ЗГОДА Ю. Н.

ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Ю. Н. Згода

ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Учебно-методическое пособие

Санкт-Петербург
Наукоемкие технологии
2024

УДК 004.415.2
ББК 32.97
З-45

Рецензент:
Мария Михайловна Ромаданова,
Санкт-Петербургский государственный архитектурно-строительный университет

З-45 Згода Ю. Н. Проектирование программного обеспечения: учебно-методическое пособие / Ю. Н. Згода. – СПб.: Наукоемкие технологии, 2024. – 74 с.

ISBN 978-5-907804-73-9

Учебно-методическое пособие предназначено для освоения студентами основ проектирования программного обеспечения, содержит основные сведения о распространенных практиках оформления программного кода и принципах объектно-ориентированного проектирования. Пособие содержит набор заданий с описанием требований и рекомендаций по их выполнению и может быть использовано преподавателями при подготовке и проведении занятий по соответствующим дисциплинам, а также студентами и аспирантами при изучении особенностей проектирования программного обеспечения.

В пособии приведен перечень задач по таким темам, как: реализация алгоритмов, организация и оформление программного кода, применение принципов проектирования SOLID и паттернов объектно-ориентированного проектирования.

ISBN 978-5-907804-73-9

Оглавление

Введение

Одним из наиболее значимых факторов при разработке ПО является управление сложностью программного кода [1]. При низкой организованности исходного кода становится все сложнее вносить в него изменения, добавлять новую функциональность в программу или исправлять ошибки. По этой причине большое внимание в разработке ПО уделяется вопросам проектирования программных систем, организации совместной работы различных модулей программы, повышению читаемости и структурированности программного кода [2, 3].

Данная работа включает в себя перечень заданий, направленных на формирование у учащихся навыков, необходимых для работы над сложными, многофункциональными программными решениями. Задания посвящены таким темам, как оформление программного кода, применение принципов проектирования SOLID [4] и паттернов объектно-ориентированного проектирования [5]. Большинство приведенных в данной работе заданий может быть выполнено с использованием практически любого языка программирования. Однако задания, связанные с переработкой унаследованного исходного кода, содержат примеры на языке программирования C#. При необходимости этот код можно адаптировать для любого другого объектно-ориентированного языка, что позволяет интегрировать его с другими языками программирования.

Данные учебно-методические рекомендации могут быть полезны студентам и учащимся, занимающимся изучением принципов проектирования ПО, а также преподавателям соответствующих дисциплин.

Задание 1. Алгоритмы

Суть заданий данного раздела заключается в формировании у учащихся навыков написания программного кода с соблюдением стандартов именования программных сущностей и грамотного оформления кода. Довольно часто можно столкнуться с ситуацией, когда студенты при реализации относительно сложных алгоритмов и программ не придают большого значения вопросам оформления. Подобные программы зачастую сложно проверять преподавателю. Более того, эти программы трудно использовать самому учащемуся в случае, если последующие задания в учебном плане предполагают повторное использование ранее написанного программного кода.

В связи с этим в рамках первого раздела заданий выбрана реализация относительно простых программных алгоритмов с обязательным соблюдением ряда распространенных практик и требований в области оформления программного кода и организации программ. В качестве требований к программному коду рекомендуется установить следующие ограничения.

Ограничения поведения

1. Программа должна иметь возможность тестирования посредством консольного ввода-вывода, в случае если у реализуемых алгоритмов имеются какие-либо входные параметры.

2. Если в задании требуется реализовать несколько методов, возвращающих один и тот же результат разными способами, то на экран необходимо вывести выходное значение каждого из методов.

3. Программа должна корректно завершаться при вводе некорректных данных и выводить соответствующее сообщение об ошибке.

Ограничения структуры

1. Недопустимо размещение логики, связанной с решением задания, непосредственно в методе Main.

2. Процесс решения задачи должен быть реализован отдельным методом, вызываемым в методе Main. При необходимости внутри этого метода код может быть дополнительно разделен на методы.

3. Чтение пользовательского ввода и вывод результатов работы программы также должны быть реализованы в виде отдельных алгоритмов.

4. В рамках чтения пользовательского ввода необходимо реализовать логику проверки корректности вводимых данных (например, если пользователь ввел строку в задаче, где на вход подается число, пользователь должен получить соответствующее сообщение об ошибке).

5. Если в программе используются обработчики исключений, то try-блок должен ограничивать только те строки кода, в которых предполагается возникновение исключений. Помещать все содержимое метода в try-блок недопустимо.

6. Программа должна завершаться только путем использования return в теле Main. Использование Environment.Exit или аналогичного метода недопустимо.

7. Рекурсивный вызов метода Main как способ перезапуска программы при обработке ошибок ввода-вывода или исключений является недопустимым.

Ограничения наименования

1. Имена переменных должно четко передавать предназначение переменной. Примеры некорректных наименований: temp, someVar. Примеры корректных наименований: sum, amountOfArticles.

2. Исключением из предыдущего правила являются распространенные в определенном контексте наименования переменных (например, i, j, k для итерирования в цикле) и распространенные математические обозначения в математических функциях, такие как f(x) или p (x, y, z).

3. В большинстве случаев имена переменных являются существительными (amountOfArticles, numberOfLines). Для булевых переменных чаще всего используются глаголы (isStudent, hasAccess).

4. При выборе наименования переменной следует придерживаться какой-либо из конвенций наименования, например наиболее распространенной в рамках используемого языка программирования (в случае с C# это PascalCase).

5. Имена методов должны быть глаголами (GetResult, EvaluateFunction, GenerateProgram). В случае с методами, возвращающими булевы значения, именование может быть вопросительным глаголом (IsAvailable, HasTasks, ShouldBeRemoved).

6. Для того чтобы упростить написание программного кода в соответствии с конвенцией, учащимся рекомендуется использовать среды разработки с автоматической подсветкой ошибок именования переменных.

Варианты заданий:

Задание 1. Реализовать два метода, каждый из которых выполняет суммирование массива вещественных чисел двойной точности (тип double). Первый метод должен использовать цикл for для суммирования элементов массива. Второй метод должен использовать рекурсию для суммирования.

Примечание: рекурсивно формула для суммирования элементов массива может быть определена следующим образом:

$$\sum_{i=1}^{n} a_i = a_1 + \sum_{i=2}^{n} a_i = a_1 + a_2 + \sum_{i=3}^{n} a_i = \ldots,$$

т. е. словами, сумма элементов массива равна сумме двух слагаемых: первого элемента массива и суммы всех остальных элементов.

Задание 2. Реализовать два метода, каждый из которых выполняет перемножение массива 32-разрядных целых чисел (тип int). Первый метод должен использовать цикл for для суммирования элементов массива. Второй метод должен использовать рекурсию для перемножения элементов.

Примечание: рекурсивно формула для перемножения элементов массива может быть определена следующим образом:

$$\prod_{i=1}^{n} a_i = a_1 \sum_{i=2}^{n} a_i = a_1 a_2 \sum_{i=3}^{n} a_i = \ldots,$$

т. е. произведение всех элементов массива равно произведению первого элемента массива на произведение всех остальных элементов массива.

Для первого метода (перемножение элементов в цикле) при реализации следует применить оптимизацию: если в ходе перемножения элементов встречается хотя бы один элемент, равный нулю, то итоговый результат также равен нулю и необходимости в продолжении вычислений нет.

Задание 3. Реализовать метод, принимающий в качестве единственного аргумента массив чисел типа int. Метод должен возвращать значение true, если во входном массиве все элементы уникальны, и false, если хотя бы два элемента массива равны друг другу. Использование стандартных средств .Net для определения количества уникальных элементов недопустимо.

Задание 4. Реализовать метод, принимающий в качестве аргумента массив строк. Необходимо определить элементы массива с минимальной и максимальной длиной строки. Вернуть конкатенацию этих строк.

Задание 5. Дан массив ненулевых чисел типа int. Вернуть массив типа int такого же размера. Каждый i-й элемент этого массива определяется как произведение всех элементов входного массива, за исключением i-го. Т. е. если входные данные – это массив [1, 3, 5, 7], то метод должен вернуть массив с элементами [3 × 5 × 7 = 105, 1 × 5 × 7 = 35, 1 × 3 × 7 = 21, 1 × 3 × 5 = 15].

Задание 6. На вход в метод подается массив чисел. Метод должен возвращать количество простых чисел в этом массиве. Проверку на простоту числа вынести в отдельный метод.

Задание 7. Аргументами метода являются координаты точки (далее – точка *A*) в двухмерном пространстве, а также два массива: первый содержит координаты точек по оси *x*, второй – координаты по оси *y*. Определить координаты точки, являющейся ближайшей к точке *A*. Все координаты хранятся в переменных типа float32.

Задание 8. Метод принимает в качестве единственного аргумента натуральное число *N*. Он должен вывести на экран простые множители, образующие это число, и вернуть сумму этих множителей.

Задание 9. На вход методу подается целое положительное число. Метод должен вернуть это целое число, записанное в обратном порядке.

Для построения такого числа запрещается использовать преобразование в строковый тип, изменение порядка символов и обратное преобразование в число. «Переворачивание» числа должно быть выполнено только за счет использования математических операций.

Примечание. Чтобы получить последнюю цифру числа, нужно вычислить остаток от деления числа на 10 (например, 123 % 10 = 3). После этого исходное число делится нацело на 10, чтобы «удалить» последнюю цифру. Повторяя этот процесс до тех пор, пока искомое число не обратится в нуль, можно получить все цифры, составляющие число.

Задание 10. Метод принимает в качестве аргументов два натуральных числа *N* и *M*. Метод должен возвращать количество простых чисел на промежутке от *N* до *M* включительно. Проверку на простоту числа необходимо оформить отдельным методом.

Задание 11. Написать метод, принимающий на вход коэффициенты квадратного уравнения и возвращающий true, если у уравнения есть решение в области вещественных чисел, и false – в противном случае. Через выходные аргументы вернуть вещественные корни (если они существуют). Если корни

кратные (дискриминант равен нулю), обе выходные переменные должны содержать значение этого корня. Если вещественных корней нет, то оба выходных аргумента должны быть равны Double.NaN.

Задание 12. На вход в метод подается массив чисел типа double. Метод должен возвращать новый массив, в котором числа отсортированы по возрастанию. Для непосредственной сортировки следует использовать алгоритм пузырьковой сортировки.

Примечание: суть метода сортировки пузырьком заключается в том, чтобы перемещаться от начала до конца неупорядоченной части массива и переставлять соседние элементы, если они расположены неправильно относительно друг друга. В результате первой итерации метода максимальный элемент массива окажется в его конце. В результате второй итерации – предпоследний элемент массива окажется на предпоследней позиции массива и т. д.

Задание 13. Метод принимает в качестве единственного аргумента массив из целых чисел. Возвращаемым значением метода является количество подпоследовательностей этого массива, состоящих из трех нулей подряд.

Примеры входных и выходных данных для данной задачи:

[1, 3, 5, 7, 0, 0, 0, 5, 1] => 1;

[2, -6, -8, 1, 3, 0, 0, 0, 1, 0, 0, 0] => 2;

[0, 0, 0, 0] => 2.

Задание 14. На вход методу подается натуральное число N. Вычислить N-ое число Фибоначчи.

Примечание: последовательность Фибоначчи определяется следующим образом: $F_0 = 0$, $F_1 = 1$, …, $F_i = F_{i-1} + F_{i-2}$, …

Задание 15. Реализовать метод без аргументов. Этот метод должен найти все трехзначные целые положительные числа (т. е. перебрать все числа от 100 до 999), квадрат которых оканчивается тремя одинаковыми цифрами. Метод должен вернуть сумму этих чисел.

Задание 16. Дан метод, принимающий одно натуральное число *N*. Вывести на экран первые *N* простых чисел.

Задание 17. Дано число. Проверить, является ли оно совершенным.

Примечание: число называется совершенным, если оно равно сумме всех своих делителей меньших, чем это число. Например, 28 = 1 + 2 + 4 + 7 + 14.

Задание 18. Метод принимает в качестве аргументов координаты точки (далее – точка А) в трехмерном пространстве, а также три массива: первый содержит координаты точек по оси *x*, второй – координаты по оси *y*, третий – координаты по оси *z*. Определить координаты точки, являющейся наиболее отдаленной от точки А. Все координаты хранятся в переменных типа double.

Задание 19. Реализовать метод без аргументов. Этот метод должен найти все четырехзначные целые положительные числа (т. е. перебрать все числа от 1000 до 9999), квадрат которых оканчивается тремя одинаковыми цифрами. Метод должен вернуть сумму этих чисел.

Примечание. Чтобы получить последнюю цифру числа, нужно определить остаток от деления всего числа на 10 (например, 123 % 10 = 3). Чтобы «удалить» из числа последнюю цифру, его нужно разделить нацело на 10.

Дополнительно. Для повышения эффективности обучения при выполнении этого задания можно предложить студентам использовать разработку через тестирование (англ. TDD – Test-Driven Development). Из-за особенностей написания кода при использовании TDD учащиеся в рамках данной методологии напишут код, удовлетворяющий существенной части ограничений структуры программы [6].

Другой возможной вариацией является разработка модульных тестов по завершении выполнения задания. В рамках модульного тестирования естественным образом происходит разбиение программы на автономные и независимые программные единицы (в данном случае – методы), что дополнительно способствует усвоению рассматриваемого материала [7].

Задание 2. Объектно-ориентированное программирование в C#

Следующее задание направлено на закрепление полученных в рамках предыдущего задания навыков в контексте использования объектно-ориентированного программирования. При выполнении данного задания учащиеся должны придерживаться тех же требований по оформлению программы, что и при выполнении первого задания. Для демонстрации функциональности разработанных классов учащиеся должны реализовать консольный интерфейс.

Задание 1. Реализуйте класс Counter (Счетчик). Счетчик должен включать в себя свойство, хранящее результаты «счета» (открытое для чтения, закрытое для записи, тип int), у него должны быть методы для инкремента счетчика, обнуления и вывода «счета» на экран. Возможный аналог из реального мира – спортивный счетчик для упражнений с экраном и кнопкой для увеличения счетчика.

Реализуйте метод, позволяющий создать новый объект – копию счетчика с сохранением текущего значения его скрытой переменной. Т. к. скрытую переменную извне изменить нельзя, для выполнения задания понадобится добавить в счетчик конструктор, принимающий в качестве аргумента стартовое значение счетчика. Затем в методе для создания копии объекта необходимо вызвать этот конструктор и передать в него текущее значение счетчика.

Создайте (не используя наследование) усовершенствованный счетчик, который при каждом «счете» записывает также момент времени, в который произошел инкремент.

Для хранения списка «щелчков» можно использовать структуру данных List (расширяемый список) объектов типа DateTime. Для получения текущего момента времени используйте статическое свойство DateTime.Now.

Добавьте метод, который выводит на экран отчет обо всех моментах времени, когда происходил инкремент.

Задание 2. Реализуйте класс Vector, представляющий собой математический вектор. Вектор определяется набором чисел a_1, a_2, …, a_n. Эти числа передаются в виде массива в конструктор. Компоненты вектора должны быть изменяемыми извне класса. Вектор состоит как минимум из одной компоненты.

Определите для вектора свойство Length (длина вектора в геометрическом смысле), а также перегрузите операции сложения и вычитания векторов.

Реализуйте метод для вычисления скалярного произведения двух векторов (первый метод доступен через this, второй метод передается как аргумент метода).

Реализуйте метод для вычисления векторного произведения двух векторов.

В случае если действия выполняются над векторами разной размерности, необходимо сгенерировать исключение.

Переопределите метод ToString для корректного отображения компонент вектора при его выводе через Console.WriteLine.

Задание 3. Создайте класс «Крестики-нолики» для реализации одноименной игры. Этот класс должен включать в себя методы для вывода текущей позиции на экран, а также метод для выполнения хода по некоторым координатам. Класс должен содержать скрытое поле, определяющее очередность хода.

Через консоль реализуйте вывод на экран позиции на доске и запрос координат для текущего хода. По окончании партии приложение должно начать игру заново.

Задание 4. Разработайте класс Polynom (полином). Полином при инициализации в конструкторе принимает массив вещественных чисел, определяющих его коэффициенты. Количество коэффициентов, образующих полином, должно быть не менее 2.

Реализуйте метод для вычисления значения полинома в точке, а также перегрузите процедуры сложения, вычитания и умножения полиномов. В случае если суммируемые или вычитаемые полиномы имеют разную степень, необходимо сгенерировать исключение.

Задание 5. Разработайте класс «Reference» (ссылка). Этот класс предназначен для хранения сведений о статьях на научные труды. Он должен хранить в себе такие сведения, как название работы, имена авторов (допустимо хранить имена авторов в виде одной строки, создавать список авторов или вспомогательный класс Authors не требуется), год издания, название журнала, номер тома и номер журнала, а также диапазон страниц в журнале.

Ссылка оформляется следующим образом: <Имена авторов>. <Наименование статьи> // <Наименование журнала>. <Год издания>. № <номер журнала>. С. <начальная страница>–<конечная страница>.

Создайте класс ListOfReferences (список источников). Этот класс должен позволять добавлять новые ссылки и выводить на консоль список литературы. Если в список не добавлено ни одной ссылки при вызове метода вывода, необходимо сгенерировать исключение.

Задание 6. Разработайте класс Car (машина). У машины должны быть марка и модель (строки), объем бензобака, текущий запас топлива и его расход на 1 километр. Расход задается через конструктор и является неизменяемым.

Реализуйте метод для перемещения автомобиля на определенное расстояние (измеряемое в километрах). Перемещение машины в данном условном задании сводится к тому, что у машины изменяются показания одометра (счетчика пройденного расстояния). Если топлива на весь путь не хватит, то метод для перемещения должен вернуть false (машина в этом случае не начинает движение и показания одометра останутся неизменными), иначе – true.

Переопределить метод ToString для корректного вывода информации о машине в консоли.

Задание 7. Реализуйте класс Classroom (аудитория) и класс Student (студент). Студент имеет фамилию, имя и отчество, а также курс и группу. Все поля – только для чтения, их инициализация производится в конструкторе.

У аудитории есть вместимость и наименование аудитории («Компьютерный класс 103», «Аудитория 104» и т. д.). Также аудитория хранит список всех студентов, находящихся в ней на данный момент. Студенты могут покидать аудиторию и заходить в нее, это должно быть реализовано через соответствующие методы класса Classroom.

Реализовать методы аудитории, позволяющие выводить список студентов в компьютерном классе в алфавитном порядке (разрешается использовать стандартные средства сортировки .Net). Реализовать метод, возвращающий true, если в классе присутствуют только студенты одной группы, и false – в противном случае.

Задание 8. Создать абстрактный класс «Элемент файловой системы». Наследники данного класса – файл и каталог. Элемент файловой системы определяется своим наименованием и размером. Размер – абстрактное свойство, реализуемое в наследниках файловой системы. Это свойство должно быть открытым для чтения и недоступным для записи.

Для наследника-файла размер – это постоянное число (количество занимаемых байт памяти). Для наследника-каталога размер – это сумма размеров всех элементов файловой системы (каталогов и файлов), входящих в этот каталог.

В каталог могут быть добавлены любые элементы файловой системы (как файлы, так и другие каталоги).

Задание 9. Разработайте класс «Экран». Экземпляры этого класса – виртуальные экраны, состоящие из «пикселей»-символов. Экран имеет прямоугольную форму (т. е. состоит из сетки N×M пикселей, N, M $\geqslant$ 1) и позволяет пользователю заполнять отдельные пиксели этого экрана, очищать экран (заполнять его пробелами, « ») и отрисовывать экран в консоли.

Размеры экрана должны передаваться через конструктор, их изменение после инициализации объекта должно быть невозможным.

При попытке задать значение для пикселя с некорректными координатами (отрицательные координаты или координаты, выходящие за пределы экрана) необходимо генерировать исключение.

Задание 10. Разработайте класс «Принтер». Принтер обладает очередью печати (список строк). У пользователя имеются специальные методы для добавления строк в очередь печати, очистки очереди и печати всех строк, хранящихся в ней на данный момент. Попытка печати документов при пустой очереди приводит к генерации исключения.

Принтер имеет определенный ресурс печати, по окончании которого печать станет невозможной. Пользователю доступен метод для пополнения ресурса печати.

Печать каждой строки из очереди печати расходует одну единицу ресурса принтера. Если в ходе печати ресурс заканчивается, печать приостанавливается. Как только пользователь пополнит ресурс принтера, он сможет продолжить печать строк из очереди.

Задание 11. Разработайте абстрактный класс «Интерполяция». Этот класс включает в себя два поля для хранения массивов и X, и Y, описывающих некоторую интерполируемую функцию $Y = f(X)$, конструктор для передачи этих массивов в объект, а также метод Interpolate(x), возвращающий интерполируемое значение функции в произвольной точке x.

В конструкторе необходимо проверить, что данные в массиве X отсортированы в порядке возрастания. Если они не отсортированы, необходимо сгенерировать исключение.

Предусмотрите в абстрактном классе проверку значения x на то, что оно лежит внутри интервала, описываемого входными данными X. Если x находится вне этого интервала, при обращении к Interpolate должно генерироваться исключение.

Чтобы реализовать генерацию исключения в абстрактном классе, а не прописывать ее в Interpolate каждого отдельного класса, можно воспользоваться следующим приемом. В абстрактном классе есть неабстрактный метод Interpolate, который выполняет проверку *x* на соответствие диапазону, а затем вызывает защищенный (protected) абстрактный метод innerInterpolate. Метод innerInterpolate следует реализовывать в наследниках абстрактного класса.

Благодаря такому подходу исчезнет необходимость в проверке значения *x* в каждом наследнике класса Interpolate, что сократит объем повторяемого кода.

Реализуйте два наследника класса для вычисления интерполяции в точке *x*.

Первый наследник должен возвращать значение *Y* ближайшей известной точки исходных данных.

Второй наследник должен возвращать значение *Y* как линейную интерполяцию двух ближайших узлов. Это значит, что если значение *x* оказалось между X[i-1] и X[i], то метод вернет значение

$$y(x) = y_{i-1} + \frac{(y_i - y_{i-1})}{x_i - x_{i-1}}(x - x_{i-1})$$

Задание 12. Необходимо реализовать пользовательский интерфейс, который выводит в консоли список доступных математических операций и просит выбрать одну из них (сложение, вычитание, умножение, деление, возведение в степень), затем программа считывает аргументы операции с клавиатуры и выводит на экран результат операции. Тип данных для проведения вычислений – double.

Существует простой вариант решения данной задачи, при котором программа выводит список всех операций, прописанных в коде вручную, а затем через switch выполняет соответствующие выбранной операции действия. Од-

нако использование switch означает, что каждый раз при реализации новой математической операции потребуется вносить изменения в существующий код вывода списка команд, а также непосредственно в Switch.

Используйте ООП для того, чтобы минимизировать количество потенциальных изменений в коде. Создайте абстрактную математическую операцию и реализуйте ее наследников для приведенных выше математических операций. У абстрактной математической операции должно быть текстовое описание и короткая команда, которую должен ввести пользователь.

При такой организации программа упростится до списка (List) математических операций. Остается лишь вывести описание каждой операции, а затем выполнить операцию, соответствующую введенной пользователем команде.

Задание 13. Реализуйте иерархию классов, описывающих человека (Person) и наследников этого класса: студента (Student) и сотрудника (Employee). У студента и сотрудника должны быть дополнительные поля в сравнении с человеком. Например, у студента может быть номер группы и курс, а у сотрудника – подразделение и стаж работы.

Переопределите для этих классов метод ToString таким образом, чтобы он возвращал в строковой форме все сведения, содержащиеся в экземпляре класса.

Создайте абстрактный класс «генератор отчетов», принимающий через конструктор список экземпляров класса Person. Генератор отчетов включает в себя абстрактный метод Print, выводящий на экран отчет, составленный по правилам, определенным в наследнике класса.

Реализуйте следующие генераторы отчетов:

- Генератор, пропускающий студентов и выводящий сведения только о сотрудниках.
- Генератор, пропускающий сотрудников и выводящий сведения только о студентах.

- Генератор, выводящий сведения о сотрудниках и студентах с сортировкой по имени.
- Генератор, выводящий сведения о сотрудниках и студентах с сортировкой по дате рождения.

Задание 14. Создайте абстрактный класс Integrator, реализующий интегрирование произвольной функции *f* методом трапеций. Наследники класса реализуют абстрактный защищенный метод func – интегрируемую функцию. Интегрирование функции должно выполняться открытым методом Integrate(a,b), принимающим пределы интегрирования в качестве аргументов и возвращающим значение соответствующего определенного интеграла.

Задание 15. Создайте класс Integrator, реализующий интегрирование произвольной функции $f(x)$ методом левых прямоугольников (интегрирование выполняется при вызове метода Integrate). Через конструктор в Integrator передается экземпляр класса IntegrableFunction, а также пределы интегрирования a и b. IntegrableFunction – это абстрактный класс с единственным методом Evaluate(double x), возвращающим значение некоторой функции в точке x. Наследники класса IntegrableFunction реализуют эту функцию и тем самым описывают конкретную математическую функцию.

Задание 3. Корректировка кода в соответствии с принципами SOLID

Данный тип заданий рекомендуется выдавать студентам после изучения ими принципов SOLID. SOLID – это акроним, введенный Майклом Фэзерсом (Michael Feathers) для пяти принципов проектирования, сформулированных Робертом Мартином [4]:

- Single responsibility principle – принцип единственной ответственности – у класса должна быть только одна причина для изменения.
- Open-closed principle – принцип открытости/закрытости – программные сущности (классы, модули, функции и т. п.) должны быть открыты для расширения, но закрыты для модификации.
- Liskov substitution principle – принцип подстановки Лисков – должна быть возможность вместо базового типа подставить любой его подтип.
- Interface segregation principle – принцип разделения интерфейса – клиенты не должны вынужденно зависеть от методов, которыми не пользуются; интерфейсы принадлежат клиентам, а не иерархиям.
- Dependency inversion principle – абстракции не должны зависеть от деталей; детали должны зависеть от абстракций.

Суть приводимых ниже заданий заключается в том, чтобы переписать рабочую, но неправильно организованную программу таким образом, чтобы она удовлетворяла принципам SOLID. Возможным дополнением для данного задания является использование студентами техник рефакторинга [3] при его выполнении с последующим указанием тех приемов рефакторинга, которые были использованы при выполнении задания.

Задание 1. Класс «Database» позволяет выполнять поиск пользователей по имени, по дате рождения, по месту рождения, по хобби и т. д. Также этот класс позволяет выводить на экран различные отчеты.

В дальнейшем планируется расширять список доступных критериев поиска, а также планируется расширять перечень генерируемых программой отчетов. Каким образом следует переписать программу, чтобы расширение ее функциональности не приводило к необходимости внесения изменений в уже разработанные классы?

Листинг 3.1. Исходный код для задания 3.1

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace RefactorTask
{
    public class Database
    {
        private List<string> usernames;
        private List<string> userCities;
        private List<DateTime> birthdays;
        private List<string> addresses;

        public Database()
        {
            usernames = new List<string>();
            userCities = new List<string>();
            birthdays = new List<DateTime>();
            addresses = new List<string>();
        }

         public void AddUser(string name, string city, DateTime birthday,
string address)
        {
            usernames.Add(name);
            userCities.Add(city);
            birthdays.Add(birthday);
            addresses.Add(address);
        }

        public int FindUserByName(string name)
        {
            return usernames.IndexOf(name);
        }

        public int FindUserByBirthday(DateTime birthday)
        {
            return birthdays.IndexOf(birthday);
        }
```

```
        public int FindUserByCity(string city)
        {
            return userCities.IndexOf(city);
        }

        public string GetNameById(int id)
        {
            return usernames[id];
        }

        public string GetCityById(int id)
        {
            return userCities[id];
        }

        public DateTime GetBirthDateById(int id)
        {
            return birthdays[id];
        }

        public string GetAddressById(int id)
        {
            return addresses[id];
        }

        public int FindUserByAddress(string address)
        {
            return addresses.IndexOf(address);
        }

        public void ReportForCity(string city)
        {
             Console.WriteLine($"---- Информация по пользователям из города
{city} ----");

           var isUserFromCity = userCities.Select(x => x == city).ToArray();

            var counter = 1;
            for (int i = 0; i < usernames.Count; i++)
            {
                if (isUserFromCity[i])
                      Console.WriteLine($"{counter}. {usernames[i]}, адрес:
{addresses[i]}");

                counter++;
            }

    Console.WriteLine();
        }
```

```
        public void ReportAboutAllUsers()
            {
                Console.WriteLine($"---- Список всех пользователей ----");

                for (int i = 0; i < usernames.Count; i++)
                   Console.WriteLine($"{i + 1}. {usernames[i]}, город: {userCi-
ties[i]} адрес: {addresses[i]}, дата и время рождения: {birthdays[i]}");

                Console.WriteLine();
            }
        }

        public class TestPolygonDatabase
        {
            public static void TestPolygon()
            {
                Database database = new Database();

                    database.AddUser("Иванов Иван", "Санкт-Петербург", new
DateTime(2001, 9, 15), "Улица высокая, дом 3");
                   database.AddUser("Петров Петр", "Москва", DateTime.Today, "Улица
вишневая, дом 10");
                    database.AddUser("Дмитриев Виктор", "Санкт-Петербург", new
DateTime(1999, 8, 14), "Улица вишневая, дом 10");


                int id = database.FindUserByName("Петров Петр");
                Console.WriteLine(database.GetCityById(id));

                database.ReportForCity("Санкт-Петербург");
                database.ReportAboutAllUsers();
            }

            public static void Main(string[] args)
            {
                TestPolygon();
            }
        }
    }
```

Задание 2. Класс ItemReport включает в себя логику по обработке данных о различных товарах, их количестве, стоимости и необходимости в доставке (в случае если заказчик не планирует воспользоваться самовывозом). Также он включает в себя методы по выводу на экран отчета по всем товарам с сортировкой по стоимости или по наименованию.

В дальнейшем планируется расширять список доступных критериев поиска, а также планируется расширять перечень генерируемых программой отчетов. Каким образом следует переписать программу, чтобы расширение ее функциональности не приводило к необходимости внесения изменений в уже разработанные классы?

Листинг 3.2. Исходный код для задания 3.2

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;

namespace RefactorTask
{
    public class ItemReport
    {
        private List<string> itemNames;
        private List<int> amounts;
        private List<float> prices;
        private List<bool> shouldBeDelivered;

        public ItemReport()
        {
            itemNames = new List<string>();
            amounts = new List<int>();
            prices = new List<float>();
            shouldBeDelivered = new List<bool>();
        }

        public void AddItem(string name, int amount, float price, bool deliver)
        {
            itemNames.Add(name);
            amounts.Add(amount);
            prices.Add(price);
            shouldBeDelivered.Add(deliver);
        И

        public int FindItemByName(string name)
        {
            return itemNames.IndexOf(name);
        }
        public string GetNameById(int id)
        {
            return itemNames[id];
        }
```

```
        public int GetAmountById(int id)
        {
            return amounts[id];
        }

        public float GetPriceById(int id)
        {
            return prices[id];
        }

        public bool GetDeliverStatusById(int id)
        {
            return shouldBeDelivered[id];
        }

        public void ReportForItemsAlphabetically()
        {
            Console.WriteLine($"---- Информация по товарам ----");


            var indices = Enumerable.Range(0, itemNames.Count);

            var items = indices.Select(i => new
            {
                name = itemNames[i],
                amount = amounts[i],
                price = prices[i],
                deliver = shouldBeDelivered[i]
            });

            var orderedItems = items.OrderBy(x => x.name);


            var counter = 1;
            foreach (var item in orderedItems)
            {
                        Console.WriteLine($"{counter}.  {item.name}.  стоит
{item.price} у. е. Кол-во: {item.amount}, доставка: {item.deliver}");
                counter++;
            }

            Console.WriteLine();
        }

        public void ReportForItemsByPrice()
        {
            Console.WriteLine($"---- Информация по товарам ----");

            var indices = Enumerable.Range(0, itemNames.Count);
```

```
                var items = indices.Select(i => new
                {
                    name = itemNames[i],
                    amount = amounts[i],
                    price = prices[i],
                    deliver = shouldBeDelivered[i]
                });

                var orderedItems = items.OrderBy(x => x.price);


                var counter = 1;
                foreach (var item in orderedItems)
                {
                            Console.WriteLine($"{counter}.  {item.name}.  стоит
{item.price} у. е. Кол-во: {item.amount}, доставка: {item.deliver}");
                    counter++;
                }

                Console.WriteLine();
            }
        }

        public static class TestPolygonItemReport
        {
            public static void TestPolygon()
            {
                ItemReport report = new ItemReport();

                report.AddItem("Утюг", 10, 20, false);
                report.AddItem("Кофеварка", 5, 40, true);
                report.AddItem("Телевизор", 20, 100, true);

                var id = report.FindItemByName("Кофеварка");
                var coffeeMachineAmount = report.GetAmountById(id);
                Console.WriteLine(coffeeMachineAmount);

                report.ReportForItemsByPrice();

                report.ReportForItemsAlphabetically();
            }

            public static void Main4(string[] args)
            {
                TestPolygon();
            }
        }
    }
```

Задание 3. Класс «ListOfUsers» хранит в себе пользователей некоторого почтового сервиса и список полученных/отправленных сообщений. Дополнительно этот класс позволяет вывести на экран список отправленных и полученных сообщений для конкретного пользователя либо переписку двух пользователей.

В текущем виде вся программная логика сконцентрирована в одном классе. Каким образом ее можно разбить на отдельные части, тем самым выполнив требования SOLID и повысив качество кода?

Листинг 3.3. Исходный код для задания 3.3

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace RefactorTask
{
    public class User
    {
        public string Nickname;
        public DateTime RegisterDate;

        public User(string nickname, DateTime registerDate)
        {
            Nickname = nickname;
            RegisterDate = registerDate;
        }
    }

    public class Message
    {
        public User Sender;
        public User Receiver;

        public string Text;
        public DateTime Time;

        public Message(User sender, User receiver, string text, DateTime time)
        {
            Sender = sender;
            Receiver = receiver;
            Text = text;
            Time = time;
        }
    }
```

```
        public class ListOfUsers
        {
            private List<Message> messages = new List<Message>();
            private List<User> users = new List<User>();

            public void AddUser(User user)
            {
                users.Add(user);
            }

            public void SendMessage(User from, User to, string text)
            {
                messages.Add(new Message(from, to, text, DateTime.Now));
            }

            public User FindUserByName(string name)
            {
                return users.First(x => x.Nickname == name);
            }

            public void PrintReceivedMessages(User user)
            {
               var queryOfMessages = messages.Where(x => x.Receiver == user).Or-
derBy(x => x.Time);

                Console.WriteLine($"----Входящие для {user.Nickname}----");
                foreach (var m in queryOfMessages)
                    Console.WriteLine($"{m.Time}: {m.Text}");

                Console.WriteLine();
            }

            public void PrintSentMessages(User user)
            {
                var queryOfMessages = messages.Where(x => x.Sender == user).Or-
derBy(x => x.Time);

                            Console.WriteLine($"----Отправленные   пользователем
{user.Nickname}----");
                foreach (var m in queryOfMessages)
                    Console.WriteLine($"{m.Time}: {m.Text}");

                Console.WriteLine();
            }

            public void PrintDialog(User user1, User user2)
            {
                var queryOfMessages = messages.Where(x => (x.Sender == user1 &&
x.Receiver == user2)
```

```
                                                                || (x.Sender == user2
&& x.Receiver == user1))
                        .OrderBy(x => x.Time);

                          Console.WriteLine($"----Переписка  между  пользователями
{user1.Nickname} и {user2.Nickname}----");
                    foreach (var m in queryOfMessages)
                        Console.WriteLine($"От {user1.Nickname} к {user2.Nickname}
{m.Time}: {m.Text}");

                    Console.WriteLine();
                }
            }

            public static class TestPolygonListOfUsers
            {
                public static void TestPolygon()
                {
                    ListOfUsers list = new ListOfUsers();
                    var Ivan = new User("Иванов Иван", DateTime.Now);
                    var Petr = new User("Петров Петр", DateTime.Now);
                    var Dima = new User("Викторов Дмитрий", DateTime.Now);
                    list.AddUser(Ivan);
                    list.AddUser(Petr);
                    list.AddUser(Dima);

                    list.SendMessage(Ivan, Dima, "Добрый день!");
                    list.SendMessage(Dima, Ivan, "Здравствуйте!");
                    list.SendMessage(Ivan, Dima, "Завтра будет турнир?");
                    list.SendMessage(Dima, Ivan, "Да, будет.");
                    list.SendMessage(Ivan, Dima, "Нужно напомнить Петру.");
                    list.SendMessage(Ivan, Petr, "Добрый день! Завтра будет турнир,
не забудьте.");

                    list.PrintReceivedMessages(Petr);
                    list.PrintSentMessages(Dima);
                    list.PrintDialog(Ivan, Dima);

                }

               public static void Main(string[] args)
                {
                    TestPolygon();
                }
            }
        }
```

Задание 4. Класс Phonebook содержит в себе список телефонных контактов. Список включает в себя фамилию и имя контакта, а также все его телефоны (у одного абонента может быть несколько номеров). Список поддерживает возможность поиска по номеру телефона и поиска по имени. В дальнейшем список сценариев поиска может быть расширен.

Дополнительно данный класс включает в себя логику по выводу на экран списка контактов. Список может быть кратким (выводится только первый телефон контакта) либо подробным (выводятся все телефоны).

В дальнейшем планируется расширять список доступных критериев поиска и перечень выводимых программой форматов списка контактов. Каким образом следует переписать Phonebook, чтобы расширение функциональности программы не приводило к необходимости внесения изменений в уже разработанные классы?

Листинг 3.4. Исходный код для задания 3.4

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace RefactorTask
{

    public class Phonebook
    {
        private List<string> Familii;
        private List<string> Imena;
        private List<List<string>> phoneNumbers;

        public Phonebook()
        {
            Familii = new List<string>();
            Imena = new List<string>();
            phoneNumbers = new List<List<string>>();
        }

     public (string, string, List<string>) FindContactByPhoneNumber(string
input)
        {
            for (int i = 0; i < phoneNumbers.Count; i++)
            {
```

```
                var n = phoneNumbers[i];
                foreach (var number in n)
                    if (input == number)
                        return (Familii[i], Imena[i], phoneNumbers[i]);
            }

            return ("Error", "Ошибка", null);
        }

        public void AddContact(string a, string b, List<string> c)
        {
            Familii.Add(a);
            Imena.Add(b);
            phoneNumbers.Add(c);
        }

        public (string, string, List<string>) FindContactByName(string d)
        {
            for (int i = 0; i < Familii.Count; i++)
            {
                if ($"{Familii[i]} {Imena[i]}" == d)
                    return (Familii[i], Imena[i], phoneNumbers[i]);
            }
            return ("Error", "Ошибка", null);
        }

        public bool PrintContactsShort(int i123 = 0)
        {
            for (int i = 0; i < Familii.Count; i++)
            {
                  Console.WriteLine($"{i}.\t{Familii[i]}\t{Imena[i]}\t{pho-
neNumbers[i][0]}");
                i123 -= 10;
            }

            return true;
        }

        public bool PrintContactsFull(int a = 10, int b = 30)
        {
            int c = -50 + a - b;
            Console.WriteLine();
            for (int i = 0; i < Familii.Count; i++)
            {
                Console.WriteLine($"{i}.\t{Familii[i]}\t{Imena[i]}");
                foreach (var phone in phoneNumbers[i])
                    Console.WriteLine(phone);
                c -= 30;
            }
```

```
                return false;
            }

        }

        public static class TestPolygonPhonebook
        {
            public static void TestPolygon()
            {
                Phonebook book = new Phonebook();
                        book.AddContact("Иванов",  "Иван",  new  List<string>()
{"+7012345678", "+7021385858"});
                        book.AddContact("Петров",  "Петр",  new  List<string>()
{"+7033333631", "+7465675651"});

                Console.WriteLine(book.FindContactByName("Петров Петр"));
               Console.WriteLine(book.FindContactByPhoneNumber("+7012345678"));

                book.PrintContactsShort();
                book.PrintContactsFull();
            }

            public static void Main(string[] args)
            {
                TestPolygon();
            }
        }
    }
```

Задание 5. Класс Notepad содержит в себе список Pages. Каждый элемент списка – это страница записной книжки, представляющая собой список строк. В отдельном массиве хранится заголовок каждой страницы.

Пользователю доступны возможности поиска, добавления строк и страниц, а также методы вывода на экран содержимого записной книжки. Возможен вывод всей информации из записной книжки, вывод конкретной страницы, вывод каждой четной страницы.

Листинг 3.5. Исходный код для задания 3.5

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace RefactorTask
{
```

```
public class Notepad
{
    private List<List<string>> data = new List<List<string>>();

    public void AddPage()
    {
        data.Add(new List<string>());
    }

    public void AddLineToPage(int nomer, string stroka)
    {
        if (nomer < data.Count)
            data[nomer].Add(stroka);
    }

    public void GetPage(int qwerty, out List<string> a)
    {
        a = data[qwerty];
    }

    public void PrintNotepad()
    {
        int m = 0;
        foreach (var page in data)
        {
            Console.WriteLine($"-----страница {m}-----");
            foreach (var line in page)
            {
                Console.WriteLine(line);
            }

            m++;
        }
    }

    public void PrintPage(int n)
    {
        Console.WriteLine($"-----страница {n}-----");
        foreach (var line in data[n])
        {

            Console.WriteLine(line);
        }
    }


}
public static class TestPolygonNotepad
{
```

```
            public static void TestPolygon()
            {
                Notepad notepad = new Notepad();
                notepad.AddPage();
                notepad.AddLineToPage(0, "День 1. Сегодня было 5 пар");
                 notepad.AddLineToPage(0, "Домашнее задание: реализовать разло-
жение экспоненты в ряд Тейлора");

                notepad.AddPage();
                notepad.AddLineToPage(1, "День 2. Сегодня было 2 пары");
                 notepad.AddLineToPage(1, "Домашнее задание: оптимизировать по-
лином Лагранжа");

                notepad.PrintNotepad();
                notepad.PrintPage(0);
            }

            public static void Main(string[] args)
            {
                TestPolygon();
            }
        }
    }
```

Задание 6. Класс Approximation позволяет по набору входных данных определить значение некоторой функции $f(x)$ в произвольной точке x. Он включает в себя различные алгоритмы интерполяции, перечень которых активно расширяется. Также он включает в себя логику по численному интегрированию функции.

Данный класс перегружен функциональностью, а добавление новых способов аппроксимации является затруднительным. Каким образом можно исправить приведенный ниже программный код?

Листинг 3.6. Исходный код для задания 3.6

```
    using System;
    using System.Linq;

    namespace RefactorTask
    {
        /// <summary>
         /// Класс принимает одномерный массив входных данных X и одномерный
массив выходных данных Y(X). На основе этих
```

```
        /// данных выполняется аппроксимация – определение значения функции в
произвольной точке.
        ///
        /// Предполагается, что inputData и outputData упорядочены таким образом,
что
        ///     1. Значения inputData отсортированы в порядке возрастания
        ///     2. inputData[i] = outputData[i] (совпадение индексов)
        ///     3. Массивы имеют одинаковый размер, размер >= 3
        ///
        /// В обновленной версии программы должна быть реализована проверка этих
условий при инциализации объекта
        /// </summary>
        public class Approximation
        {
            // Предполагается, что inputData будет отсортирован
            private double[] inputData;
            private double[] outputData;

            public Approximation(double[] input, double[] output)
            {
                // Защита от изменений данных извне
                inputData = (double[]) input.Clone();
                outputData = (double[]) output.Clone();
            }

            /// <summary>
             /// Метод ближайшего соседа. Выполняет поиск аргумента, наиболее
близкого к X. Возвращает соответствующий ему Y
            /// </summary>
            public double EvaluateNearestNeighbour(double x)
            {
                var p = inputData.OrderBy(data => Math.Abs(x - data)).First();
                var index = Array.IndexOf(inputData, p);

                return outputData[index];
            }

            /// <summary>
            /// Линейная интерполяция. Берутся две точки, ближайшие к указанной.
На получившемся отрезке функция
              /// интерполируется линией. Если X больше максимума или меньше
минимума, то возвращается max или min
            /// соответственно
            /// </summary>
            /// <param name="x"></param>
            /// <returns></returns>
            public double LinearInterpolation(double x)
            {
                if (x > inputData[^1])
                    return outputData[^1];
```

```
            else if (x < inputData[0])
                return outputData[0];

            var highPoint = inputData.First(t => t >= x);

            var highIndex = Array.IndexOf(inputData, highPoint);
            var lowIndex = highIndex - 1;

            double y1 = outputData[lowIndex];
            double y2 = outputData[highIndex];

            double x1 = inputData[lowIndex];
            double x2 = inputData[highIndex];

            double k = (y1 - y2) / (x1 - x2);
            double b = (y1 * x2 - x1 * y2) / (x2 - x1);

            return k * x + b;
        }


        /// <summary>
        /// Простой алгоритм интегрирования.
        /// </summary>
        /// <returns></returns>
        public double IntegrateSimple()
        {
            // Нужно определить среднее значение y, расстояние между началом
и концом отрезка интегрирования.
            // Их произведение даст примерную площадь фигуры

            var y = outputData.Average();
            var x = inputData[^1] - inputData[0];

            return x * y;

        }
    }

    public static class TestPolygonApproximation
    {
        public static void Main1(string[] args)
        {
            double[] input =  {0, 1, 2, 3, 4, 5};
            double[] output = {4, 5, 6, 7, 5, 4};

            Approximation approximation = new Approximation(input, output);

              Console.WriteLine($"Аппроксимация №1. f(3.5) = {approxima-
tion.EvaluateNearestNeighbour(3.5)}");
```

```
                    Console.WriteLine($"Аппроксимация №2. f(3.5) = {approxima-
tion.LinearInterpolation(3.5)}");
                        Console.WriteLine($"Интегрирование: {approximation.Inte-
grateSimple()}");
                }
            }
        }
```

Задание 7. Класс Calculator включает в себя логику по выполнению различных математических операций. Каждая математическая операция выполняет некоторую операцию над двумя аргументами и возвращает результат вычислений.

Пользователь через консоль может выбрать одну из доступных операций или завершить выполнение программы. Если пользователь выбрал операцию и ввел входные данные, на экране должен отобразиться результат вычислений.

Функциональность класса активно расширяется. Каким образом необходимо модифицировать программу, чтобы функциональность можно было расширять без внесения изменений в класс «Калькулятор»? Каким образом нужно модифицировать программу, чтобы не приходилось каждый раз вручную прописывать новую математическую операцию в пользовательском интерфейсе?

Листинг 3.7. Исходный код для задания 3.7

```
using System;

namespace RefactorTask
{
    public class Calculator
    {

        public void Add()
        {
            Console.WriteLine("Введите два числа: ");
                var t = Array.ConvertAll<string, Int32>(Console.Read-
Line().Split(' '), Int32.Parse);
            if (t.Length != 2)
```

```
                    throw new Exception("Количество аргументов отличается от
двух");

                Console.WriteLine($"Результат вычислений: {t[0] + t[1]}");

            }

            public void Sub()
            {
                Console.WriteLine("Введите два числа: ");
                    var t = Array.ConvertAll<string, Int32>(Console.Read-
Line().Split(' '), Int32.Parse);
                if (t.Length != 2)
                  throw new Exception("Количество аргументов отличается от двух");

                Console.WriteLine($"Результат вычислений: {t[0] - t[1]}");
            }

            public void Mult()
            {
                Console.WriteLine("Введите два числа: ");
                    var t = Array.ConvertAll<string, Int32>(Console.Read-
Line().Split(' '), Int32.Parse);
                if (t.Length != 2)
                  throw new Exception("Количество аргументов отличается от двух");

                Console.WriteLine($"Результат вычислений: {t[0] * t[1]}");
            }

            public void Div()
            {
                Console.WriteLine("Введите два числа: ");
                    var t = Array.ConvertAll<string, Int32>(Console.Read-
Line().Split(' '), Int32.Parse);
                if (t.Length != 2)
                  throw new Exception("Количество аргументов отличается от двух");

                Console.WriteLine($"Результат вычислений: {t[0] / t[1]}");

            }

            public void Factorial()
            {
                Console.WriteLine("Введите одно целое положительное число: ");
                int num = 1;
                    var t = Array.ConvertAll<string, Int32>(Console.Read-
Line().Split(' '), Int32.Parse);
                if (t.Length != 1 || t[0] < 1)
                    throw new Exception("Количество аргументов отличается от
одного");
```

```
        num = t[0];

        int res = 1;

        for (int i = 2; i <= num; i++)
            res *= i;

        Console.WriteLine($"Факториал числа: {res}");
    }
}


public static class TestPolygonCalculator
{
    public static void TestPolygon()
    {
        Calculator calculator = new Calculator();

        while (true)
        {

            try
            {
                Console.WriteLine("Доступные операции:");
                Console.WriteLine("+ Сложение");
                Console.WriteLine("- Вычитание");
                Console.WriteLine("* Умножение");
                Console.WriteLine("/ Деление");
                Console.WriteLine("! Факториал");
                Console.WriteLine("Exit Выход");


                var input = Console.ReadLine();

                if (input == "+")
                    calculator.Add();
                else if (input == "-")
                    calculator.Sub();
                else if (input == "*")
                    calculator.Mult();
                else if (input == "/")
                    calculator.Div();
                else if (input == "!")
                    calculator.Factorial();
                else if (input == "Exit")
                    break;
                else
                    Console.WriteLine("Неизвестная команда");

            }
```

```
                catch (Exception e)
                {
                    Console.WriteLine(e);
                }
            }
        }

        public static void Main(string[] args)
        {
            TestPolygon();
        }
    }
}
```

Задание 8. Рассматривается подмодуль САПР, позволяющий моделировать стены здания. Стена определяется двумя крайними точками, материалом и толщиной. Помимо построения стены по двум точкам, возможно построение стены как копии другой стены со смещением, построение через начальную точку и полярные координаты. Функциональность САПР активно расширяется, поэтому в дальнейшем предполагается увеличивать количество доступных пользователю способов построения стен.

Каким образом можно вынести за пределы класса логику создания новых экземпляров?

Листинг 3.8. Исходный код для задания 3.8

```
using System;
using System.ComponentModel;

namespace RefactorTask
{
    public struct Point
    {
        public double X, Y;
        public Point(double x, double y)
        {
            X = x;
            Y = y;
        }

        public override string ToString()
        {
```

```
            return $"x = {X}, y = {Y}";
        }
    }

    public class Wall
    {
        public readonly Point startPoint;
        public readonly Point endPoint;

        public Wall(Point start, Point end)
        {
            startPoint = start;
            endPoint = end;
        }

        public Wall(Wall wall, Point offset)
        {
                startPoint = new Point(wall.startPoint.X + offset.X,
wall.startPoint.Y + offset.Y);
            endPoint = new Point(wall.endPoint.X + offset.X, wall.endPoint.Y
+ offset.Y);
        }

        public Wall(Point start, double distance, double angle)
        {
            startPoint = start;
              endPoint = new Point(start.X + distance * Math.Cos(angle),
start.Y + distance * Math.Sin(angle));
        }

        public override string ToString()
        {
            return $"P1: {startPoint}, P2: {endPoint}";
        }

    }
    public class TestPolygonTask3
    {
        public static void TestPolygon()
        {
            Wall w1 = new Wall(new Point(0, 0), new Point(0, 10) );
            Wall w2 = new Wall(w1, new Point(5, 0));
            Wall w3 = new Wall(new Point(5, 5), 5, Math.PI/2.0);

            Console.WriteLine(w1);
            Console.WriteLine(w2);
            Console.WriteLine(w3);
        }
        public static void Main(string[] args)
        {
```

```
            TestPolygon();
        }

    }
}
```

Задание 9. Класс WeatherForecast позволяет выполнить прогноз погоды на основе имеющихся метеорологических наблюдений за предыдущие дни. Количество методов прогноза постоянно расширяется. Количество возможных вариантов прогноза также может увеличиваться.

Дополнительно, данный класс позволяет выводить на экран прогноз погоды на следующий день, на неделю и на месяц. Отчеты отличаются степенью подробности. Каким образом можно усовершенствовать код этого класса?

Листинг 3.9. Исходный код для задания 3.9

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace RefactorTask
{
    public enum Weather
    {
        Sun, Clouds, Rain, Snow
    }

    public class WeatherForecast
    {
        private List<Weather> Data = new List<Weather>();

        public WeatherForecast()
        {
            Data = new List<Weather>();
        }
        // Приватный конструктор, используемый для создания клонов объекта
        private WeatherForecast(List<Weather> data)
        {
            Data = data.ToList();
        }
        private WeatherForecast Clone()
        {
```

```
            return new WeatherForecast(Data);
        }

        public void AddData(Weather data)
        {
            Data.Add(data);
        }

        public Weather SimpleForecast()
        {
            return Data.Last();
        }

        public Weather AnotherSimpleForecast()
        {
            if (Data[^1] == Weather.Clouds)
            {
                if (Data[^2] == Weather.Rain)
                    return Weather.Sun;
                else if (Data[^2] == Weather.Snow)
                    return Weather.Snow;
            }

            if (Data[^2] == Data[^1])
                return Weather.Sun;

            return Data.Last();
        }

        public Weather AdvancedForecast()
        {
            return Data.GroupBy(i => i)
                    .OrderByDescending(g => g.Count())
                    .First().First();
        }

        public void PrintResultsForTomorrow()
        {
            Console.WriteLine(
                $"Прогноз погоды на завтра: {SimpleForecast()} или {Anoth-
erSimpleForecast()}, скорее всего {AdvancedForecast()}");
        }
        public void PrintResultsForNextWeek()
        {
            var futureForecaster = Clone();
            for (int i = 1; i <= 7; i++)
            {
                var res = futureForecaster.AdvancedForecast();
                Console.WriteLine($"День {i}: {res}");
```

```
                futureForecaster.AddData(res);
            }
        }

        public void PrintResultsForNextMonth()
        {
            var futureForecaster = Clone();
            for (int j = 1; j <= 4; j++)
            {
                for (int i = 1; i <= 7; i++)
                {
                    var res = futureForecaster.AnotherSimpleForecast();
                    Console.WriteLine($"Неделя {j}, день {i}: {res}");
                    futureForecaster.AddData(res);
                }
                Console.WriteLine();
            }
        }
    }

    public static class TestPolygonTask4
    {
        public static void TestPolygon()
        {
            WeatherForecast forecast = new WeatherForecast();

            forecast.AddData(Weather.Rain);
            forecast.AddData(Weather.Clouds);
            forecast.AddData(Weather.Sun);
            forecast.AddData(Weather.Clouds);
            forecast.AddData(Weather.Snow);
            forecast.AddData(Weather.Clouds);

            forecast.PrintResultsForTomorrow();
            forecast.PrintResultsForNextMonth();
        }

        public static void Main(string[] args)
        {
            TestPolygon();
        }
    }
}
```

Задание 10. Класс Figure включает в себя данные о некоторой геометрической фигуре. У каждой геометрической фигуры есть координаты центра фигуры на двухмерной плоскости, а также методы для вычисления площади и периметра.

В текущем виде добавление нового типа фигуры или изменение программной логики является проблематичным. Каким образом можно переписать данную программу?

Листинг 3.10. Исходный код для задания 3.10

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace RefactorTask
{
    public class Figure
    {
        public float X, Y;

        // Если фигура - квадрат
        // Data1 = ширина, Data2 = длина

        // Если фигура - треугольник
        // Data1 = высота
        // Data2 = основание

        // Если фигура - круг
        // Data1 = радиус
        // Data2 = диаметр
        public float Data1, Data2, Data3, Data4, Data5;

        // В зависимости от значения Type определяется тип фигуры
        // 0 = Прямоугольный треугольник
        // 1 = Квадрат
        // 2 = Круг
        public int Type;

        public Figure(float x, float y, float type, float d1, float d2 = 0,
float d3 = 0, float d4 = 0)
        {
            X = x;
            Y = y;
            Type = (int)type;
```

```
            Data1 = d1;
            Data2 = d2;
            Data3 = d3;
            Data4 = d4;
        }


        public float Perimiter()
        {
            float res = 0;
            switch(Type)
            {
                case 0:
                    res = 2*(Data1+Data2);
                    break;
                case 1:
                                                    res    =    Data1    +    Data2    +
(float)Math.Sqrt(Data1*Data1+Data2*Data2);
                    break;
                case 2:
                    res = 2 * (float)Math.PI * Data1;
                    break;
            }

            return res;
        }

        public float Area()
        {
            float res = 0;
            switch(Type)
            {
                case 0:
                    res = Data1 * Data2;
                    break;
                case 1:
                    res = 0.5f * Data1 * Data2;
                    break;
                case 2:
                    res = (float)Math.PI * Data1 * Data1;
                    break;
            }
            return res;
        }

    }

    public static class TestPolygonTask10
    {
        public static void TestPolygon()
```

```
            {
                var rectangle = new Figure(5, 6, 0, 2, 4);
                var triangle = new Figure(1, 1, 1, 2, 3);
                var circle = new Figure(1, 3, 2, 1);

                        Console.WriteLine($"{rectangle.Perimiter()}   {rectan-
gle.Area()}");
                Console.WriteLine($"{triangle.Perimiter()} {triangle.Area()}");
                Console.WriteLine($"{circle.Perimiter()} {circle.Area()}");

            }

            public static void Main(string[] args)
            {
                TestPolygon();
            }
        }
    }
```

Задание 4. Написание кода в соответствии с принципами SOLID

В рамках данного задания учащийся должен самостоятельно разработать всю программу, руководствуясь ее словесным описанием. При этом он также должен руководствоваться принципами SOLID и учитывать возможность последующего расширения функциональности программы.

Задание 1 – электрооборудование. Электроприбор имеет уникальный идентификатор и наименование. Он может быть включен или выключен. Когда электроприбор «включается» или «выключается», на экране выводится соответствующее сообщение. Необходимые для реализации в рамках данного задания электроприборы: люстра, холодильник, телевизор.

Для включения электроприбора используются переключатели. Если переключатель находится в активном положении, оборудование должно быть включено, иначе – выключено. Один переключатель может управлять несколькими электроприборами.

Задание 2 – университет. В базе данных университета хранятся данные о студентах и сотрудниках. У студентов и сотрудников есть как общие данные (ФИО, год рождения), так и уникальные (размер заработной платы и стаж работы у сотрудников, номер группы и факультет у студентов).

Абстрактная система генерации отчета университета способна по запросу сгенерировать строку-отчет, включающую в себя некоторые данные о студентах и/или сотрудниках. Наследники класса реализуют различные варианты формирования отчета. Обязательные для реализации отчеты:

- Все сотрудники, отсортированные по алфавиту.
- Все студенты, отсортированные по алфавиту.
- Студенты и сотрудники, отсортированные по году рождения.

Задание 3 – файловая система. Файловая система состоит из элементов файловой системы – файлов и каталогов. У любого элемента файловой системы есть наименование. Для файла и каталога возможно определение его размера.

Размер каталога равен размеру всех элементов, входящих в этот каталог. Размер файла может быть изменен через соответствующий открытый метод.

Для взаимодействия с файловой системой используется файловый обозреватель. В рамках данного задания можно считать файловый обозреватель специальным классом, включающим в себя методы добавления и удаления файлов и каталогов. Также файловый обозреватель включает в себя метод для вывода содержимого каталога на экран.

Примечание: наличие в файловом обозревателе методов для добавления/удаления/отображения в данном случае не считается нарушением SRP, т. к. можно считать единственной обязанностью обозревателя взаимодействие с файловой системой.

При желании можно разбить на отдельные классы-компоненты логику отображения на экране содержимого каталога и логику взаимодействия с файловой системой.

Задание 4 – операционная система. Операционная система состоит из множества приложений. У каждого приложения есть наименование и программная логика, выполняемая при запуске. В рамках задания необходимо реализовать приложение-калькулятор (пользователь вводит операцию и два числа, приложение выводит результат) и приложение-часы (приложение выводит текущее время с точностью до секунд).

Операционная система ожидает ввода наименования какого-либо из приложений. Если приложение с таким наименованием существует, оно должно быть запущено. Если такое приложение отсутствует, то на экране появляется соответствующее сообщение.

Задание 5 – графический редактор. Существует абстрактная геометрическая фигура. Подклассами геометрической фигуры являются прямая линия, квадрат, прямоугольник, круг, эллипс.

- Прямая линия описывается координатами начальной и конечной точек.
- Квадрат описывается координатами центра и размером стороны.
- Прямоугольник описывается координатами центра, шириной и высотой.
- Круг описывается координатами центра и радиусом.
- Эллипс описывается координатами центра и двумя полуосями.

Документ состоит из множества геометрических фигур. Также документ имеет наименование и максимальную ширину с высотой. Документ включает в себя методы для добавления и удаления геометрических фигур с проверкой корректности их местоположения.

Документ включает в себя метод для удаления объекта по координатам. Этот метод принимает в качестве входных данных координаты некоторой точки. Если указанные координаты находятся внутри или на границе геометрической фигуры, то эта фигура удаляется из документа.

Задание 5. Паттерны проектирования

В рамках данного задания необходимо внести корректировки в уже написанный программный код с целью его совершенствования. Задания составлены таким образом, чтобы в рамках их выполнения одним из наиболее эффективных способов улучшения программного кода являлось применение одного из классических паттернов объектно-ориентированного проектирования.

Задание 1. В объекте-микросхеме хранится произвольная «программа», информация о версии прошивки и дате изготовления микросхемы. «Программа» микросхемы является закрытым полем, и получить к ней доступ извне невозможно.

Расширьте функциональность класса «Микросхема» таким образом, чтобы его экземпляры можно было копировать. Изменять модификаторы доступа полей класса запрещается.

Листинг 5.1. Исходный код для задания 5.1

```
using System;

namespace DesignPatterns.Task4
{
    class Microchip
    {
        private string program;
        private DateTime creationTime;
        private uint version;

       public Microchip(string program, DateTime creationTime, uint version)
        {
            this.program = program;
            this.creationTime = creationTime;
            this.version = version;
        }
    }

    class TestPolygon_1
    {
        public static void MainMethod(string[] args)
        {
```

```
                var chip = new Microchip("return a + b;", DateTime.Now, 1);
                var chipNo2 = new Microchip("for (int i = 0; i < 10; i++) cout
<< i;", DateTime.Today, 2);
            }

        }
    }
```

Задание 2. Система журналирования событий используется практически в каждом модуле разрабатываемой программы. Из-за отсутствия глобальных переменных в C#, ссылку на экземпляр системы журналирования приходится передавать в каждый использующий эту систему объект, что неудобно.

Каким образом нужно изменить систему журналирования, чтобы решить эту проблему?

Примечание: для выполнения данного задания нельзя использовать статические классы, но разрешается использовать статические поля, свойства и/или методы.

Листинг 5.2. Исходный код для задания 5.2

```
using System;
using System.Drawing;
using System.Text;

namespace DesignPatterns.Task4
{
    class Logger
    {
        private double startTime = 0;
        public Logger()
        {
            startTime = DateTime.Now.TimeOfDay.TotalMilliseconds;
        }

        private double GetCurrentTime()
        {
            return DateTime.Now.TimeOfDay.TotalMilliseconds - startTime;
        }
        public void PrintInfo(string str)
        {
                Console.WriteLine($"{GetCurrentTime():0.####}  Информация:
{str}", Color.White);
        }
```

```
        public void PrintWarning(string str)
        {
            Console.WriteLine($"{GetCurrentTime():0.####} Предупреждение:
{str}", Color.Orange);
        }

        public void PrintError(string str)
        {
            Console.WriteLine($"{GetCurrentTime():0.####} Ошибка: {str}",
Color.Red);
        }
    }

    class Calculator
    {
        public static int AddTwoNumbers(int a, int b, Logger logger)
        {
            logger.PrintInfo("Начинается вычислительный процесс");
            int t = a + b;
            logger.PrintInfo("Вычисления завершены");
            return t;
        }

        public static int DivideTwoNumbers(int a, int b, Logger logger)
        {
            logger.PrintInfo("Начинается вычислительный процесс");
            if (b == 0)
            {
                logger.PrintError("Деление на ноль");
                return int.MaxValue;
            }
            int t = a / b;
            logger.PrintInfo("Вычисления завершены");
            return t;
        }
    }

    class StringGenerator
    {
        public int Amount;
        public string Substring;

        public string Generate(Logger logger)
        {
            logger.PrintInfo("Начинается генерация строки");

            if (Amount <= 0)
                logger.PrintWarning("Будет сгенерирована пустая строка");

            StringBuilder builder = new StringBuilder();
```

```
            for (int i = 0; i < Amount; i++)
                builder.Append(Substring);

            logger.PrintInfo("Генерация строки успешно завершена");

            return builder.ToString();
        }
    }

    public static class TestPolygon_2
    {
        public static void MainMethod(string[] args)
        {
            Logger logger = new Logger();
            Calculator.AddTwoNumbers(2, 2, logger);
            Calculator.DivideTwoNumbers(5, 0, logger);

            StringGenerator generator = new StringGenerator();
            generator.Amount = 5;
            generator.Substring = "qwerty";
            generator.Generate(logger);
        }
    }

}
```

Задание 3. Система автоматизированного проектирования зданий и сооружений позволяет моделировать основные элементы здания: стены, перекрытия, окна и т. д. Многие из полей этих классов являются закрытыми и недоступны для чтения.

Необходимо реализовать возможность копирования элементов здания, при этом нельзя изменять модификаторы доступа полей класса.

Листинг 5.3. Исходный код для задания 5.3

```
namespace DesignPatterns.Task4
{
    class ModelElement
    {

    }

    class Wall : ModelElement
    {
        public float Height;
```

```
        public float Thickness;
        public float Length;

        private int material;
        private string place;


        public Wall(float height, float thickness, float length, int mate-
rial, string place)
        {
            Height = height;
            Thickness = thickness;
            Length = length;
            this.material = material;
            this.place = place;
        }
    }

    class Floor : ModelElement
    {
        public float Width;
        public float Height;
        public float Thickness;

        private float price;
        private bool isAnalytical;

       public Floor(float width, float height, float thickness, float price,
bool isAnalytical)
        {
            Width = width;
            Height = height;
            Thickness = thickness;
            this.price = price;
            this.isAnalytical = isAnalytical;
        }
    }

    class Window : ModelElement
    {
        public float Width;
        public float Height;

        private string modelName;
        private int catalogNumber;

          public Window(float width, float height, string modelName, int
catalogNumber)
        {
            Width = width;
```

```
            Height = height;
            this.modelName = modelName;
            this.catalogNumber = catalogNumber;
        }
    }

    public static class TestPolygon_3
    {
        public static void MainMethod(string[] args)
        {
            Wall wall = new Wall(2700, 200, 3000, 4, "Гостиная");
            Window window = new Window(1500, 1500, "Распашное", 4);
            Floor floor = new Floor(5000, 6000, 500, 12345, false);
        }
    }
}
```

Задание 4. Изначально, для авторизации в банковской системе было достаточно указать пин-код банковской карты. Затем был добавлен дополнительный уровень безопасности: необходимо также указать номер банковской карты. В последнем обновлении была также добавлена проверка по паролю интернет-банка.

Становится очевидным, что количество дополнительных этапов проверки постоянно расширяется и оформлять их в виде вложенных условных операторов неэффективно. Каким образом следует переработать программу, чтобы процедура авторизации пользователя была более гибкой и позволяла эффективно добавлять новые способы проверки пользователя?

Листинг 5.4. Исходный код для задания 5.4

```
using System;

namespace RiderProj
{
    public class BankSystem
    {
        public bool AutorizeUser(User user)
        {
            Console.WriteLine("Введите пин-код: ");
            var pinCode = Console.ReadLine();

            if (pinCode != user.pinCode)
```

```
                return false;

            Console.WriteLine("Введите номер банковской карты: ");
            var cardNumber = Console.ReadLine();

            if (cardNumber != user.cardNumber)
                return false;

            // Обновление

            Console.WriteLine("Введите пароль интернет-банка: ");
            var pass = Console.ReadLine();

            if (pass != user.password)
                return false;

            return true;
        }
    }

    public class User
    {
        public string cardNumber;
        public string password;
        public string pinCode;

        public User(string cardNumber, string password, string pinCode)
        {
            this.cardNumber = cardNumber;
            this.password = password;
            this.pinCode = pinCode;
        }
    }

    public class TestPolygon_4
    {
        public static void MainMethod(string[] args)
        {
            BankSystem system = new BankSystem();
            User Ivan = new User("124-356-789", "Qwerty123", "12345");

            if (system.AutorizeUser(Ivan))
                Console.WriteLine("Авторизация прошла успешно");
            else
                Console.WriteLine("Пользователь ввел некорректные данные");
        }
    }
}
```

Задание 5. Для прохождения поста охраны в некоторой компании сотрудника просят предъявить электронный пропуск. Если у сотрудника нет при себе электронного пропуска, его просят показать бумажный пропуск. Если при себе у него нет и бумажного пропуска, то ему необходимо предъявить паспорт. Если у сотрудника нет и паспорта, то его нельзя пропустить в организацию.

Каким образом следует изменить структуру программы, чтобы можно было легко добавлять новые способы проверки сотрудников?

Листинг 5.5. Исходный код для задания 5.5

```
using System;

namespace RiderProj.Task4
{
    public class Security
    {
        public bool CheckPerson(Person person)
        {
             // Электронный пропуск имеется, номер пропуска соответствует
сотруднику
            if (person.ElectronicPass != null && person.ElectronicPass.Per-
sonId == person.Id)
                return true;
            // Имеется старый пропуск, ID соответствует сотруднику
             else if (person.OldPass != null && person.OldPass.PersonId ==
person.Id)
                return true;
            //  Имеется паспорт. ФИО совпадает с ФИО сотрудника
             else if (person.Passport != null && person.FIO == person.Pass-
port.FIO)
                return true;
            else
                return false;
        }
    }

    public class Person
    {
        public int Id;
        public ElectronicPass ElectronicPass;
        public Pass OldPass;
        public string FIO;
        public Passport Passport;
    }
```

```
    public class Pass
    {
        public int PersonId;

        public Pass(Person p)
        {
            PersonId = p.Id;
        }
    }

    public class ElectronicPass : Pass
    {
        public ElectronicPass(Person p) : base(p) { }
    }

    public class Passport
    {
        public string FIO;
        public int Series;
        public int Number;

        public Passport(Person p, int s, int n)
        {
            FIO = p.FIO;
            Series = s;
            Number = n;
        }
    }

    public class TestPolygon_5
    {
        public static void MainMethod(string[] args)
        {
            var security = new Security();

            Person ivan = new Person();
            ivan.FIO = "Иванов Иван Иванович";
            ivan.Passport = new Passport(ivan, 1234, 567890);
            ivan.ElectronicPass = new ElectronicPass(ivan);
            ivan.OldPass = new Pass(ivan);

            Person petr = new Person();
            petr.FIO = "Петров Петр Петрович";

            Console.WriteLine(security.CheckPerson(ivan));
            Console.WriteLine(security.CheckPerson(petr));

        }
    }
}
```

Задание 6. Для анализа данных о пользователях необходимо реализовать систему генерации отчетов. Отчеты могут генерироваться для отображения в веб-браузере (формат HTML) или в виде TXT-документа. Возможно, в дальнейшем перечень возможных форматов отображения отчета будет расширяться.

В текущей реализации присутствует повторение кода. Каким образом можно улучшить текущую реализацию программы и при этом предусмотреть возможность добавления новых форматов отчета?

Листинг 5.6. Исходный код для задания 5.6

```
using System;
using System.Collections.Generic;
using System.Text;

namespace RiderProj.Task4
{
    public class ReportGenerator
    {
        public static void MainMethod(string[] args)
        {
            List<User> users = new List<User>();

            users.Add(new User("Петров", "Петр", "Петрович", "Москва", 22));
            users.Add(new User("Иванов", "Иван", "Иванович", "СПб", 28));

            string reportType = "HTML"; // = "TXT"

            StringBuilder builder = new StringBuilder();

            if (reportType == "TXT")
            {
                builder.Append("Отчет по пользователям\n");

                builder.Append("Фамилия\t\tИмя\t\tОтчество\t\tГород\t\t
Возраст\n");

                foreach (var user in users)
                {
                    builder.Append($"{user.SecondName}\t\t");
                    builder.Append($"{user.FirstName}\t\t");
                    builder.Append($"{user.Patronymic}\t\t");
                    builder.Append($"{user.City}\t\t");
                    builder.Append($"{user.Age}");
```

```
            builder.Append("\n");
        }

        builder.Append("\n\n");
        builder.Append($"Отчет от {DateTime.Now}");
    }
    else if (reportType == "HTML")
    {
        builder.Append("<h1>Отчет по пользователям</h1>\n");

        builder.Append("<tr>");

        builder.Append("<th>Фамилия</th>");
        builder.Append("<th>Имя</th>");
        builder.Append("<th>Отчество</th>");
        builder.Append("<th>Город</th>");
        builder.Append("<th>Возраст</th>");

        builder.Append("</tr>\n");

        foreach (var user in users)
        {
            builder.Append("<tr>\n");

            builder.Append($"<td>{user.SecondName}</td>");
            builder.Append($"<td>{user.FirstName}</td>");
            builder.Append($"<td>{user.Patronymic}</td>");
            builder.Append($"<td>{user.City}</td>");
            builder.Append($"<td>{user.Age}</td>\n");
            builder.Append("</tr>\n");
        }

        builder.Append($"<p>Отчет от {DateTime.Now}</p>");
    }
    else
        builder.Append("Неизвестный формат отчета");

    var result = builder.ToString();
    Console.WriteLine(result);
}



public class User
{
    public string SecondName;
    public string FirstName;
    public string Patronymic;

    public string City;
```

```
                public int Age;

                 public User(string secondName, string firstName, string patro-
nymic, string city, int age)
                {
                    SecondName = secondName;
                    FirstName = firstName;
                    Patronymic = patronymic;
                    City = city;
                    Age = age;
                }
            }

        }
    }
```

Задание 7. Функциональность первой версии рассматриваемой программы сводилась к тому, чтобы при нажатии на некоторую абстрактную кнопку в консоли выводилось соответствующее сообщение.

Затем возникла необходимость записывать информацию о каждом нажатии на кнопку в журнал событий. После этого обнаружилось, что информацию о нажатии на кнопку необходимо учитывать в модуле подсчета износа кнопки. При появлении каждой новой функциональности приходилось вносить изменение в класс кнопки. При этом удаление из проекта любого из элементов, использующих кнопку, потенциально приведет к неработоспособности приложения (т. к. в классе-кнопке сохранится ссылка на экземпляр класса, который не определен в пространстве имен). Каким образом можно решить данную проблему?

Листинг 5.7. Исходный код для задания 5.7

```
using System;
using System.Collections.Generic;
using System.Text.Encodings.Web;

namespace RiderProj.Task4
{
    public class Screen
    {
        public void PrintMessage()
        {
```

```
            Console.WriteLine("Кнопка нажата");
        }
    }

    public static class EventLogger
    {
        public static List<DateTime> pressMoments = new List<DateTime>();

        public static void HandleButtonPress()
        {
            pressMoments.Add(DateTime.Now);
        }
    }

    public class ButtonResourceController
    {
        private uint maxButtonResource = 10;
        private int amountOfPresses;

        public ButtonResourceController(uint buttonResource = 5)
        {
            maxButtonResource = buttonResource;
            amountOfPresses = 0;
        }

        public void HandlePress()
        {
            amountOfPresses += 1;

            if (amountOfPresses > maxButtonResource)
               Console.WriteLine("Вероятно, кнопка скоро выйдет из строя");
        }
    }

    public class UniversalButton
    {
        private Screen screen;
        private ButtonResourceController controller;

        public UniversalButton(Screen screen, ButtonResourceController con-
troller)
        {
            this.screen = screen;
            this.controller = controller;
        }

        public void Press()
        {
            screen.PrintMessage();
            EventLogger.HandleButtonPress();
```

```
            controller.HandlePress();
        }

    }


    public static class TestPolygon_7
    {
        public static void MainMethod(string[] args)
        {
            var screen = new Screen();
            var controller = new ButtonResourceController();
            var button = new UniversalButton(screen, controller);

            for (int i = 0; i < 10; i++)
                button.Press();
        }
    }
}
```

Задание 8. В средстве создания электронных чертежей для отображения каждой геометрической фигуры используются экземпляры наследников класса GeometricShape. В приложении все графические элементы хранятся в виде списка GeometricShape.

Возникла необходимость в реализации новой функциональности: отображении на экране списка, в котором будут указаны тип каждой фигуры и ее площадь. В то же время код иерархии геометрических фигур отлажен, протестирован и закрыт для редактирования.

Разработчики иерархии классов предполагали, что может возникнуть необходимость в реализации дополнительной функциональности. Поэтому они реализовали в программе паттерн, позволяющий определить новую операцию, не изменяя классы этих объектов.

Каким образом можно расширить функциональность средства создания электронных чертежей?

Листинг 5.8. Исходный код для задания 5.8

```
using System.Collections.Generic;
namespace RiderProj.Task4
{

    public struct Point
    {
        private float X, Y;

        public Point(float x, float y)
        {
            X = x;
            Y = y;
        }
    }
    public abstract class GeometricShape
    {
        public Point Position;

        public void MoveTo(Point newPoint)
        {
            Position = newPoint;
        }

        public abstract void Accept(Visitor visitor);
    }

    public class Line : GeometricShape
    {
        public float Length;
        public Point Direction;

        public Line(Point p1, Point direction, float length)
        {
            Position = p1;
            Direction = direction;
            Length = length;
        }
        public override void Accept(Visitor visitor)
        {
            visitor.VisitLine(this);
        }
    }

    public class Rectangle : GeometricShape
    {
        public Point SecondCorner;
```

```
    public Rectangle(Point p1, Point p2)
    {
        Position = p1;
        SecondCorner = p2;
    }

    public override void Accept(Visitor visitor)
    {
        visitor.VisitRectangle(this);
    }

}

public class Triangle : GeometricShape
{
    public Point Position2, Position3;

    public Triangle(Point p1, Point p2, Point p3)
    {
        Position = p1;
        Position2 = p2;
        Position3 = p3;
    }

    public override void Accept(Visitor visitor)
    {
        visitor.VisitTriangle(this);
    }
}

public class Drawing
{
    private List<GeometricShape> objects = new List<GeometricShape>();

    public void Add(GeometricShape shape)
    {
        objects.Add(shape);
    }

    public void Accept(Visitor visitor)
    {
        foreach (var element in objects)
            element.Accept(visitor);
    }
}

public abstract class Visitor
{
    public abstract void VisitLine(GeometricShape shape);
    public abstract void VisitRectangle(GeometricShape shape);
```

```
        public abstract void VisitTriangle(GeometricShape shape);
    }


    public static class TestPolygon_8
    {
        public static void MainMethod(string[] args)
        {
            var drawing = new Drawing();
            drawing.Add(new Line(
                new Point(0, 0),
                new Point(1, 0),
                3.14f));

            drawing.Add(new Rectangle(
                new Point(0, 5),
                new Point(-5, 7)));
            drawing.Add(new Triangle(
                new Point(0, 0),
                new Point(0, 1),
                new Point(1, 0)));

            // Возможный результат вызова метода для вывода площади:
            //
            // 1. Линия. S = 0
            // 2. Прямоугольник. S = 60
            // 3. Треугольник. S = 0.5

        }
    }
}
```

Задание 9. Пользовательский интерфейс приложения может быть построен путем использования классических или современных элементов пользовательского интерфейса. Как в классическом, так и в современном UI есть базовые элементы интерфейса: кнопки, текстовые поля, переключатели, диалоговые окна и т. д.

Пользователь выбирает стиль пользовательского интерфейса при запуске приложения. Далее на основе его выбора весь пользовательский интерфейс строится в соответствующем стиле.

Процесс построения пользовательского интерфейса в классическом и современном стилях практически идентичен. По этой причине настройка элементов пользовательского интерфейса содержит большое количество повторяющегося кода. Каким образом можно улучшить структуру программы?

Листинг 5.9. Исходный код для задания 5.9

```
using System;
using System.Collections.Generic;

namespace RiderProj.Task4
{
    public abstract class ClassicUIElement
    {
        public int PosX, PosY;
        public int Width, Height;

       protected ClassicUIElement(int posX, int posY, int width, int height)
        {
            PosX = posX;
            PosY = posY;
            Width = width;
            Height = height;
        }
    }

    public class ClassicWindow
    {
        public string Header;
        public int Width, Height;

         private List<ClassicUIElement> elements = new List<ClassicUIEle-
ment>();


        public void MoveWindow()
        {
            Console.WriteLine("Окно следует за курсором мыши");
        }

        public void AddElement(ClassicUIElement element)
        {
            elements.Add(element);
        }
    }
```

```
        public class ClassicButton : ClassicUIElement
        {
            public string Text;
            public int BackgroundColor;

             public ClassicButton(int posX, int posY, int width, int height,
string text, int backgroundColor) : base(posX, posY, width, height)
            {
                Text = text;
                BackgroundColor = backgroundColor;
            }

            public void Click()
            {
                Console.WriteLine("При нажатии, кнопка незначительно изменилась
в цвете");
            }
        }

        public class ClassicTextField : ClassicUIElement
        {
            public string PlaceHolder;

             public ClassicTextField(int posX, int posY, int width, int height,
string placeHolder) : base(posX, posY, width, height)
            {
                PlaceHolder = placeHolder;
            }

            public void TypeText()
            {
                Console.WriteLine("Каждый набранный символ сразу отображается в
текстовом поле");
            }
        }

        public abstract class ModernUIElement
        {
            public int PosX, PosY;
            public int Width, Height;

            protected ModernUIElement(int posX, int posY, int width, int height)
            {
                PosX = posX;
                PosY = posY;
                Width = width;
                Height = height;
            }
        }
```

```
        public class ModernWindow
        {
            public string Header;
            public int Width, Height;

            public List<ModernUIElement> listOfElements = new List<ModernUIEle-
ment>();


            public void MoveWindow()
            {
                Console.WriteLine("Окно плавно скользит за курсором мыши, уско-
ряясь и замедляясь около краев экрана");
            }

            public void AddElement(ModernUIElement element)
            {
                listOfElements.Add(element);
            }
        }

        public class ModernButton : ModernUIElement
        {
            public string Text;
            public int BackgroundColor;
            public int AnimationSpeed;

           public ModernButton(int posX, int posY, int width, int height, string
text, int backgroundColor) : base(posX, posY, width, height)
            {
                Text = text;
                BackgroundColor = backgroundColor;
            }

            public void Click()
            {
                 Console.WriteLine($"При нажатии, кнопка начинает вращаться со
скоростью {AnimationSpeed} оборотов в минуту");
            }
        }

        public class ModernTextField : ModernUIElement
        {
            public string PlaceHolder;

             public ModernTextField(int posX, int posY, int width, int height,
string placeHolder) : base(posX, posY, width, height)
            {
                PlaceHolder = placeHolder;
            }
```

```
            public void TypeText()
            {
                Console.WriteLine("Каждый набранный символ появляется как трех-
мерный объект, а затем постепенно выравнивается и становится двухмерным");
            }
        }

        public static class TestPolygon_9
        {
            public static void MainMethod(string[] args)
            {
                bool newUi = false; // true

                if (newUi)
                {
                    ModernWindow window = new ModernWindow();
                    ModernButton b1 = new ModernButton(0, 0, 50, 50, "OK", 0);
                   ModernButton b2 = new ModernButton(0, 100, 50, 50, "Отмена",
0);

                     ModernTextField t1 = new ModernTextField(100, 0, 100, 50,
"Введите имя");
                     ModernTextField t2 = new ModernTextField(100, 0, 100, 50,
"Введите фамилию");

                    window.AddElement(b1);
                    window.AddElement(b2);
                    window.AddElement(t1);
                    window.AddElement(t2);
                }
                else
                {
                    ClassicWindow window = new ClassicWindow();
                   ClassicButton b1 = new ClassicButton(0, 0, 50, 50, "OK", 0);
                    ClassicButton b2 = new ClassicButton(0, 100, 50, 50, "Отмена", 0);

                    ClassicTextField t1 = new ClassicTextField(100, 0, 100, 50,
"Введите имя");
                    ClassicTextField t2 = new ClassicTextField(100, 0, 100, 50,
"Введите фамилию");

                    window.AddElement(b1);
                    window.AddElement(b2);
                    window.AddElement(t1);
                    window.AddElement(t2);
                }
            }
        }

    }
```

Заключение

В рамках данной работы приведен набор заданий и пояснений к ним, выполнение которых позволит учащимся повысить свои навыки программирования в очень важном направлении развития – повышении читаемости и организованности программного кода. В то же время, в отличие от таких направлений, как освоение новых языков программирования или изучение алгоритмов и структур данных, для которых существует большое количество упражнений и заданий на закрепление, для проектирования программного обеспечения и организации программного кода сборников упражнений подобного рода крайне малочисленны.

Данное учебно-методическое пособие предназначено для решения этой проблемы. Последовательно выполняя приведенные в них задания, учащиеся могут постепенно формировать в себе навыки, которые на первый взгляд не должны требовать такой же усердной тренировки, как алгоритмы и структуры данных или изучение синтаксиса новых языков программирования, но при этом крайне востребованы в процессе работы над сложными проектами.

Список использованной литературы

1. Макконнелл С. Совершенный код. Мастер-класс: практическое руководство по разработке программного обеспечения / С. Макконнелл; пер. с англ. В. Г. Вшивцева. – Москва: Русская редакция, 2010. – 896 с.

2. Мартин Р. Чистый код: создание, анализ и рефакторинг. Библиотека программиста / Р. Мартин; пер. с англ. Е. Матвеева. – Санкт-Петербург: Питер, 2013. – 464 с.

3. Фаулер М. Рефакторинг: улучшение дизайна существующего кода / М. Фаулер; пер. с англ. – Санкт-Петербург: Питер, 2021. – 448 с.

4. Мартин Р. Принципы, паттерны и методики гибкой разработки на языке C# / Р. Мартин, М. Мартин; пер. с англ. – Санкт-Петербург: Символ-Плюс, 2011. – 768 с.

5. Паттерны объектно-ориентированного проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес; пер. с англ. – Санкт-Петербург: Питер, 2021. – 448 с.

6. Бек К. Экстремальное программирование: разработка через тестирование / К. Бек; пер. с англ. – Санкт-Петербург: Питер, 2021. – 192 с.

7. Хориков В. Принципы юнит-тестирования / В. Хориков. – Санкт-Петербург: Питер, 2021. – 320 с.

Учебное издание

Згода Юрий Николаевич

Проектирование программного обеспечения

Учебно-методическое пособие

Издательство «Наукоемкие технологии»
ООО «Корпорация «Интел Групп»
https://publishing.intelgr.com
E-mail: publishing@intelgr.com
Тел.: +7 (812) 945-50-63
Интернет-магазин издательства
https://shop.intelgr.com/

Подписано в печать 16.08.2024.
Формат 60x84/16
Объем 4,5625 п.л.
Тираж 100 экз.

ISBN 978-5-907804-73-9