей;

– как только все десять позиций списка наполняются – картриджи переносятся в текстовый файл, а список очищается;

– картриджи хранятся в файле (*crt.txt*) без возможности просмотра содержимого через консоль.

**Вывод (*ToString()*):**

$$ 1. *Hewlett-Packard CE310A* (1000 стр.) [19.10.2017] $$

\* – статический класс.

**Вариант №9**

# Создайте класс: «Студент».

Возможное имя класса: «*clsStudent*».

**Поля (*fields*):**

– идентификатор (тип: *int*);

– фамилия, имя, отчество (тип: *string*);

– дата рождения (тип: *long*);

– номер группы (тип: *string*);

– номер студенческого билета (тип: *long*);

– успеваемость (тип: *byte[5]*).

**Конструктор (*instance constructor*):**

1. Пустой конструктор студента.

2. Все входные параметры конструктора соответствующих типов (без конвертации в теле конструктора).

**Методы (*methods*):**

1. Фильтр студентов по учебной группе (имя: *filtByGroup()*; тип: *clsStudent[]* или *List<clsStudent>*);

2. Фильтр студентов по успеваемости (с оценками не ниже указанного значения) (имя: *filtByMarks()*; тип: *clsStudent[]* или *List<clsStudent>*);

3. Определение среднего балла (имя: *getAvgMark()*; тип: *float*);

4. Вычисление возраста (полных лет, имя: *getAge()*; тип: *byte*);

5. Определение подходящей строковой константы (год / года / лет) для входного числового значения (имя: *printAgeAddition()*; тип: *string*);

6. Упорядочивание студентов по возрастанию/убыванию среднего балла (имя: *sortByAvgMark()*; тип: -);

7. Инициализация студента для случая, когда объект был создан через пустой конструктор (имя: *initStudent()*; тип: -).

**Хранение элементов (*storage*):**

– студенты с чётными идентификаторами записываются в файл и считываются из файла (*studEven.txt*);

– студенты с нечётными идентификаторами записываются в файл и считываются из файла (*studOdd.txt*).

**Вывод (*ToString()*):**

«1. Иванов Иван И., билет: 8765 4321; 22 года; ТКИ-541; ср.балл: 3.75»;

«7. Петров Пётр П., билет: 1234 5678; 20 лет; ТУУ-311; ср.балл: 4.50».

**Вариант №10**

# Создайте класс: «Ресторан».

Возможное имя класса: «*clsRestaurant*».

**Поля (*fields*):**

– идентификатор ресторана (тип: *int*);

– наименование ресторана (тип: *string*);

– рейтинг ресторана (тип: *byte*);

– город расположения ресторана (тип: *enum*);

– адрес ресторана (тип: *string*);

– кухня ресторана (тип: *enum*);

– уточнение / комментарий о расположении ресторана (тип: *string*);

– время открытия ресторана (тип: *int*);

– время закрытия ресторана (тип: *int*).

**Конструктор (*instance constructor*):**

1. Входные параметры передаются неразобранной строкой текста, считанной из файла.

**Методы (*methods*):**

1. Фильтр ресторанов по городу (имя: *filtByCityName()*; тип: *clsRestaurant[]* или *List<clsRestaurant>*);

2. Фильтр ресторанов по указанному рейтингу (имя: *filtByRate()*; тип: *clsRestaurant[]* или *List<clsRestaurant>*);

3. Фильтр ресторанов по указанной кухне (имя: *filtByCuisine()*; тип: *clsRestaurant[]* или *List<clsRestaurant>*);

4. Принудительное присвоение конкретного значения рейтинга (имя: *forceRateValue()*; тип: *-*);

5. Расчёт длительности работы в часах (имя: *countWorkHours()*; тип: *double*);

6. Формирование полного адреса ресторана (имя: *printAddress()*; тип: *string*);

7. Сортировка ресторанов по рейтингу (имя: *sortByRate()*; тип: *clsRestaurant[]* или *List<clsRestaurant>*);

8. Формирование времени работы ресторана (имя: *printTimeString()*; тип: *string*).

\* – рейтинг ресторана можно изменять в пределах от 1 до 5 звёзд.

**Хранение элементов (*storage*):**

– в текстовом файле (*PupularRestaurants.txt*) с возможностью дополнения.

**Вывод (*ToString()*):**

«1. «Федя, дичь!» (русская кухня); рейтинг: "\*\*\*\*"; адрес: г. Москва, Мытная ул. 74 (Даниловский рынок); время работы: 12:00 – 23:30».

**Вариант №11**

# Создайте класс: «Абонент».

Возможное имя класса: «*clsSubscriber*».

**Поля (*fields*):**

– идентификатор абонента (тип: *int*);

– фамилия абонента (тип: *string*);

– имя абонента (тип: *string*);

– отчество абонента (тип: *string*);

– адрес проживания абонента (тип: *string*);

– дебет счёта абонента (тип: *double*);

– кредит счёта абонента (тип: *float*);

– номер кредитной карты абонента (тип: *string*);

– номер телефона абонента (тип: *long*);

– время разговоров абонента (тип: *long*).

**Конструктор (*instance constructor*):**

1. Входной параметр – одномерный массив строк.

**Методы (*methods*):**

1. Фильтр абонентов, превысивших время разговоров по тарифу (имя: *filtOverNorm()*; тип: *clsSubscriber[]* или *List<clsSubscriber>*).

2. Определение должников (имя: *checkDebt()*; тип: *bool*) – считать должником (*true*) абонента, у которого величина дебета в отрицательной области по модулю превышает величину кредита.

3. Определение по адресу города проживания (имя: *getCityName()*; тип: *string*).

4. Маскирование номера кредитной карты предложенными служебными символами кроме последних четырёх цифр (имя: *printMaskedCardNum()*; тип: *string*).

5. Сортировка абонентов в алфавитном порядке (имя: *sortByName()*; тип: *-*).

6. Вывод номера телефона в формате +x (xxx) xxx-xx-xx (имя: *printPhoneFormat()*; тип: *string*).

**Хранение элементов (*storage*):**

– константно в коде программы в модуле\*-классе *mdlInfo*, в массиве, содержатся 12 абонентов (*clsSubscriber[12]*);

– имеется возможностью дополнения перечня абонентов в сессию работы программы, но они записываются в структурный список (*List*<*clsSubscriber*>) модуля-класса *mdlInfo*;

– чтение из файла и запись в файл не предусмотрены.

**Вывод (*ToString()*):**

«0001. Иван Иванович И. -Тарифище- г. Москва +7 (499) 777-88-99»

\* – статический класс.

**Вариант №12**

# Создайте класс: «Тариф».

Возможное имя класса: «*clsTariff*».

**Поля (*fields*):**

– идентификатор тарифа (тип: *int*);

– номер телефона привязки тарифа (тип: *long*);

– лимит времени разговоров по тарифу в секундах (тип: *long*);

– лимит коротких сообщений по тарифу в штуках (тип: *int*);

– абонентская плата по тарифу (тип: *float*).

**Конструктор (*instance constructor*):**

1. Входные параметры – строки (разбор по полям и типам в теле конструктора).

**Методы (*methods*):**

1. Формирование строки в указанием часов, минут и секунд согласно лимиту (имя: *printNaturalTimeFormat()*; тип: *string*).

2. Изменение величины абонентской платы с учётом скидки (имя: *changePriceWithDiscount()*; тип: *float*).

3. Вывод номера телефона в формате +x(xxx)xxxxxxx (имя: *printPhoneFormat()*; тип: *string*).

4. Увеличение / уменьшение лимита коротких сообщений (имя: *changeSMSLimit()*; тип: *-*).

5. Увеличение / уменьшение лимита времени разговоров на указанное количество часов (имя: *changeHourTimeLimit()*; тип: *-*).

6. Увеличение / уменьшение лимита времени разговоров на указанное количество минут (имя: *changeMinTimeLimit()*; тип: *-*).

**Хранение элементов (*storage*):**

– предусмотреть возможность хранения тарифов в структурном списке (*List<clsTariff>*) в модуле\*-классе *mdlMenu* только во время сессии работы с программой.

– чтение в файл и запись в файл не предусмотрены.

**Вывод (*ToString()*):**

«01. Тарифище (а/п: 750 р./мес.; лимит: 1000 минут; sms: 50 шт.)»

\* – статический класс.

**Вариант №13**

# Создайте класс: «Маршрут электробуса».

Возможное имя класса: «*clsElectroBusRoute*»).

**Поля (*fields*):**

– идентификатор маршрута электробуса (тип: *int*);

– номер маршрута\* электробуса (тип: *string*);

– остановка начала маршрута электробуса (тип: *string*);

– остановка конца маршрута электробуса (тип: *string*);

– промежуточные остановки электробуса на маршрута (тип: *string[]*);

– время выхода на маршрут электробуса (тип: *long*);

– время ухода с маршрута электробуса (тип: *long*);

– привязка парку / депо маршрута электробуса (тип: *enum*).

\* – номер маршрута может быть представлен численно-буквенной комбинацией, например, «т16», «с10», «мц1», «76» и т.д.

**Конструктор (*instance constructor*):**

1. Каждый входной параметр – строковый. Приведение типов выполняется внутри конструктора.

**Методы (*methods*):**

1. Установка указанного значения числовой части номера маршрута (имя: *setRouteNum*; тип: *-*).

2. Добавление нового стартового остановочного пункта с переносом предыдущего стартового в массив промежуточных остановок (имя: *addNewFirstTerminal*; тип: *-*).

3. Добавление нового конечного остановочного пункта с переносом предыдущего конечного пункта в массив промежуточных остановок (имя: *addNewLastTerminal*; тип: *-*).

4. Добавление нового промежуточного остановочного пункта (имя: *addNewRegularStop*; тип: *void*).

5. Увеличение / сокращение времени выхода на маршрут / ухода с маршрута на указанное количество минут (имя: *incomingMinsIncDec*; тип: *long*).

6. Увеличение / сокращение времени выхода на маршрут / ухода с маршрута на указанное количество часов (имя: *incomingHoursIncDec*; тип: *long*).

7. Увеличение / сокращение времени выхода на маршрут / ухода с маршрута на указанное количество секунд (имя: *incomingSecIncDec*; тип: *long*).

**Хранение элементов (*storage*):**

– в текстовом файле содержатся три маршрута электробуса, которые не подлежат ни удалению, ни корректировке, ни перестановке;

– текстовый файл можно дополнять маршрутами электробуса;

– все добавленные маршруты, кроме первых трёх фиксированных, можно удалять, изменять / обновлять, обменивать местами;

– в *Program.cs* предусмотреть глобальный, статический список (тип: *List<clsElectroBusRoute>*) для хранения маршрутов электробусов в сессию работы программы.

**Вывод (*ToString()*):**

&&&&&&&&

Т76. Холмогорская улица – метро ВДНХ

Работает с 5:40 до 2:10

6-й автобусный парк

&&&&&&&&

**Вариант №14**

# Создайте класс: «Четырёхугольник».

Возможное имя класса: «*clsTetragon*».

**Поля (*fields*):**

– идентификатор четырёхугольника (тип: *ushort*);

– координата *x* (абсцисса) левой верхней точки четырёхугольника (тип: *int*);

– координата *y* (ордината) левой верхней точки четырёхугольника (тип: *int*);

– наименование левой верхней точки четырёхугольника (тип: *char*);

– координата *x* (абсцисса) правой верхней точки четырёхугольника (тип: *int*);

– координата *y* (ордината) правой верхней точки четырёхугольника (тип: *int*);

– наименование правой верхней точки четырёхугольника (тип: *char*);

– координата *x* (абсцисса) левой нижней точки четырёхугольника (тип: *int*);

– координата *y* (ордината) левой нижней точки четырёхугольника (тип: *int*);

– наименование левой нижней точки четырёхугольника (тип: *char*);

– координата *x* (абсцисса) правой нижней точки четырёхугольника (тип: *int*);

– координата *y* (ордината) правой нижней точки четырёхугольника (тип: *int*);

– наименование правой нижней точки четырёхугольника (тип: *char*).

**Конструктор (*instance constructor*):**

1. Каждый входной параметр – строковый. Приведение типов выполняется внутри конструктора.

**Методы (*methods*):**

1. Вычисление периметра четырёхугольника (имя: *getPerimeter*; тип: *int*).

2. Вычисление площади четырёхугольника с предварительной проверкой равенства всех его углов 90 градусов; при наличии иных углов площадь не считать (имя: *getSquare*; тип: *int*).

3. Расчёт длины только верхней стороны четырёхугольника (имя: *getTopLenght*; имя: тип: *int*).

4. Расчёт длины только нижней стороны четырёхугольника (имя: *getBottomLenght*; тип: *int*).

5. Расчёт длины только левой стороны четырёхугольника (имя: *getLeftLenght*; тип: *int*).

6. Расчёт длины только правой стороны четырёхугольника (имя: *getRightLenght*; тип: *int*).

7. Вывод значений длин всех сторон четырёхугольника с указанием наименований этих сторон (имя: *eachSideLength*; тип: *-*).

8. Контроль корректности\* указания наименования точки четырёхугольника (имя: *controlPointName*; тип: *bool*).

\* – название точки может быть только латинской или кириллической буквой заглавного начертания. При обнаружении некорректного символа менять его на ближайший свободный из числа корректных.

**Хранение элементов (*storage*):**

– в модуле\*\*-классе *mdlTemplates* в статическом массиве из 20 элементов константно содержатся стандартные, не нулевые заготовки четырёхугольников; в сессию работы программы содержимое каждой заготовки можно менять; создание новых заготовок не предусматривать; удаление существующих заготовок не предусматривать.

\*\* – статический класс.

**Вывод (*ToString()*):**

«Четырёхугольник 1 (А: 0, 0; B: 10, 0; C: 0, 5; D: 10, 5)»

**Вариант №15**

Создайте класс «Ноутбук».

Возможное имя класса: «*clsNotebook*».

**Поля (*fields*):**

– артикул ноутбука (тип: *int*);

– оперативная память ноутбука (тип: *int*);

– диагональ монитора ноутбука (тип: *string*);

– разрешающая способность монитора ноутбука по ширине (тип: *int*);

– разрешающая способность монитора ноутбука по высоте (тип: *int*);

– вес ноутбука (тип: *float*);

– объём жёсткого диска ноутбука (тип: *float*);

– тип жёсткого диска ноутбука (тип: *enum*);

– тип матрицы дисплея ноутбука (тип: *enum*);

– цена ноутбука (тип: *float*);

– производитель ноутбука (тип: *enum*).

**Методы (*methods*):**

1. Фильтр ноутбуков по величине диагонали (имя: *filtByDiag()*; тип: *clsNotebook[]* или *List<clsNotebook>*);

2. Фильтр ноутбуков по рейтингу (имя: *filtByRate()*; тип: *clsNotebook[]* или *List<clsNotebook>*);

3. Фильтр ноутбуков по весовому диапазону (имя: *filtByWeight()*; тип: *clsNotebook[]* или *List<clsNotebook>*);

4. Принудительное присвоение конкретного значения рейтинга (имя: *forceRateValue()*; тип: *-*);

5. Формирование полного перечня параметров дисплея (имя: *printAllAboutDisplay()*; тип: *string*);

6. Определение цены с указанным значением скидки (имя: *printPriceWithDiscount()*; тип: *string*);

7. Формирование полного перечня параметров памяти (имя: *printAllAboutMemory()*; тип: *string*).

**Хранение элементов (*storage*):**

– массив из 8 ноутбуков (*clsNotebook[8]*), хранимый в модуле\*-классе *mdlMarket*, предполагающий возможность увеличения и сокращения;

– удаление всех элементов из массива запрещено – хотя бы три ноутбука должны содержаться в нём на постоянной основе.

**Вывод (*ToString()*):**

5264830162. Ноутбук *ASUS Vivobook 15 X1504VA-BQ868* 15.6" 1920x1080 *IPS* / *Intel Core i3 1315U* / 8 ГБ *DDR4* / *SSD* 512 ГБ / 1.7 кг – 80 539,59 руб.

\* – статический класс.

1. Помнить, что каждый пользовательский класс основывается на Объекте (*Object*). Объект содержит три виртуальных метода: *Equals()*, *ToString()*, *GetHashCode()*. Объект содержит один метод: *GetType()*. [↑](#footnote-ref-1)
2. Пример корректного подбора тестовых примеров в нужном количестве представлен в разделе 7 Сборника задач [Сафронов, А. И. Получение первичных профессиональных умений и навыков научно-исследовательской деятельности: Сборник задач для проведения аудиторных занятий по учебной практике / А. И. Сафронов, Н. Н. Зольникова, В. Г. Новиков. – Москва: РУТ. – 2019. – 91 с. – *EDN SXMWOD*]. Скачать можно на сайте Национальной Электронной Библиотеки (НЭБ) *elibrary.ru*: [*https://www.elibrary.ru/item.asp?id=46307421*](https://www.elibrary.ru/item.asp?id=46307421). [↑](#footnote-ref-2)
3. То же, что и сноска 2. [↑](#footnote-ref-3)
4. Пункт 6 на странице 391 [Сафронов, А. И. Способы проектирования эргономичных графических пользовательских интерфейсов для интеллектуальных транспортных систем / А. И. Сафронов // Интеллектуальные транспортные системы: Материалы III Международной научно-практической конференции, Москва, 30 мая 2024 года. – Москва: РУТ. – 2024. – С. 385-394. – *DOI 10.30932/9785002446094-2024-385-394*. – *EDN TGVERU*]. Скачать можно на сайте Национальной Электронной Библиотеки (НЭБ) *elibrary.ru*: [*https://www.elibrary.ru/item.asp?id=68533620*](https://www.elibrary.ru/item.asp?id=68533620). [↑](#footnote-ref-4)
5. Пример оформления отчётной документации [Сафронов, А. И. Составление отчётной документации по решённым задачам алгоритмизации и программирования: Учебно-методическое пособие для проведения аудиторных занятий по учебной практике / А. И. Сафронов, Н. Н. Зольникова, В. Г. Новиков. – Москва: РУТ. – 2018. – 83 с. – *EDN WNAIFP*]. Скачать можно на сайте Национальной Электронной Библиотеки (НЭБ) *elibrary.ru*: [*https://www.elibrary.ru/item.asp?id=46271697*](https://www.elibrary.ru/item.asp?id=46271697). [↑](#footnote-ref-5)
6. Правила изложены в разделе 8 того же учебно-методического пособия, что указано по сноске 5. [↑](#footnote-ref-6)
7. То же, что и сноска 2. [↑](#footnote-ref-7)
8. То же, что и сноска 2. [↑](#footnote-ref-8)
9. То же, что и сноска 4. [↑](#footnote-ref-9)