Шаблоны проектирования

Лабораторная работа 1. Принципы объектно-ориентированного проектирования

Лабораторная работа №1

Принципы объектно-ориентированного проектирования

(6 часов)

Задание 1:

Для приложения своего варианта постройте диаграмму классов, указав на ней все поля и методы и их области видимости, а также связи между классами.

Задание 2:

Для своего варианта на основе построенной диаграммы классов реализуйте приложение на языке программирования Java.

Варианты:

1. Приложение, которое читает данные из файла о координатах и размерах различных графических примитивов (отрезок, прямоугольник, эллипс, полигон и т.д.), их цвете (контура и заливки) и номере слоя. Далее приложение должно строить изображение, которое состоит из этих примитивов. При этом следует помнить, что хранение информации о графическом объекте и его рисование на форме – две отдельные задачи. Для разделения задач хранения необходимой информации и рисование примитивов можно использовать механизм обобщении (Generic).

   Структура приложения должна соответствовать принципу единственной ответственности.

   Примечание:

   Для изображений, состоящих из нескольких графических объектов, которые могут перекрываться друг с другом, используется понятие слоя. Слой содержит один объект – это аналог прозрачной кальки, на которой нарисован этот объект. Такие кальки накладываются друг на друга. При этом слои нумеруются, минимальный номер слоя соответствует либо нижнему слою, либо верхнему, по выбору разработчика. При выводе изображения на экран сначала рисуется объект, находящийся на самом нижнем слое, затем поверх него – объект на следующем слое, и т.д. до последнего слоя.

2. Приложение, которое для одной из трех функций (по выбору пользователя):

   - $f_1(x) = e^x \cos x$
   - $f_2(x) = (x^2 + 3x - 4)^{-1}$
   - $f_3(x) = \ln \sqrt{x}$

   вычисляет определенный интеграл на выбранном пользователем отрезке, при этом вычисление должно осуществляться либо по формуле левых прямоугольников, либо по формуле правых прямоугольников, либо по формуле трапеций (также по выбору пользователя). При этом вычисления необходимо производить с точностью до четырех знаков после запятой. Точность контролировать пересчетом с увеличением вдвое количества частей разбиения отрезка интегрирования.

   Структура приложения должна соответствовать принципу открытости/закрытости.

   Примечание:

   И для формул прямоугольников, и для формул трапеций отрезок интегрирования разбивается на n частей (количество частей разбиения выбирается произвольным образом, например равным 10), далее рассчитываются частичные суммы:

   $L_n = \sum_{i=0}^{n-1} f(x_{i+1})\Delta x$ – формула левых прямоугольников,

   $P_n = \sum_{i=0}^{n-1} f(x_i)\Delta x$ – формула правых прямоугольников,

$T_n = \sum_{i=0}^{n-1} \frac{f(x_i)+f(x_{i+1})}{2}\Delta x$ – формула трапеций,

где $f(x)$ – интегрируемая функция, $\Delta x = \frac{b-a}{n}$, $x_0 = a$, $x_n = b$, $[a, b]$ – отрезок интегрирования.

Далее вычисляются суммы L_{2n}, P_{2n} или T_{2n}, затем L_{4n}, P_{4n} или T_{4n}, и так далее, пока модуль разности сумм при двух последних способах разбиения меньше 0,0001. Как видно, эти формулы имеют много общего, что приводит к одинаковым циклам для каждой формулы. Необходимо разработать такую структуру приложения, которая позволит избежать дублирования кода по организации таких циклов.

3. Консольное приложение, которое считывает из файла (или файлов, по решению разработчика) список платежей разных типов некоторой организации:
   - по счетам юридическим лицам (назначение платежа, сумма платежа, название банка, номер счета, название юридического лица, номер государственной регистрации, фамилия руководителя, имя руководителя, отчество руководителя)
   - по счетам физическим лицам (назначение платежа, сумма платежа, название банка, номер счета, фамилия, имя, отчество, серия паспорта, номер паспорта)
   - наличными физическим лицам (назначение платежа, сумма платежа, номер чека, фамилия, имя, отчество, серия паспорта, номер паспорта);

   Затем выводит в виде таблицы платежи определенного типа (при этом набор столбцов таблицы должен зависеть от типа платежа). Выводимый список платежей должен сортироваться по выбранному пользователем столбцу. Тип выводимых платежей так же определяется пользователем.

   Структура приложения должна соответствовать принципу подстановки Лисков.

   Примечание:

   Для построения таблицы в консоли можно использовать псевдографику (специальные символы для имитации границ таблицы). Данный функционал реализуется достаточно сложно, поэтому алгоритм визуализации необходимо отделить от хранения данных. Это можно сделать, реализовав в каждом классе, хранящем необходимые данные, возможность перебора всех его полей, например, используя карты отображений. В качестве ключей карты можно хранить название столбца, в качестве значений – значение, возвращаемое соответствующим методом, преобразованное в строку. Кстати, в один столбец можно свести данные нескольких полей, например три поля: фамилия, имя, отчество – можно свести в один столбец с фамилией и инициалами.

4. Приложение, позволяющее пользователю вводить две последовательности чисел, одна из которых размещается в бинарное дерево, вторая в стек. Далее пользователю выводится содержимое дерева и стека, после чего из сформированного стека (из вершины) по одному извлекаются элементы, и те из них, которые присутствуют в дереве, добавляются в новый стек. После чего пользователю выводится содержимое нового стека. Дерево и стек должны быть реализованы самостоятельно, ввод и вывод элементов в/из дерева/стека должны осуществляться единообразно.

   Структура приложения должна соответствовать принципу изоляции интерфейсов.

   Примечание:

   Для того чтобы ввод или вывод значений из стека и дерева осуществлялся единообразно, необходимо реализовать единообразие перебора элементов стека и дерева и единообразие добавление элемента в стек и дерево. Сама реализация стека и дерева может быть любой (по выбору разработчика), однако следует реализовать такую структуру в приложении, чтобы минимальными изменениями можно было добавить новую реализацию стека или дерева.

5. Приложение, управляющее балансом абонентов мобильной телефонной сети. В отдельном файле должна храниться информация об абонентах: телефонный номер абонента, баланс счета, тарифный план. В отдельном файле находятся данные о совершенных абонентом звонках, отосланных SMS-сообщениях и установленных соединений с сетью Интернет. Необходимо считать данные о совершенных операциях и рассчитать новый баланс абонента. Все номера записываются в формате XXXXX-YY-ZZZZZZZ, где XXXXX – код страны, YY – код оператора мобильной связи или код города в стационарной телефонной сети, ZZZZZZZ – сам телефонный номер абонента. Все телефонные звонки на номера, у которых первые 5 цифр отличны от 00375 тарифицируются как роуминг (считаем, что звонки доступны в любую страну по одинаковым тарифам). Все звонки на номера, начинающиеся с 00375-55, тарифицируются как внутрисетевые. Все звонки на номера, начинающиеся с 00375-25, 00375-29, 00375-33, 00375-44, тарифицируются как звонки в другие мобильные сети. Остальные номера тарифицируются как звонки на стационарную сеть. В приложении реализовать поддержку следующих тарифных планов (все тарифные планы не имеют абонентской платы):
   - «план А», звонки на номера из этой же сети 25 руб./мин., звонки другим мобильным операторам 125 руб./мин., звонки на стационарную сеть 95 руб./мин., роуминг 2500 руб./мин., тарификация по 10 сек., SMS 150 руб., SMS в роуминге 1500 руб., Интернет 3000 руб./час.
   - «план Б», звонки на номера из этой же сети 5 руб./мин. (первая минута) и 50 руб./мин. (последующие), звонки другим мобильным операторам 75 руб./мин. (первая минута) и 500 руб./мин. (последующие), звонки на стационарную сеть 50 руб./мин. (первая минута) и 250 руб./мин. (последующие), роуминг 5000 руб./мин., тарификация по 10 сек., тарификация при роуминге по 1 мин., SMS 150 руб., SMS в роуминге 1000 руб., Интернет 1000 руб./Мб за первых 50 Мб и 1250 руб./Мб за все последующие.

- «план В», звонки на номера из этой же сети 100 руб./мин. (первая минута) и 5 руб./мин. (последующие), звонки другим мобильным операторам 100 руб./мин. (первая минута) и 25 руб./мин. (последующие), звонки на стационарную сеть 100 руб./мин. (первая минута) и 25 руб./мин. (последующие), роуминг 1000 руб./мин., тарификация по 1 мин., SMS 150 руб., SMS в роуминге 150 руб., Интернет 1000 руб./Мб за первых 50 Мб и 1250 руб./Мб за все последующие.
- «план Г», звонки на номера из этой же сети 500 руб./мин. (первая минута) и 100 руб./мин. (последующие), звонки другим мобильным операторам 500 руб./мин. (первая минута) и 100 руб./мин. (последующие), звонки на стационарную сеть 500 руб./мин. (первая минута) и 100 руб./мин. (последующие), роуминг 500 руб./мин., тарификация по 10 сек., SMS 150 руб., SMS в роуминге 5000 руб., Интернет 500 руб./Мб за первых 100 Мб, 1000 руб./Мб при трафике от 100 Мб до 500 Мб, 5000 руб./Мб за все последующие.

Структура приложения должна соответствовать принципу инверсии зависимостей.

Примечание:

В файле со звонками, SMS и подключению к сети Интернет должны быть только сведения о том, кто и куда позвонил (отправил SMS), дата и время начала звонка (подключения к сети Интернет), дата и время окончания звонка (подключения к сети Интернет), объем переданных данных. Тарификация (учет стоимости операции) выполняет приложение в зависимости от тарифного плана вызывающего абонента. При этом структура приложения должна позволять легко добавлять новый тарифный план. Также должна быть возможность относительно легко добавить новую услугу (отправка MMS, выполнение платного USSD запроса и т.д.).

6. Приложение, читающее из входного файла в формате CSV, данные о продажах товаров (название товара, стоимость одной единицы; количество проданных штук), и записывающее в выходной файл (также в формате CSV) данные о выручке по каждому товару (название товара; объем выручки), отсортированные по убыванию выручки.

   Структура приложения должна соответствовать принципу единственной ответственности.

   Примечание:

   Формат CSV (Comma Separated Values) – формат текстовых файлов для хранения таблиц. Каждая строка файла (последовательность символов, разделенная символами перехода строки и возврата каретки) соответствует одной строке таблицы. Ячейки в пределах одной строки таблицы разделяются символом ‘;’. Если ячейка содержит символ ‘;’, то вся строка берется в двойные кавычки, если же внутри ячейки, заключенной в кавычки, содержится символ двойной кавычки, то он удваивается. Но следует помнить, что построчное чтение и запись файла, а также обработка одной ее строки – разные задачи. Кроме того, при некорректном формате входного файла необходимо проинформировать пользователя о номере строки и столбца, в котором содержатся некорректные данные.

7. Приложение, вычисляющее корни системы линейных алгебраических уравнений методом Гаусса или Жордана-Гаусса (по выбору пользователя) с выбором главного элемента по матрице.

   Структура приложения должна соответствовать принципу открытости/закрытости.

   Примечание:

   По методу Гаусса матрица системы приводится к верхнетреугольному виду, затем последовательно выражаются все неизвестные. По методу Жордано-Гаусса матрица системы приводится к единичному виду, при этом в столбце свободных членов остается столбец решений. Но для получения нулевых значений в нужных элементах матрицы, применяются преобразования (умножение строки на ненулевое число, сложение строк), с помощью которых, используя элемент на главной диагонали, получаются нули в том же столбце. Однако, если диагональный элемент равен нулю, то выполнить такие преобразования становиться невозможным. И даже если элемент на главной диагонали не равен нулю, но очень близок к нему, из-за ограниченности разрядной сетки такая ситуация приводит к значительной вычислительной погрешности. Для минимизации этой погрешности используют выбор максимального по модулю элемента в той части матрицы, которая правее и ниже элемента на главной диагонали, а затем обменом строк и столбцов этот элемент ставиться на место диагонального. При этом обмен строк при решении систем линейных алгебраических уравнений не влияет на результат, а обмен столбцов приводит к изменению порядка значений в векторе решений. Поэтому при обмене столбцов необходимо запоминать индексы меняемых местами строк, чтобы затем в обратном порядке поменять местами значения в векторе решений. Поскольку идея выбора главного элемента по матрице одинакова для обоих методов решения системы, реализовывать этот выбор необходимо только один раз.

8. Приложение – матричный калькулятор. Приложение должно позволять работать с квадратными матрицами размера $m \times m$ (при старте приложения пользователь выбирает размер m). Калькулятор должен поддерживать четыре операции: сложение, вычитание, умножение, деление ($A/B = A \cdot B^{-1}$, где B^{-1} – обратная матрица). При вводе пользователем матрицы-операнда, он выбирает тип матрицы (диагональная; верхне- или нижнетреугольная; симметричная; обычная; константная – все элементы которой равны одному и тому же значению; константная диагональная – все диагональные элементы которой равны одному и тому же значению, а остальные – нулевые). При этом хранение элементов матриц необходимо оптимизировать по объему используемой памяти.

   Структура приложения должна соответствовать принципу подстановки Лисков.

9. Приложение, осуществляющее выполнение различных операций со счетами клиентов в некотором банке. В некотором файле хранится таблица с информацией о клиентских счетах: номер счета, баланс счета, тип клиента (физическое или юридическое лицо). Также в некотором файле хранится баланс счета самого банка, на который поступает комиссия с проводимых операций. Еще в

одном файле хранится баланс счета налоговой инспекции, на который поступает налог государству с совершаемых операций. Доступно несколько способов перечисления денег с одного счета на другой:

- перевод между двумя физическими лицами средства в объеме до 1 млн. обычным переводом без налога и комиссии;
- перевод между двумя физическими лицами средства в объеме до 10 млн. перевод с комиссией банку 2.5% от суммы перевода;
- перевод между двумя физическими лицами средства в объеме свыше 10 млн. перевод с комиссией банку 5% и налогом государству 15% от суммы перевода;
- перевод от физического лица юридическому лицу с комиссией банку 10% от суммы перевода;
- перевод от юридического лица физическому лицу с комиссией банку 2% и налогом государству 5% от суммы перевода;
- перевод между двумя юридическими лицами с налогом государству 20% от суммы перевода.

Пользователь приложения (банковский работник) выбирает номер счета отправителя платежа и получателя платежа. Приложение, в зависимости от типа счетов (физическое или юридическое лицо) и (если необходимо) от суммы платежа, выбирает нужный способ перечисления платежа, выводит информацию о проводимом платеже и выдает пользователю запрос на подтверждение действия. Если пользователь подтверждает операцию, в файл с состоянием счетов записываются соответствующие изменения, в файлы с состоянием счетов самого банка и налоговой инспекции также вносятся при необходимости соответствующие изменения, в файл истории платежей вносится информация о проведенном платеже.

Информация о проводимом платеже содержит: номера счетов отправителя и получателя, сумма платежа, комиссия банка, налог государству, дата и время операции.

Структура приложения должна соответствовать принципу изоляции интерфейсов.

Примечание:

Для каждого типа платежа, подразумевающего банковскую комиссию, комиссия должна списываться единообразно. Также и для каждого типа платежа, подразумевающего государственный налог, он должен списываться единообразно. При списывании комиссии или налога не нужно знать тип проведенного платежа, нужно уметь получить сумму комиссии или налога, и сделать соответствующие изменения в нужном файле.

10. Приложение — эмулятор простейшей вычислительной системы. Приложение должно считывать программу из текстового файла и выполнять её по шагам. Программа должна иметь следующий синтаксис.

    Каждая команда программы располагается на отдельной строке файла. Формат команды: *КОП <операнды>*. Здесь КОП - название команды, состоящее из английских букв. В зависимости от типа команды у неё может быть один или два операнда, отделённые от названия команды пробелом. Два операнда разделяются запятой. Операнды могут быть именем переменной (строка, состоящая из английских букв) или числом (вещественное число с точкой в качетсве разделителя целой и дробной части). В эмуляторе должны быть реализованы следующие команды:

    MOV операнд1,операнд2
    копирование содержимого операнд2 в операнд1, операнд1 может быть только названием переменной, при этом, если переменная не существовала, она создаётся, иначе изменяется значение существующей переменной;

    ADD операнд1,операнд2
    сложение операндов с присвоением результата в операнд1, который может быть только названием переменной;

    SUB операнд1,операнд2
    вычитание из операнд1 значения операнд2 с присвоением результата в операнд1, который может быть только названием переменной;

    MUL операнд1,операнд2
    умножение операндов с присвоением результата в операнд1, который может быть только названием переменной;

    DIV операнд1,операнд2
    деление операнд1 на операнд2 с присвоением результата в операнд1, который может быть только названием переменной;

    IN операнд
    считывает с клавиатуры число и заноси его в операнд, который должен быть названием переменной;

    OUT операнд
    выводит значение из операнд в консоль;

    CMP операнд1,операнд2
    сравнивает два операнда и запоминает результат в специальную переменную, результатом может быть -1, если операнд1 меньше операнд2; 0, если операнды равны; 1, если операнд1 больше операнд2;

    JMP операнд
    переходит к выполнению команды в строке с номером, равным операнду;

    JE операнд
    переходит к выполнению команды в строке с номером, равным операнду, если результат сравнения равен 0;

    JNE операнд
    переходит к выполнению команды в строке с номером, равным операнду, если результат сравнения не равен 0;

JG операнд
переходит к выполнению команды в строке с номером, равным операнду, если результат сравнения равен 1;

JNG операнд
переходит к выполнению команды в строке с номером, равным операнду, если результат сравнения не равен 1;

JL операнд
переходит к выполнению команды в строке с номером, равным операнду, если результат сравнения равен -1;

JNL операнд
переходит к выполнению команды в строке с номером, равным операнду, если результат сравнения не равен -1;

Приложение должно обрабатывать команды, в случае же, если команду выполнить невозможно, выполнение программы должно прерываться с выводом сообщения об ошибке.

Добавление новых команд в приложение должно происходить с минимальными изменениями в существующих классах.

Структура приложения должна удовлетворять принципу инверсии зависимости.

Указания:

В данной лабораторной работе предполагается изучение основ объектно-ориентированного проектирования, что в первую очередь предполагает использование объектно-ориентированного программирования как парадигмы, а не как возможности языка программирования. В первом случае речь идет о применении на практике принципов инкапсуляции, наследования, абстракции и полиморфизма. Во втором случае это лишь возможность объявления классов и описания в них полей и методов.

Задание каждого варианта предполагает соблюдение одного из принципов объектно-ориентированного проектирования. Напомним, различие объектно-ориентированного программирования и объектно-ориентированного проектирования заключается в том, что программирование – это инструмент, а проектирование – это метод его применения. То есть объектно-ориентированное программирование (инкапсуляция, наследование, абстракция, полиморфизм) – это лишь способ декомпозиции программного кода на классы (их поля и методы). В то время как объектно-ориентированное проектирование – это принципы, согласно которым программный код разбивается на классы определенным образом (в общем случае, такую декомпозицию можно произвести множеством способов), который обеспечивает:

- удобство командной разработки приложений;
- гибкость расширения функциональных возможностей приложения в будущем;
- легкость изменения приложения в случае изменившихся требований пользователя;
- простоту понимания структуры приложения разработчиками;

и т.д.

По разным источникам выделяются разное количество принципов объектно-ориентированного проектирования. Тем не менее, существует основные пять принципов, которые встречаются у всех авторов. Остальные принципы дополняют базовые пять принципов применительно к особенностям разработки приложений (существуют специализированные принципы для web-приложений, для настольных приложений и т.д.), к особенностям языков программирования (существуют принципы проектирования специально для Java или Small Talk). Напомним основные пять принципов.

1. Single Responsibility Principle (принцип единственной ответственности) – принцип, согласно которому должна быть только одна причина изменения класса. Этой причиной может быть только изменение требований, напрямую касающихся цели, с которой создавался этот класс. Отсюда следует, что при создании класса должна быть только одна цель, которую преследует существование класса.

   Более простыми словами у класса должна быть только одна предельно четкая обязанность. Наличие размытого круга обязанностей класса свидетельствует о нарушении данного принципа.

2. Open Closed Principle (принцип открытости / закрытости) – принцип, согласно которому класс должен быть закрыт для изменения, но открыт для расширения. То есть изменение набора полей и методов класса может производиться только в случае изменения требований, которые касаются этого класса (смотри предыдущий принцип). В иных случаях (например, при расширении функций приложения) поля и методы класса меняться не должны, так как для новых функций должны создаваться новые классы. Такая неизменяемость ранее созданных классов и называется закрытостью для изменений.

   Если новый функционал похож на ранее реализованный, то весьма вероятно, что класс для нового функционала будет похож на уже существующий. Часто в таком случае новый класс создается как копия существующего класса с незначительными изменениями, но на практике такой подход приводит к тому, что обе копии должны модифицироваться зачастую одновременно, что усложняет поддержку такого приложения. В таком случае одним из способов избавления от дублирующегося кода является применение наследования, т.е. новый класс наследуется от существующего, добавляя нужный функционал.

   Проектирование класса таким образом, чтобы от него можно было наследоваться при добавлении нового функционала, и называется открытостью класса для расширения.

3. Liskov Substitution Principle (принцип подстановки Лисков) – принцип, сформулированный Барбарой Лисков, гласит, что экземпляр подкласса может использоваться всюду, где может быть использован экземпляр суперкласса.

   На самом деле, это утверждение – одна из особенностей полиморфизма, то есть работать должно и так всегда. Однако на практике часто приходится сталкиваться с ситуациями, когда спроектированный класс используется в некотором методе только за счет того, что в этом методе известны некоторые особенности реализации используемого класса. Но когда создается новый подкласс данного класса, в котором реализация изменяется и подкласс теряет указанную особенность, то формально (согласно принципу полиморфизма) экземпляр этого подкласса можно передать в метод вместо экземпляра суперкласса, но работать корректно этот метод уже не сможет.

   Такое поведение и называют нарушением принципа подстановки Лисков. Таким образом, можно рекомендовать всегда при проектировании класса предполагать, как будет работать приложение, если появиться новый подкласс спроектированного класса.

4. Dependency Inversion Principle (принцип инверсии зависимости) – класс не должен зависеть от другого класса напрямую, он должен зависеть от него через интерфейс.

   Рассмотрим пример: в приложении необходимо считать из файла список некоторых записей по некоторому критерию (например, все книги одного автора). Далее этот список сортируется (например, по названию книги) и в нем производится двоичный поиск по запросу пользователя. Для удобства реализации этих операций создается отдельный класс для хранения списка в виде массива. Тогда классы: читающий список из файла; отсеивающий записи по нужному критерию; сортирующий список; выполняющий поиск в списке – все эти классы зависят от этого списка. В случае, если для увеличения производительности встанет задача использования списка на основе компонентов связности, придется все классы, зависящие от массива, незначительно модифицировать. Такая ситуация, особенно в крупных проектах, может привести к большим временным затратам при модификации или сопровождении приложения.

   Наличие упомянутых прямых зависимостей и является нарушением принципа инверсии зависимости. Для соблюдения данного принципа в упомянутом примере необходимо, чтобы классы чтения, фильтрации, сортировки и поиска зависили не от списка в виде массива на прямую, а от некоторого общего для всех списков интерфейса. В таком интерфейсе могут быть объявлены методы добавления и доступа к элементам списка. В классах, зависящих от списков, может храниться ссылка интерфейсного типа, который может принимать значения ссылки на экземпляр любого класса, реализующего этот интерфейс. Тогда все изменения для перевода приложения с использования одного вида списка на другой, будут касаться только создания экземпляра класса списка, все остальные действия, которые выполняются через интерфейс, изменены не будут.

5. Interface Segregation Principle (принцип изоляции интерфейса) – принцип гласит, что класс не должен зависеть от тех методов интерфейса, которые ему не нужно реализовывать. Данный принцип используется при проектировании интерфейсов (или абстрактных классов).

   Если в интерфейсе объявлено несколько методов, то любой класс, реализующий этот интерфейс, должен реализовать все эти методы, хотя не всегда такая возможность может представляться.

   Например, если говорить о тех же списках, разработчик может применить в приложении список на основе массивов, однако, предполагая, что в будущем может понадобиться изменить этот список на двунаправленный список на основе компонентов связности, можно заранее объявить интерфейс, в котором будут присутствовать методы итерирования списка в прямом и обратном направлении. Но если в будущем понадобиться создать класс для однонаправленного списка на основе компонентов связности, реализовать такой интерфейс будет затруднительно, так как итерирование однонаправленного списка в обратном направлении реализовать очень сложно, да и работать такой перебор будет чрезвычайно медленно. А это значит, что даже если эти методы и будут реализованы (с применением больших трудовых и временных затрат), то все равно их будут стараться не использовать.

   С другой стороны, можно «ненужные» методы не реализовывать (то есть оставлять некоторую пустую их реализацию, например, с генерацией некоторой исключительной ситуации). Но такой подход тоже является не совсем корректным, так как лишает приложение определенной гибкости.

   Создание «раздутых» интерфейсов является нарушением принципа изоляции интерфейса. В таких случаях рекомендуют разбивать интерфейс на несколько интерфейсов. Для приведенного примера один интерфейс может содержать методы прямого итерирования, второй – методы обратного итерирования.

В каждом варианте предложено задание на соблюдение одного из этих пяти принципов. Но не стоит забывать, что все принципы объектно-ориентированного проектирования взаимосвязаны между собой. Как правило, невозможно нарушить один из этих принципов, не нарушив остальные. Поэтому при выполнении лабораторной работы нужно знать все пять этих принципов и стараться соблюдать их все.

Состояние ответа

Состояние ответа на задание	Ни одной попытки

Состояние оценивания	Не оценено
Последнее изменение	-
Комментарии к ответу	Комментарии (0)

Добавить ответ на задание

Вы пока не предоставили ответ на задание

◄ 1. Принципы объектно-ориентированного проектирования

Перейти на...

2. Введение в шаблоны и их классификация ►